

Linux for LEON processors

LEON Linux

LEON Linux User's Manual

Table of Contents

1. Introduction	4
1.1. Conventions in this document	4
2. The LEON Linux 5.10 releases	5
2.1. Following the upstream stable Linux 5.10 branch	5
3. Configuring the Linux Kernel	6
4. Pin multiplexing	7
4.1. Mklinuxing configuration	7
4.2. GR740	7
5. Subsystems and drivers	8
5.1. CAN subsystem	8
5.1.1. Kernel configuration for CAN	8
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	10
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. Ethernet subsystem	14
5.3.1. GRETH 10/100 and GBIT driver	14
5.4. Framebuffer subsystem	15
5.4.1. Kernel configuration for framebuffer	15
5.4.2. SVGACTRL driver	15
5.5. GPIO subsystem	15
5.5.1. Kernel configuration for GPIO	15
5.5.2. GRGPIO driver	15
5.5.3. GPIO kernel space interface	16
5.6. I ² C subsystem	17
5.6.1. Kernel configuration for I ² C	17
5.6.2. I2CMST driver	17
5.6.3. I ² C user space interface	17
5.6.4. I ² C in Buildroot	17
5.7. Memory Management	18
5.7.1. Memory Models	18
5.7.2. Defer memory initialisation	18
5.8. PCI subsystem	18
5.8.1. Kernel configuration for PCI	18
5.8.2. GRPCI driver	19
5.8.3. GRPCI2 driver	19
5.9. Perf subsystem	19
5.9.1. Kernel configuration for perf	19
5.9.2. Perf in Buildroot	19
5.9.3. L4STAT driver	19
5.10. PS/2 subsystem	21
5.10.1. Kernel configuration for PS/2	21
5.10.2. APBPS2 driver	21
5.11. SPI subsystem	21
5.11.1. Kernel configuration for SPI	21
5.11.2. SPICTRL driver	21
5.11.3. Important changes between Linux 4.9 and Linux 5.10	23
5.11.4. SPI user space interface	23
5.11.5. SPI in Buildroot	24
5.12. TTY subsystem	24
5.12.1. Kernel configuration for TTY	24
5.12.2. APBUART driver	24

5.12.3. Mlinuxing configuration	25
5.13. USB device subsystem	25
5.13.1. Kernel configuration for USB device	25
5.13.2. GRUSBDC driver	25
5.14. USB host subsystem	26
5.14.1. Kernel configuration for USB host	26
5.14.2. GRUSBHC EHCI driver	26
5.14.3. GRUSBHC UHCI driver	26
5.15. Watchdog subsystem	26
5.15.1. Kernel configuration for Watchdog	27
5.15.2. GPTIMER Watchdog Driver	27
5.15.3. Early Watchdog Support	28
5.15.4. Watchdogs with short maximum hw-timeout	28
5.15.5. Watchdog in Buildroot	28
6. Frequently Asked Questions	30
6.1. What causes DMA-API warnings on addresses 0xf0005000-0xf0007fff?	30
6.2. Initialisation of unavailable ranges when using Sparse memory	30
7. Support	31

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 LIN-UXBUILD linux build environment, the mklinuxing 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 variable 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 **mklinuxing**. These xml files are used with the `-xml` option of **mklinuxing**. See the Mklinuxing manual for details.

2. The LEON Linux 5.10 releases

The LEON Linux 5.10 series of releases contains a number of patches on top of the official upstream long term stable (LTS) Linux 5.10 kernel from kernel.org.

2.1. Following the upstream stable Linux 5.10 branch

Our LEON Linux kernel releases are set up as a relatively small number of patches on top of upstream long term stable releases. Installing our standalone release sets by default up a git repository with a branch with our patches. The idea is that it should be easy for the user to fetch the latest updates from the upstream stable branch and rebase the LEON branch on top of the latest patches of the corresponding stable branch. Thus, one can get the latest fixes without necessarily having to wait for a new LEON Linux release from us.

The upstream Linux kernel stable branches can be found from `git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git`. It has a branch `linux-5.10.y` (with a literal `y`) containing the latest state of the Linux 5.10 LTS branch. Official upstream stable releases are tagged on the form `v5.10.n` where `n` is replaced by a number.

