StarFive
赛昉科技

Software SDK Developer Guide for GPIO

VisionFive 2

Version: 1.0

Date: 2022/11/10

Doc ID: JH7110-DGEN-002

Legal Statements

Important legal notice before reading this documentation.

Contact Us

Address: Room 502, Building 2, No. 61 Shengxia Rd., China (Shanghai) Pilot Free Trade Zone, Shanghai, 201203, China

Website: http://www.starfivetech.com

Email:

- Sales: sales@starfivetech.com
- Support: support@starfivetech.com

Preface

About this guide and technical support information.

About this document

This document mainly provides the SDK developers with the programing basics and debugging know-how for the GPIO of the StarFive next generation SoC platform - JH7110.

Audience

This document mainly serves the GPIO relevant driver developers. If you are developing other modules, place a request to your sales or support consultant for our complete documentation set on JH7110.

Revision History

Table 0-1 Revision History

Version	Released	Revision
1.0		First official release.

Notes and notices

The following notes and notices might appear in this guide:

- **Tip:**
  Suggests how to apply the information in a topic or step.
- **Note:**
  Explains a special case or expands on an important point.
- **Important:**
  Points out critical information concerning a topic or step.
- **CAUTION:**
  Indicates that an action or step can cause loss of data, security problems, or performance issues.
- **Warning:**
  Indicates that an action or step can result in physical harm or cause damage to hardware.

Contents

List of Tables

List of Figures

1. Introduction

GPIO is mainly based on the pinctrl (pin control) framework.

Pinctrl framework is the Linux system framework designed to unify pin management on different SoC platforms. The framework includes the SoC providers, including StarFive from the massive and duplicated effort on creating a pin management subsystem.

1.1. Function Introduction

The StarFive JH7110 SoC platform provides a pin controller which allows developers to configure one or one group of pin functions and capabilities. The pinctrl driver from the Linux kernel can help developers with the following tasks.

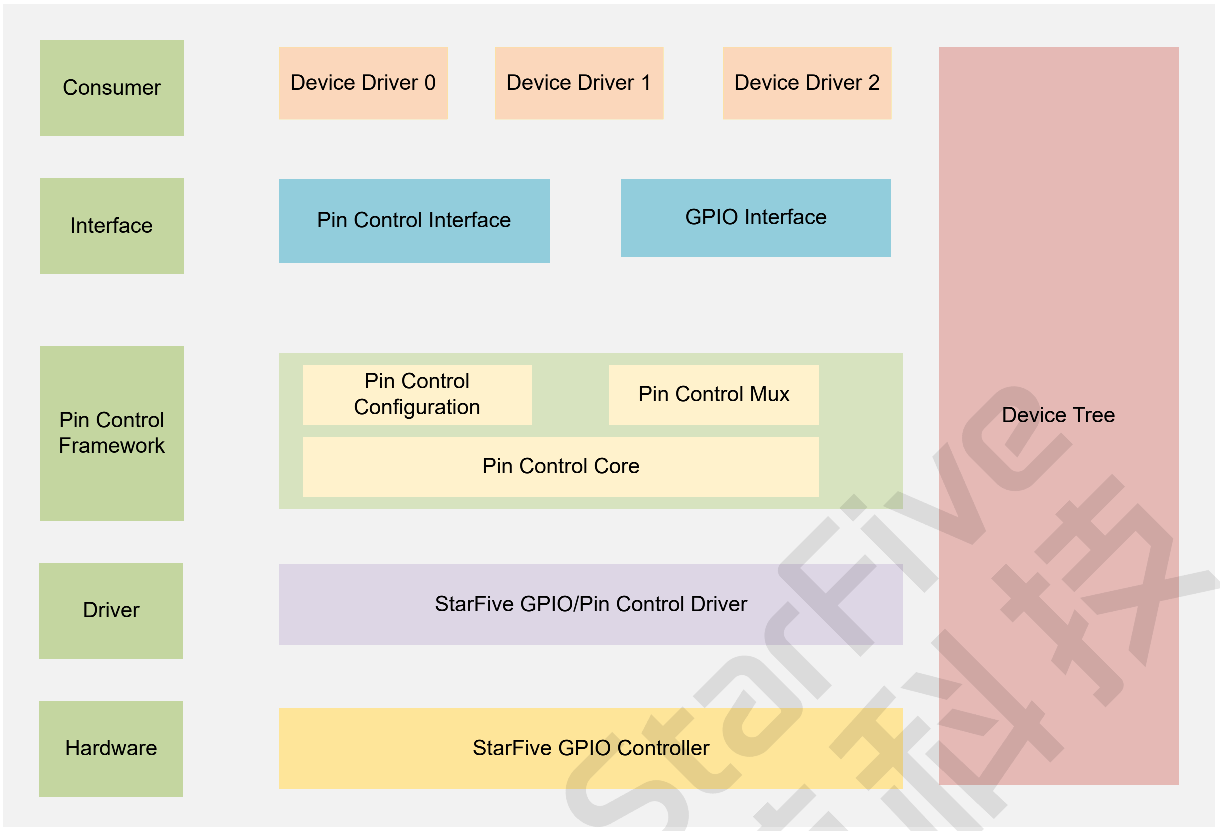
- Locate and name all the pins which the pin controller can manage.
- Provide pin multiplexing.
- Provide options for pin configuration, for example, drive strength, power up, power down, and data attribute, etc.
- Interact with the GPIO subsystem.
- Realize pin interrupt.

