LEON Linux User's Manual

LEON specific Linux documentation

LINLEON
Version 1.0.0
November 2014

Kungsgatan 12
413 11 Gothenburg
Sweden

tel +46 31 7758650
fax +46 31 421407
www.aeroflex.com/gaisler

LEON Linux User's Manual

Table of Contents

1. Introduction

This document contains documentation on configuring the Linux kernel, and configuring and using device drivers for Aeroflex Gaisler 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 agains 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 3 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. 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 3 for configuration of device drivers for Aeroflex Gaisler 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”.

3. 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 http://gaisler.com.

Table 3.1. Overview of Cores and Drivers

Core	Section	Linux 3.10 Source Path	Comments
APBPS2	Section 3.7.2	`drivers/input/serio/apbps2.c`	In mainline since v3.10
APBUART	Section 3.9.2	`drivers/tty/serial/apbuart.c`	In mainline since v2.6.33
CAN_OC	Section 3.1.3	`drivers/net/can/sja1000/sja1000_of_platform.c`	In mainline since v3.8
GRCAN	Section 3.1.2	`drivers/net/can/grcan.c`	In mainline since v3.8
GRETH/ GRETH_GBIT	Section 3.2.1	`drivers/net/ethernet/aeroflex/greth.c`	In mainline since v2.6.34
GRGPIO	Section 3.4.2	`drivers/gpio/gpio-grgpio.c`	In mainline since v3.10
GRHCAN	Section 3.1.2	`drivers/net/can/grcan.c`	In mainline since v3.8
GRPCI	Section 3.6.2	`arch/sparc/kernel/leon_pci_grpci1.c`	In mainline since v3.10
GRPCI2	Section 3.6.3	`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 3.10.2	`drivers/usb/gadget/gr_udc.c`	Available in leon-linux-3.10; in mainline since v3.14
GRUSBHC	Section 3.11.2 & Section 3.11.3	`drivers/usb/host/{e,u}hci-grlib.c`	In mainline since v3.0
I2CMST	Section 3.5.2	`drivers/i2c/busses/i2c-ocores.c`	In mainline since v3.8
SVGACTRL	Section 3.3.2	`drivers/video/grvga.c`	In mainline since v3.2
SPICTRL	Section 3.8.2	`drivers/spi/spi-fsl-spi.c`	In mainline since v3.10

3.1. CAN subsystem

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

3.1.1. 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)

3.1.2. 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.

3.1.2.1. 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 3.2.

Table 3.2. 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.

3.1.2.1.1. 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.

3.1.3. 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.

3.1.3.1. Kernel configuration

To use CAN_OC, select:

- "Philips/NXP SJA1000 devices" (CAN_SJA1000)
- "Generic OF Platform Bus based SJA1000 driver" (CAN_SJA1000_OF_PLATFORM)

3.1.3.2. 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.

3.1.4. CAN kernel space interface

Not applicable

3.1.5. 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.

3.1.5.1. 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.

3.1.6. 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**. Under “Network applications” select:

- “iproute2” (BR2_PACKAGE_IPROUTE2)

The “can-utils” package contains the can-utils toolset (see Section 3.1.5.1). Under “Network applications” select:

- “can-utils” (BR2_PACKAGE_CAN_UTILS)

3.2. Ethernet subsystem

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

3.2.1. GRETH 10/100 and GBIT driver

3.2.1.1. 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 3.3. 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.

3.2.1.2. 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.

3.3. 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.

3.3.1. 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".

3.3.2. 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".

3.4. 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.

3.4.1. Kernel configuration for GPIO

Enable GPIO support in the Linux kernel by selecting:

- "GPIO Support" (GPIOLIB)

under "Device Drivers".

3.4.2. GRGPIO driver

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

3.4.2.1. Kernel configuration

When configuring the kernel, select:

- "Aeroflex Gaisler GRGPIO support" (GPIO_GRGPIO)

under "GPIO Support".

3.4.2.2. 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.

3.4.2.3. 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 3.8.2.1.

3.4.3. 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.

3.5. 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.

3.5.1. Kernel configuration for I²C

Enable I²C support in the Linux kernel by selecting:

- “I2C support” (I2C)

under “Device Drivers”

3.5.2. 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”.

3.5.2.1. 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>
```

3.5.3. I²C user space interface

3.5.3.1. 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”.

3.5.4. I²C in Buildroot

The “i2c-tools” package contains the i2c-tools toolset (see Section 3.5.3.1). 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.

3.6. 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.

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.

> **Note**
> 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.

3.6.1. 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.)”

3.6.2. GRPCI driver

When configuring the kernel, select:

- “GRPCI Host Bridge Support” (SPARC_GRPCI1)

under “Bus options (PCI etc.)”

3.6.3. GRPCI2 driver

When configuring the kernel, select:

- “GRPCI2 Host Bridge Support” (SPARC_GRPCI2)

under “Bus options (PCI etc.)”

3.6.3.1. 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 3.4. 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. • 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 GRPCI2 host driver will use the GRPCI2 register interface to reset the PCI bus on boot.

3.7. PS/2 subsystem

3.7.1. 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”.

3.7.2. 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”.

3.8. 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.

3.8.1. Kernel configuration for SPI

Enable SPI support in the Linux kernel by selecting:

- “SPI support” (SPI)

under “Device Drivers”.

3.8.2. 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.

3.8.2.1. 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. 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" and "spi-lsb-first" properties that are added as empty properties. See `Documentation/devicetree/bindings/spi/spi-bus.txt` for details.

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>
```

3.9. TTY subsystem

3.9.1. Kernel configuration for TTY

Enable TTY support in the Linux kernel by selecting:

- "Enable TTY" (TTY)

under "Character devices".

3.9.2. APBUART driver

The APBUART driver follows the standard Linux TTY kernel interface.

3.9.2.1. 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.

3.9.2.2. 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>
```

3.9.3. 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.

3.10. 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.

3.10.1. 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.

3.10.2. 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.

3.10.2.1. 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".

3.10.2.2. Mklinuximg configuration

If the GRUSBDC has non-default (1024) buffer sizes for the endpoints, this needs to be specified in the devicetree 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>
```

3.11. 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.

3.11.1. Kernel configuration for USB host

Enable USB host support in the Linux kernel by selecting:

- “USB support” (USB_SUPPORT)

under “Device drivers”

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

3.11.2. 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.

3.11.3. 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.

4. Support

For Support, contact the Aeroflex Gaisler support team at support@gaisler.com.

5. Disclaimer

Aeroflex Gaisler AB, reserves the right to make changes to any products and services described herein at any time without notice. Consult Aeroflex or an authorized sales representative to verify that the information in this document is current before using this product. Aeroflex 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 Aeroflex; nor does the purchase, lease, or use of a product or service from Aeroflex convey a license under any patent rights, copyrights, trademark rights, or any other of the intellectual rights of Aeroflex or of third parties.