To set up for following the stable branch:

```
git remote add stable git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
```

To pull in the latest stable patches, assuming the LEON patches from the LEON Linux release are in a branch `leon-5.10-1.0`, do:

```
git fetch stable linux-5.10.y
git rebase stable/linux-5.10.y leon-5.10-1.0
```

3. Configuring the Linux Kernel

This chapter 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 5.10.2	drivers/input/serio/apbps2.c	In mainline since v3.10
APUART	Section 5.12.2	drivers/tty/serial/apuart.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 5.15	drivers/watchdog/gptimer_wdt.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 5.3.1	drivers/net/ethernet/aeroflex/greth.c	In mainline since v2.6.34
GRGPIO	Section 5.5.2	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 5.8.2	arch/sparc/kernel/leon_pci_grpci1.c	In mainline since v3.10
GRPCI2	Section 5.8.3	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 5.13.2	drivers/usb/gadget/udc/gr_udc.c	In mainline since v3.14
GRUSBHC	Section 5.14.2 & Section 5.14.3	drivers/usb/host/{e,u}hci-grlib.c	In mainline since v3.0
I2CMST	Section 5.6.2	drivers/i2c/busses/i2c-ocores.c	In mainline since v3.8
L4STAT	Section 5.9.3	drivers/perf/l4stat_pmu.c	Only in Gaisler LEON Linux release
SVGACTRL	Section 5.4.2	drivers/video/fbdev/grvga.c	In mainline since v3.2
SPICTRL	Section 5.11.2	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 overridden 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,#FFFFFFF &
# 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
<code>clk_ignore_unused</code>	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    0 50000
clkname1              1      1      0      0      0      0    0 50000
clkname2              1      1      0      0      0      0    0 50000
clkname ...
clknameN              0      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.

Mklinuxing 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. Mklinuxing configuration for GR740

The device tree additions for the clock gating unit and all connected peripherals are automatically handled for GR740 by Mklinuxing since 2.0.17.

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. Mklinuxing configuration for a custom system

For components without built-in support in Mklinuxing 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="clk0"/>
    <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="clk0"/>
    <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="clk0"/>
    <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="clk0"/>
    <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="clk0"/> <!-- 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="clk0"/> <!-- 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="clk0"/> <!-- 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. Ethernet subsystem

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

5.3.1. GRETH 10/100 and GBIT driver

5.3.1.1. 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 (described under mklinuximg configuration further down), where the module parameter takes precedence. The 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

Parameter	Default	Description
		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 mklinuxing 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.

5.3.1.2. Mklinuxing configuration

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

5.4. 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.

5.4.1. Kernel configuration for framebuffer

Enable framebuffer support in the Linux kernel by selecting:

- “Support for frame buffer devices” (FB)

under “Graphics support” under “Device drivers”.

5.4.2. 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”.

5.5. 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.

5.5.1. Kernel configuration for GPIO

Enable GPIO support in the Linux kernel by selecting:

- “GPIO Support” (GPIOLIB)

under “Device Drivers”.

5.5.2. GRGPIO driver

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

5.5.2.1. Kernel configuration

When configuring the kernel, select:

- “Aeroflex Gaisler GRGPIO support” (GPIO_GRGPIO)

under “GPIO Support”.

5.5.2.2. Mklinuxing configuration

The GRGPIO driver needs special devicetree properties. Mklinuxing from version 2.0.6 have support for generating them from explicitly specified information or from autoprobng 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.

5.5.2.3. Mklinuxing 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 mklinuxing 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 5.11.2.1.

5.5.3. 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. 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.

5.6.1. Kernel configuration for I²C

Enable I²C support in the Linux kernel by selecting:

- “I2C support” (I2C)

under “Device Drivers”

5.6.2. 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”.

5.6.2.1. Mklinuxing configuration

The I2CMST driver needs special devicetree properties. Mklinuxing 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 “compatible” property should contain the name in the `struct i2c_device_id` of the slave device driver.