1.2. Block Diagram

The following figure displays the block diagram of the JH7100 GPIO pin control (pinctrl) driver module. The driver module has the following four parts:

- Pin control interfaces
- Pin control general framework
- StarFive JH7110 pin control driver
- Board level configuration

Figure 1-1 Block Diagram



The above image shows the following layers.

- **Consumer**: The pin control interfaces and GPIO interfaces used by the device driver, such as SDIO, PCIE, etc.
- **Interface**: The pin control and GPIO interfaces for user, see Interface Description *(on page 18)* for more information.
- **Pinctrl framework**: The original Linux system pin control framework. The framework enables developers to configure one or a group of pin functions and capabilities.
- **Driver**: The GPIO and pin control drivers of the StarFive JH7110 SoC platform.
- **Hardware**: The GPIO controller of the StarFive JH7110 SoC platform.

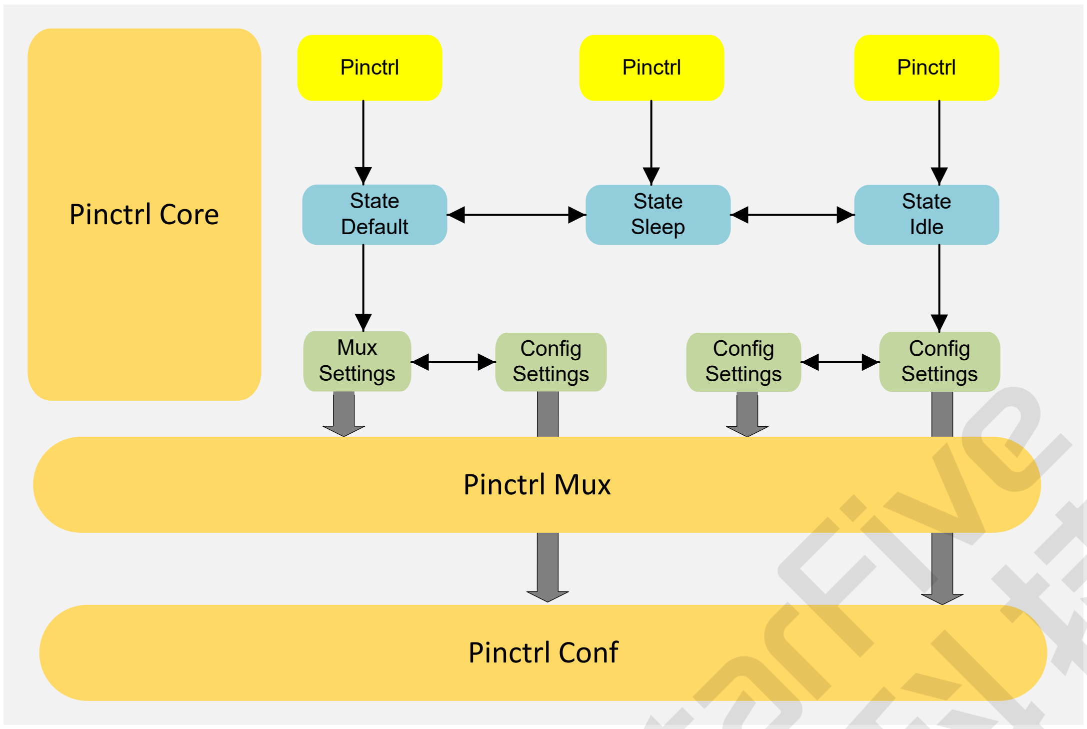
1.3. Pin Control Framework

The *Pin control (Pinctrl)* framework of the JH7110 SoC platform is mainly made up of the following three components.

- **Pinctrl Core**: The core layer of the pin control framework. The states of default, sleep or idle, refer to the power management status of pin control.
- **Pinctrol Mux**: Pin multiplexing functions.
- **Pinctrl Conf**: Pin configuration settings.

The following figure shows the relationship of them.

Figure 1-2 Pin Control Framework



Pin configuration can be different for a specific system work mode. For example, the default pin configuration works for normal mode, and the power-saving pin configuration works for standby mode. Thus you can use the above pin control framework to manage pin configuration based on the work mode of the device.

1.4. Source Code Structure

The source code structure of the GPIO is listed as follows:

```
linux
├── drivers
|   ├── pinctrl
|   |   ├── core.c
|   |   ├── core.h
|   |   ├── devicetree.c
|   |   ├── devicetree.h
|   |   ├── pinconf.c
|   |   ├── pinconf-generic.c
|   |   ├── pinconf.h
|   |   ├── pinmux.c
|   |   ├── pinmux.h
|   |   ├── starfive
|   |   |   ├── Kconfig
|   |   |   ├── Makefile
|   |   |   ├── pinctrl-starfive.c
|   |   |   ├── pinctrl-starfive.h
|   |   |   └── pinctrl-starfive-jh7110.c

├── include
|   ├── dt-bindings
|   |   ├── pinctrl
|   |   |   ├── starfive,jh7110-pinfunc.h
|   ├── linux
|   |   ├── pinctrl
|   |   |   ├── consumer.h
|   |   |   ├── devinfo.h
|   |   |   ├── machine.h
|   |   |   ├── pinconf-generic.h
|   |   |   ├── pinconf.h
|   |   |   ├── pinctrl.h
|   |   |   ├── pinctrl-state.h
|   |   |   └── pinmux.h
```

1.5. Device Tree Overview

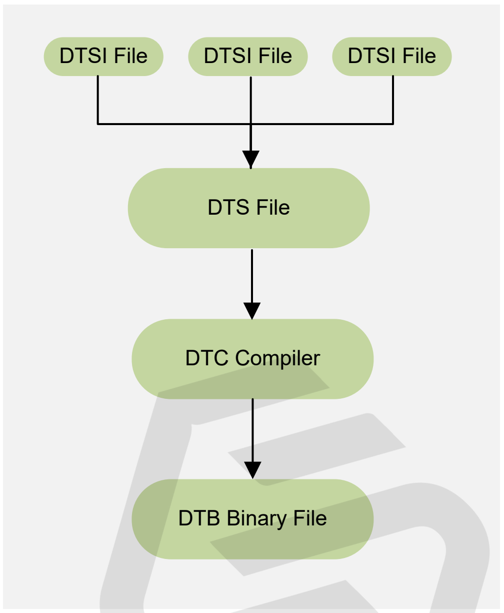
Since Linux 3.x, device tree is introduced as a data structure and language to describe hardware configuration. It is a system-readable description of hardware settings so that the operating system doesn't have to hard code details of the machine.

A device tree is primarily represented in the following forms.

- *Device Tree Compiler (DTC)*: The tool used to compile device tree into system-readable binaries.
- *Device Tree Source (DTS)*: The human-readable device tree description file. You can locate the target parameters and modify hardware configuration in this file.
- *Device Tree Source Information (DTSI)*: The human-readable header file which you can include in device tree description. You can locate the target parameters and modify hardware configuration in this file.
- *Device Tree Blob (DTB)*: The system-readable device tree binary blob files which is burned in system for execution.

The following diagram shows the relationship (workflow) of the above forms.

Figure 1-3 Device Tree Workflow



1.6. Device Tree Source Code

Overview Structure

The device tree source code of JH7110 is listed as follows:

```
linux
├── arch
|   ├── riscv
|   |   ├── boot
|   |   |   ├── dts
|   |   |   |   └── starfive
|   |   |   |       ├── codecs
|   |   |   |       |   ├── sf_pdm.dtsi
|   |   |   |       |   ├── sf_pwmdac.dtsi
|   |   |   |       |   ├── sf_spdif.dtsi
|   |   |   |       |   ├── sf_tdm.dtsi
|   |   |   |       |   └── sf_wm8960.dtsi
|   |   |   |       ├── evb-overlay
|   |   |   |       |   ├── jh7110-evb-overlay-can.dts
|   |   |   |       |   ├── jh7110-evb-overlay-rgb2hdmi.dts
|   |   |   |       |   ├── jh7110-evb-overlay-sdio.dts
|   |   |   |       |   ├── jh7110-evb-overlay-spi.dts
|   |   |   |       |   ├── jh7110-evb-overlay-uart4-emmc.dts
|   |   |   |       |   ├── jh7110-evb-overlay-uart5-pwm.dts
|   |   |   |       |   └── Makefile
|   |   |   |       ├── jh7110-clk.dtsi
|   |   |   |       ├── jh7110-common.dtsi
|   |   |   |       ├── jh7110.dtsi
|   |   |   |       ├── jh7110-evb-can-pdm-pwmdac.dts
|   |   |   |       ├── jh7110-evb.dts
|   |   |   |       ├── jh7110-evb.dtsi
|   |   |   |       ├── jh7110-evb-dvp-rgb2hdmi.dts
|   |   |   |       ├── jh7110-evb-pcie-i2s-sd.dts
|   |   |   |       ├── jh7110-evb-pinctrl.dtsi
|   |   |   |       ├── jh7110-evb-spi-uart2.dts
|   |   |   |       ├── jh7110-evb-uart1-rgb2hdmi.dts
|   |   |   |       ├── jh7110-evb-uart4-emmc-spdif.dts
|   |   |   |       ├── jh7110-evb-uart5-pwm-i2c-tdm.dts
|   |   |   |       ├── jh7110-fpga.dts
|   |   |   |       ├── jh7110-visionfive-v2.dts
|   |   |   |       ├── Makefile
|   |   |   |       └── vf2-overlay
|   |   |   |           ├── Makefile
|   |   |   |           └── vf2-overlay-uart3-i2c.dts
```

SoC Platform

The device tree source code of the JH7110 SoC platform is in the following path:

```
freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110.dtsi
```

VisionFive 2

The device tree source code of the VisionFive 2 *Single Board Computer (SBC)* is in the following path:

```
freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110-visionfive-v2.dts
-- freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110-common.dtsi
-- freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110.dtsi
```