The following example xml file for mklinuxing 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>dsl672</string> <!-- match with driver -->
      </add-prop>
    </add-node>
  </match-core>
</matches>
```

5.6.3. I²C user space interface

5.6.3.1. 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”.

5.6.4. I²C in Buildroot

The “i2c-tools” package contains the i2c-tools toolset (see Section 5.6.3.1). 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`.

5.7. Memory Management

5.7.1. 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 5.7.2). 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”

5.7.2. 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 5.7.1)

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. 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.

5.8.1. 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.)”

5.8.2. GRPCI driver

When configuring the kernel, select:

- “GRPCI Host Bridge Support” (SPARC_GRPCI1)

under “Bus options (PCI etc.)”

5.8.3. GRPCI2 driver

When configuring the kernel, select:

- “GRPCI2 Host Bridge Support” (SPARC_GRPCI2)

under “Bus options (PCI etc.)”

5.8.3.1. Mklinuxing 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. <ul style="list-style-type: none"> • bit0 = PCI INTA# • bit1 = PCI INTB# • bit2 = PCI INTC# • bit3 = PCI INTD#
reset	word, boolean	0=no-rst	Force PCI reset on startup. If the property value is set to non-zero the GR-PCI2 host driver will use the GRPCI2 register interface to reset the PCI bus on boot.

5.9. 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 5.9.3.3.

5.9.1. Kernel configuration for perf

Enable perf support in the Linux kernel by selecting:

- “Kernel performance events and counters” (PERF_EVENTS)

under “General setup”

5.9.2. Perf in Buildroot

Under “Linux Kernel Tools” select:

- “perf” (BR2_PACKAGE_LINUX_TOOLS_PERF)

5.9.3. 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.

5.9.3.1. Kernel configuration

To use L4STAT, select:

- “Gaisler L4STAT statistics unit support” (L4STAT_PMU)

under “Performance monitor support”.

5.9.3.2. 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

5.9.3.3. 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
```

5.10. PS/2 subsystem

5.10.1. 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”.

5.10.2. 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”.

5.11. 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.

5.11.1. Kernel configuration for SPI

Enable SPI support in the Linux kernel by selecting:

- “SPI support” (SPI)

under “Device Drivers”.

See also Section 5.5 on how to set up GPIO to be able to use GPIO for chipselects.

5.11.2. 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`.

5.11.2.1. 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 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 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>

```

5.11.3. Important changes between Linux 4.9 and Linux 5.10

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.

5.11.4. 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 specifying “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 5.11.2.1 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 5.11.2.1 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
```

5.11.4.1. 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.

5.11.4.2. 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.

5.11.5. SPI in Buildroot

The “spi-tools” package contains the spi-tools toolset (see Section 5.11.4.2). Under “Hardware handling” select:

- “spi-tools” (BR2_PACKAGE_SPI_TOOLS)

The “spidev_test” package contains the spidev_test tool (see Section 5.11.4.1). Under “Debugging, profiling and benchmark” select:

- “spidev_test” (BR2_PACKAGE_SPIDEV_TEST)

5.12. TTY subsystem

5.12.1. Kernel configuration for TTY

Enable TTY support in the Linux kernel by selecting:

- “Enable TTY” (TTY)

under “Character devices”.

5.12.2. APBUART driver

The APBUART driver follows the standard Linux TTY kernel interface.

5.12.2.1. 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.

5.12.2.2. Mklinuxing 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 amptopts property can be set in two different ways:

One way to add amptopts properties is to add it using the `-amp` flag with `mklinuxing`. 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 amptopts 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>

```

5.12.3. 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.

5.13. 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.

5.13.1. 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.

5.13.2. GRUSBDC driver

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

5.13.2.1. 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 peripheral 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”.

5.13.2.2. 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 endpoints. 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 -->
    </add-prop>
  </match-core>
</matches>

```

```

        <!-- OUT endpoint 2 has default buffer size 1024, but need no entry -->
    </add-prop>
</match-core>
</matches>

```

5.14. 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.

5.14.1. 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.

5.14.2. GRUSBHC EHCI driver

The GRUSBHC core supports 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.

5.14.3. GRUSBHC UHCI driver

The GRUSBHC core supports 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.

5.15. 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.

5.15.1. 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.

5.15.2. 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”.

5.15.2.1. 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.

5.15.3. 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 5.15.1.

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”

5.15.3.1. Early GPTIMER Watchdog Driver

The Early GPTIMER Watchdog driver supports the watchdog timers mentioned in Section 5.15.2

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”.

5.15.3.2. Mklinuximg configuration

Please refer to Section 5.15.2.

5.15.4. 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 5.7.2 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 5.15.3) 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 5.15.1.

5.15.5. 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.
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. Frequently Asked Questions

6.1. 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:

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

6.2. 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
```

7. 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

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

Copyright © 2024 Frontgrade Gaisler AB