2. Configuration

2.1. Kernel Menu Configuration

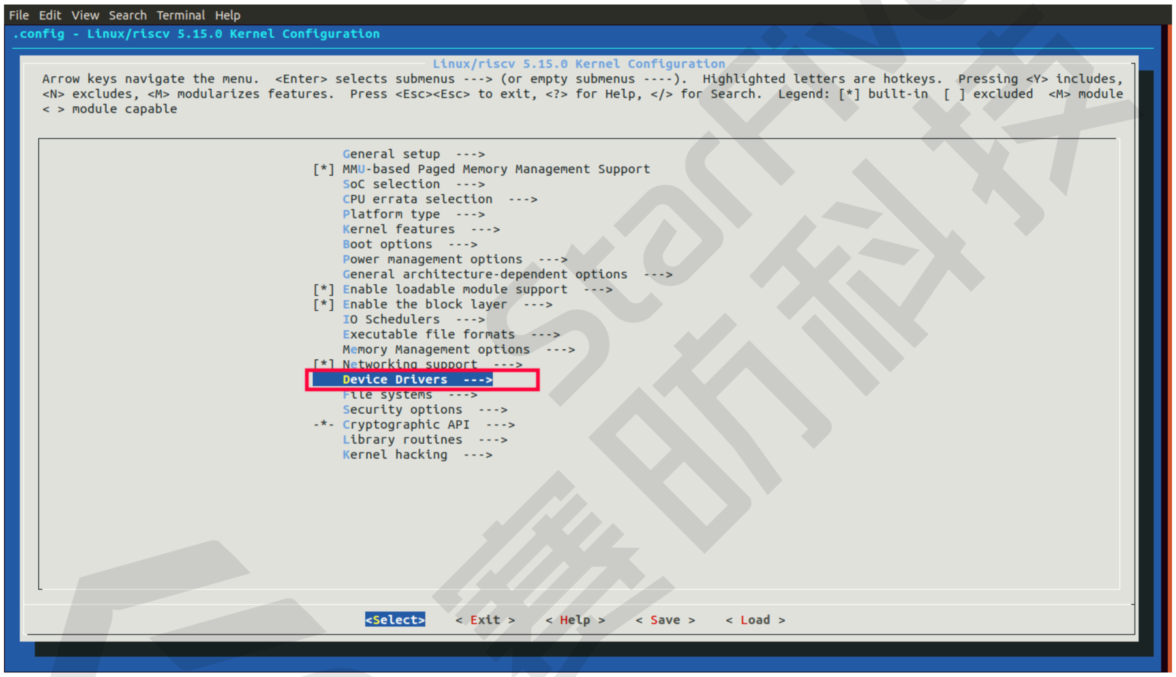
Follow the steps below to enable the kernel configuration for GPIO.

1. Under the root directory of `freelight-u-sdk`, type the following command to enter the kernel menu configuration GUI.

```
make linux-menuconfig
```

2. Enter the **Device Drivers** menu.

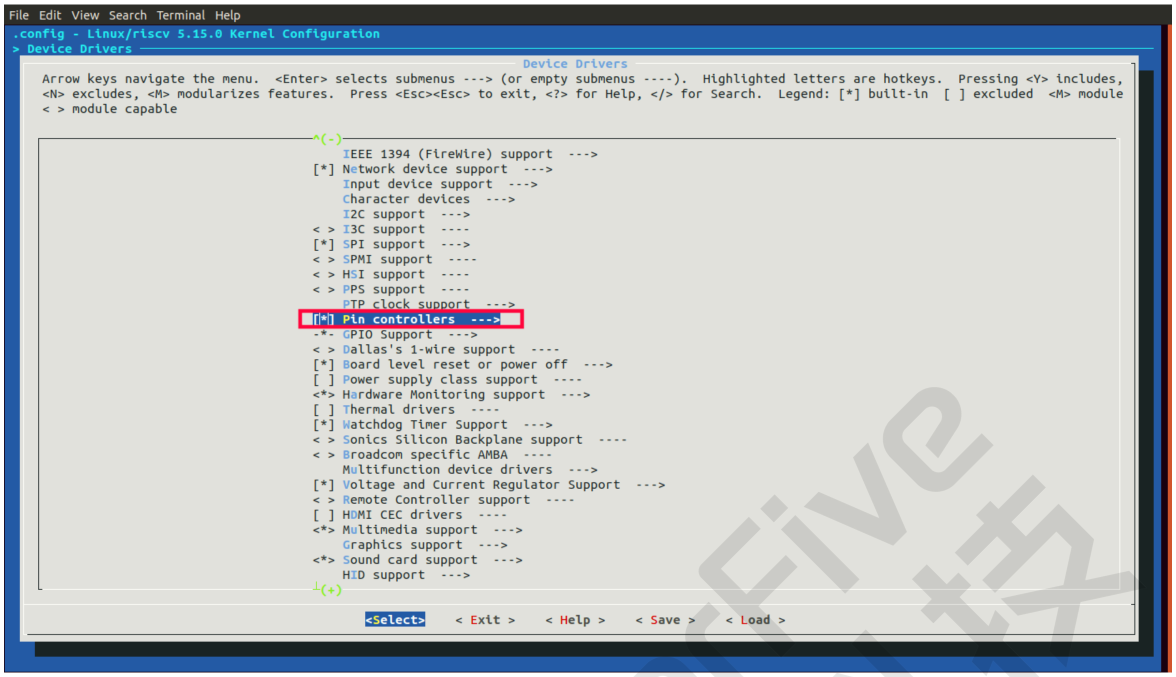
Figure 2-1 Device Drivers



```
File  Edit  View  Search  Terminal  Help
.config - Linux/riscv 5.15.0 Kernel Configuration
                         Linux/riscv 5.15.0 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> includes,
  <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module
  < > module capable

                                        General setup  --->
                                    [*] MMU-based Paged Memory Management Support
                                        SoC selection  --->
                                        CPU errata selection  --->
                                        Platform type  --->
                                        Kernel features  --->
                                        Boot options  --->
                                        Power management options  --->
                                        General architecture-dependent options  --->
                                    [*] Enable loadable module support  --->
                                    [*] Enable the block layer  --->
                                        IO Schedulers  --->
                                        Executable file formats  --->
                                        Memory Management options  --->
                                    [*] Networking support  --->
                                        Device Drivers  --->
                                        File systems  --->
                                        Security options  --->
                                    -*- Cryptographic API  --->
                                        Library routines  --->
                                        Kernel hacking  --->

                    <Select>    < Exit >    < Help >    < Save >    < Load >
```

3. Enter the **Pin controllers** menu.

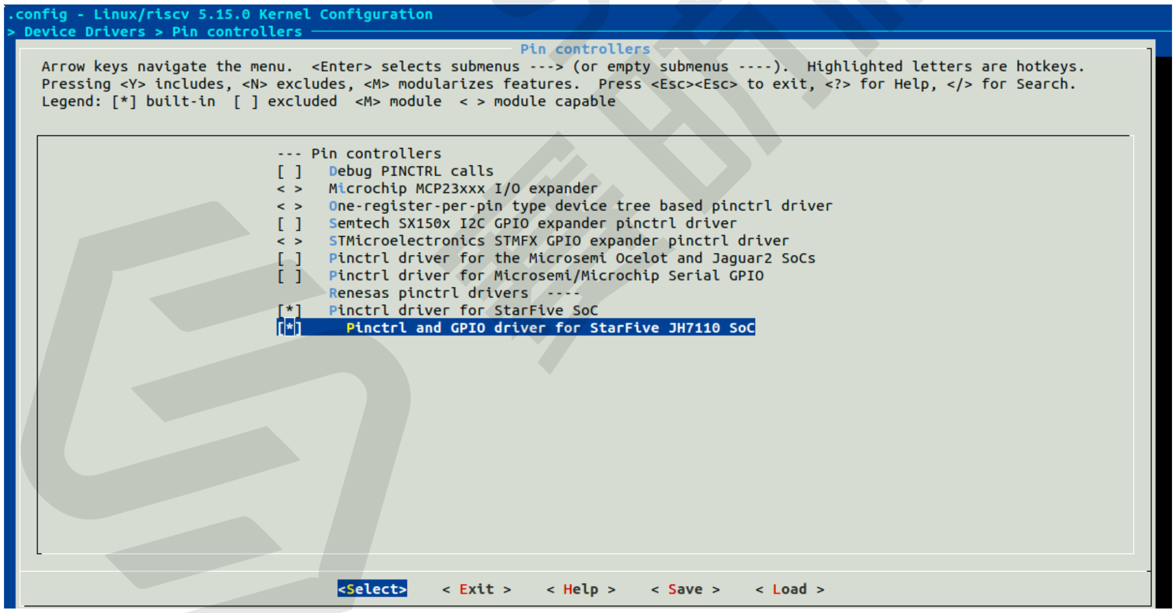
Figure 2-2 Pin Controllers



```
.config - Linux/riscv 5.15.0 Kernel Configuration
> Device Drivers
                                    Device Drivers
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> includes,
  <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module
  < > module capable

                                    ^(-)
                                        IEEE 1394 (FireWire) support  --->
                                    [*] Network device support  --->
                                        Input device support  --->
                                        Character devices  --->
                                        I2C support  --->
                                    < > I3C support  ----
                                    [*] SPI support  --->
                                    < > SPMI support  ----
                                    < > HSI support  ----
                                    < > PPS support  ----
                                        PTP clock support  --->
                                    [*] Pin controllers  --->
                                    -*- GPIO Support  --->
                                    < > Dallas's 1-wire support  ----
                                    [*] Board level reset or power off  --->
                                    [ ] Power supply class support  ----
                                    <*> Hardware Monitoring support  --->
                                    [ ] Thermal drivers  ----
                                    [*] Watchdog Timer Support  --->
                                    < > Sonics Silicon Backplane support  ----
                                    < > Broadcom specific AMBA  ----
                                        Multifunction device drivers  --->
                                    [*] Voltage and Current Regulator Support  --->
                                    < > Remote Controller support  ----
                                    [ ] HDMI CEC drivers  ----
                                    <*> Multimedia support  --->
                                        Graphics support  --->
                                    <*> Sound card support  --->
                                        HID support  --->
                                    ↓(+)

                      <Select>    < Exit >    < Help >    < Save >    < Load >
```

4. Select the **Pinctrl driver for StarFive SoC** option to enable the pinctrl driver, and select the **Pinctrl and GPIO driver for StarFive JH7110 SoC** option to enable the GPIO driver respectively.

Figure 2-3 Driver Menu



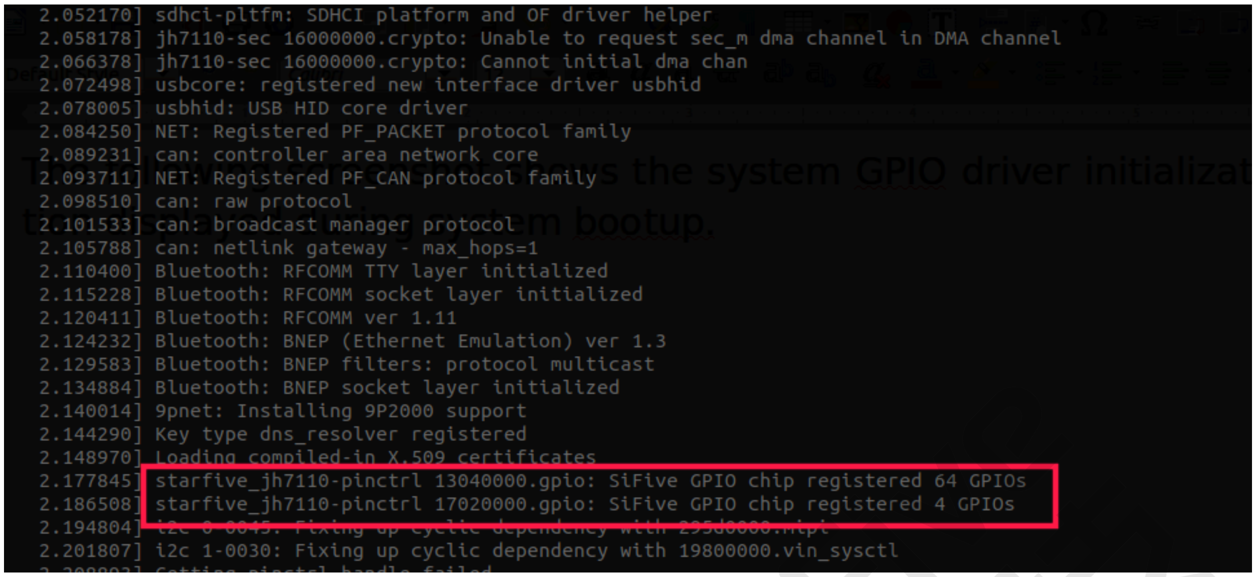
```
.config - Linux/riscv 5.15.0 Kernel Configuration
> Device Drivers > Pin controllers
                                  Pin controllers
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.
  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                             --- Pin controllers
                             [ ]   Debug PINCTRL calls
                             < >   Microchip MCP23xxx I/O expander
                             < >   One-register-per-pin type device tree based pinctrl driver
                             [ ]   Semtech SX150x I2C GPIO expander pinctrl driver
                             < >   STMicroelectronics STMFX GPIO expander pinctrl driver
                             [ ]   Pinctrl driver for the Microsemi Ocelot and Jaguar2 SoCs
                             [ ]   Pinctrl driver for Microsemi/Microchip Serial GPIO
                                   Renesas pinctrl drivers  ----
                             [*]   Pinctrl driver for StarFive SoC
                             [*]     Pinctrl and GPIO driver for StarFive JH7110 SoC

                    <Select>    < Exit >    < Help >    < Save >    < Load >
```

5. Save your change before you exit the kernel configuration dialog.

2.2. Driver Initialization

During system boot-up, when you see the following highlighted information, it means the GPIO driver has been initialized.

Figure 2-4 GPIO Driver Initialization



```
  2.052170] sdhci-pltfm: SDHCI platform and OF driver helper
  2.058178] jh7110-sec 16000000.crypto: Unable to request sec_m dma channel in DMA channel
  2.066378] jh7110-sec 16000000.crypto: Cannot initial dma chan
  2.072498] usbcore: registered new interface driver usbhid
  2.078005] usbhid: USB HID core driver
  2.084250] NET: Registered PF_PACKET protocol family
  2.089231] can: controller area network core
  2.093711] NET: Registered PF_CAN protocol family
  2.098510] can: raw protocol
  2.101533] can: broadcast manager protocol
  2.105788] can: netlink gateway - max_hops=1
  2.110400] Bluetooth: RFCOMM TTY layer initialized
  2.115228] Bluetooth: RFCOMM socket layer initialized
  2.120411] Bluetooth: RFCOMM ver 1.11
  2.124232] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
  2.129583] Bluetooth: BNEP filters: protocol multicast
  2.134884] Bluetooth: BNEP socket layer initialized
  2.140014] 9pnet: Installing 9P2000 support
  2.144290] Key type dns_resolver registered
  2.148970] Loading compiled-in X.509 certificates
  2.177845] starfive_jh7110-pinctrl 13040000.gpio: SiFive GPIO chip registered 64 GPIOs
  2.186508] starfive_jh7110-pinctrl 17020000.gpio: SiFive GPIO chip registered 4 GPIOs
  2.194804] i2c 0-0045: Fixing up cyclic dependency with 295d0000.mipi
  2.201807] i2c 1-0030: Fixing up cyclic dependency with 19800000.vin_sysctl
  2.208893] Getting pinctrl handle failed
```

2.3. Driver Validation

After system boot-up, you can run the following commands to verify if the GPIO driver is working properly.

```
# cd /sys/class/gpio/
# ls
export gpiochip0 gpiochip64 unexport
# echo 44 > export
# ls
export gpio44 gpiochip0 gpiochip64 unexport
# cd gpio44/
# ls
active_low direction subsystem value
device edge uevent
# cat direction
in
# cat value
1
```

2.4. Device Tree Configuration

The general configuration profile of the JH7110 SoC platform is in the following file:

```
linux/arch/riscv/boot/dts/starfive/jh7110.dtsi
```

Important:
DO NOT make any change to the above file.

The following code block shows the content of the configuration file.

```
        gpio: gpio@13040000 {
                compatible = "starfive,jh7110-sys-pinctrl";
                reg = <0x0 0x13040000 0x0 0x10000>;
                reg-names = "control";
                clocks = <&clkgen JH7110_SYS_IOMUX_PCLK>;
                resets = <&rstgen RSTN_U0_SYS_IOMUX_PRESETN>;
                interrupts = <86>;
                interrupt-controller;
                #gpio-cells = <2>;
                ngpios = <64>;
                status = "okay";
        };

        gpioa: gpio@17020000 {
                compatible = "starfive,jh7110-aon-pinctrl";
                reg = <0x0 0x17020000 0x0 0x10000>;
                reg-names = "control";
                resets = <&rstgen RSTN_U0_AON_IOMUX_PRESETN>;
                interrupts = <85>;
                interrupt-controller;
                #gpio-cells = <2>;
                ngpios = <4>;
                status = "okay";
        };
```

The following list provides explanations for the parameters included in the above code block.

- **compatible**: Compatibility information, used to associate the driver and its target device.
- **reg**: Register base address "`0x13040000`" and range "`0x10000`".
- **reg-names**: Names of the registers used by the GPIO module.
- **clocks**: The clocks used by the GPIO module.
- **clock-names**: The names of the above clocks.
- **resets**: The reset signals used by the GPIO module.
- **interrupts**: Hardware interrupt ID.
- **status**: The work status of the GPIO module. To enable the module, set this bit as "`okay`" or to disable the module, set this bit as "`disabled`".

Make sure you do not change the bits of **gpio-cells** and **ngpios**.

2.5. Board Level Configuration

The `board.dts` file is used to store the configuration profiles at the board level.

For the VisionFive 2 SBC, the `board.dts` file is in the following path:

```
linux/arch/riscv/boot/dts/starfive/jh7110-visionfive-v2.dts
```

Take UART0 module as an example, its `board.dts` file is in the following path:

```
linux/arch/riscv/boot/dts/starfive/jh7110-visionfive-v2.dts
```

In the file, you can find the following configuration information for UART pin control configuration:

```
&gpio {
        uart0_pins: uart0-pins {
                uart0-pins-tx {
                        sf,pins = <PAD_GPIO5>;
                        sf,pin-ioconfig = <IO(GPIO_IE(1) | GPIO_DS(3))>;
                        sf,pin-gpio-dout = <GPO_UART0_SOUT>;
                        sf,pin-gpio-doen = <OEN_LOW>;
                };

                uart0-pins-rx {
                        sf,pins = <PAD_GPIO6>;
                        sf,pinmux = <PAD_GPIO6_FUNC_SEL 0>;
                        sf,pin-ioconfig = <IO(GPIO_IE(1) | GPIO_PU(1))>;
                        sf,pin-gpio-doen = <OEN_HIGH>;
                        sf,pin-gpio-din =  <GPI_UART0_SIN>;
                };
        };
};
```

And you can also find the following configuration information for pin control.

```
&uart0 {
        pinctrl-names = "default";
        pinctrl-0 = <&uart0_pins>;
        status = "okay";
};
```

3. Interface Description

3.1. Pin Control Interfaces

The GPIO module of JH7110 has the following pin control interfaces.

3.1.1. pinctrl_get

The interface has the following parameters.

- **Synopsis**:

  ```
  struct pinctrl *pinctrl_get(struct device *dev)
  ```

- **Description**: The interface is used to load the pin operations handle from the device. All operations are based on this pin control handle.
- **Parameter**:
  - **dev**: The device to which the pin operation applies.
- **Return**:
  - **Success**: Pinctrl handle.
  - **Failure**: Error code.

3.1.2. pinctrl_put

The interface has the following parameters.

- **Synopsis**:

  ```
  void pinctrl_put(struct pinctrl *p)
  ```

- **Description**: The interface is used to discount usage on a previously claimed pin control handle. It must be used in pair with the `pinctrl_get()` function.
- **Parameter**:
  - **p**: The pinctrl handle to release.
- **Return**: None

Note:
The interface can be used only after the pinctrl_get *(on page 18)* interface has been used. Otherwise, the use is invalid.

3.1.3. devm_pinctrl_get

The interface has the following parameters.

- **Synopsis**:

  ```
  struct pinctrl *devm_pinctrl_get(struct device *dev)
  ```

- **Description**: The interface is equal to the `pinctrl_get()` (pinctrl_get *(on page 18)*) interface, but with resource management capabilities.
- **Parameter**:

  - **dev**: The device to which the pin operation applies.

- **Return**:
  - **Success**: Pin control handle.
  - **Failure**: Error code.

3.1.4. devm_pinctrl_put

The interface has the following parameters.

- **Synopsis**:

```
void devm_pinctrl_put(struct pinctrl *p)
```

- **Description**: The interface is equal to the `pinctrl_put()` function, but with resource management capabilities. You can deallocate a `struct pinctrl` obtained via `devm_pinctrl_get()`.

  Note:
  Normally, you may not need this function. Since it may release all resources.

- **Parameter**:
  - **p**: The pin control handle to release.
- **Return**: None

Note:
The interface can be used only after the devm_pinctrl_get *(on page 18)* interface has been used. Otherwise, the use is invalid.

3.1.5. pinctrl_lookup_state

The interface has the following parameters.

- **Synopsis**:

```
struct pinctrl_state *pinctrl_lookup_state(struct pinctrl *p, const char *name)
```

- **Description**: The interface is used to retrieve a state handle from a pin control handle.
- **Parameter**:
  - **p**: The pin control handle to retrieve the state.
  - **name**: The name of the state you wish to retrieve to.
- **Return**:
  - **Success**: The state handle from a pin control handle.
  - **Failure**: Error code.

3.1.6. pinctrl_select_state

The interface has the following parameters.

- **Synopsis**:

```
int pinctrl_select_state(struct pinctrl *p, struct pinctrl_state *state)
```

- **Description**: The interface is used to select, activate or program a pin control state to "HW".
- **Parameter**:

- **p**: The pin control handle for the device that requests configuration.
- **state**: The state handle to select, *activate* or *program*.

- **Return**:
  - **Success**: 0.
  - **Failure**: Error code.

3.2. GPIO Interface

The GPIO module of JH7110 has the following GPIO interfaces.

3.2.1. gpio_request

The interface has the following parameters.

- **Synopsis**:

```
int gpio_request(unsigned gpio, const char * label)
```

- **Description**: The function is used to request access to a GPIO interface.
- **Parameter**:
  - **gpio**: The GPIO index number.
  - **label**: The GPIO name. Leave it as NULL if you have no idea.
- **Return**:
  - **Success**: 0.
  - **Failure**: Error code.

3.2.2. gpio_free

The interface has the following parameters.

- **Synopsis**:

```
void gpio_free(unsigned gpio)
```

- **Description**: The function is used to release a GPIO interface.
- **Parameter**:
  - **gpio**: The GPIO index number.
- **Return**: None.

3.2.3. gpio_direction_input

The interface has the following parameters.

- **Synopsis**:

```
int gpio_direction_input(unsigned gpio)
```

- **Description**: The function is used to set a GPIO interface as input.
- **Parameter**:
  - **gpio**: The GPIO index number.

- **Return**:
    - **Success**: 0.
    - **Failure**: Error code.

3.2.4. gpio_direction_output

The interface has the following parameters.

- **Synopsis**:

```
int gpio_direction_output(unsigned gpio, int value)
```

- **Description**: The function is used to set a GPIO interface as output.
- **Parameter**:
    - **gpio**: The GPIO index number.
    - **value**: The expected level, 0 as low level, 1 as high level
- **Return**:
    - **Success**: 0.
    - **Failure**: Error code.

3.2.5. gpio_get_value

The interface has the following parameters.

- **Synopsis**:

```
int gpio_get_value(unsigned gpio)
```

- **Description**: The interface is used to get the level from a target GPIO interface. The level is used to define whether the GPIO is an input or an output.
- **Parameter**:
    - **gpio**: The GPIO index number.
- **Return**:
    - **Success**: The level of the target GPIO interface: 0 as low level, 1 as high level.
    - **Failure**: -1.

3.2.6. gpio_set_value

The interface has the following parameters.

- **Synopsis**:

```
void gpio_set_value(unsigned gpio, int value)
```

- **Description**: The function is used to set the level to a target GPIO interface. The level can be used to alter the GPIO from an output to an output.

    Note:
    The GPIO has to be an output, otherwise, this function may have no effect.

- **Parameter**:

    ◦ **gpio**: The GPIO index number.
    ◦ **value**: The expected level: 0 as low level, 1 as high level.

- **Return**: None.

3.2.7. of_get_named_gpio

The interface has the following parameters.

- **Synopsis**:

```
int of_get_named_gpio(const struct device_node *np, const char *propname, int index)
```

- **Description**: The function is used to obtain a GPIO interface via which you can use the GPIO *Application Interface (API)*.
- **Parameter**:
    - **np**: Device node to get GPIO from.
    - **propname**: Name of property containing GPIO specifier(s).
    - **index**: Index of the GPIO.
- **Return**:
    - **Success**: The GPIO index number via which you can use the GPIO API.
    - **Failure**: Error code.

3.2.8. of_get_named_gpio_flags

The interface has the following parameters.

- **Synopsis**:

```
int of_get_named_gpio_flags(const struct device_node *np, const char *list_name, int index, enum
 of_gpio_flags *flags)
```

- **Description**: The function is used to obtain a GPIO number from the DTS file and analyze its GPIO properties.
- **Parameter**:
    - **np**: Device node to get GPIO from.
    - **propname**: Name of the property containing GPIO specifier(s).
    - **index**: Index of the GPIO
    - **flags**: Enumerate of_gpio_flags variables, including IO configuration, pull-up and pull-down settings, drive strength settings, etc.
- **Return**:
    - **Success**: The GPIO index number.
    - **Failure**: Error code.

4. Use Case

4.1. Using Pin to Drive DTS

The drivers are mainly used to configure the frequently-used functions of a pin, for example:

- General GPIO are only used for input, output and interrupt.
- Functional GPIO are mainly used for pin mux, for example, pin for UART, pin for I2C, and special functions.
- Some pins are both general and functional GPIO.

4.1.1. General GPIO

The following code block shows an example of a general GPIO device tree configuration.

```
&sdio0 {
        clock-frequency = <102400000>;
        max-frequency = <200000000>;
        card-detect-delay = <300>;
        bus-width = <4>;
        broken-cd;
        cap-sd-highspeed;
        post-power-on-delay-ms = <200>;
        cd-gpios = <&gpio 23 0>;
        status = "okay";
};
```

The input, output and interrupt of a general GPIO are all configured in the DTS configuration file.

In the line `cd-gpios = <&gpio 23 0>`, make sure you understand the use of the following values:

- **gpio**: The GPIO controller.
- **23**: The GPIO index number.
- **0**: Active level status. 0: low level active; 1: high level active.

4.1.2. Functional GPIO

The following code blocks show an example of a functional GPIO device tree configuration.

```
        pcie0_perst_default: pcie0_perst_default {
                perst-pins {
                        sf,pins = <PAD_GPIO26>;
                        sf,pinmux = <PAD_GPIO26_FUNC_SEL 0>;
                        sf,pin-ioconfig = <IO(GPIO_IE(1))>;
                        sf,pin-gpio-dout = <GPO_HIGH>;
                        sf,pin-gpio-doen = <OEN_LOW>;
                };
        };

        pcie0_perst_active: pcie0_perst_active {
                perst-pins {
                        sf,pins = <PAD_GPIO26>;
                        sf,pinmux = <PAD_GPIO26_FUNC_SEL 0>;
                        sf,pin-ioconfig = <IO(GPIO_IE(1))>;
                        sf,pin-gpio-dout = <GPO_LOW>;
                        sf,pin-gpio-doen = <OEN_LOW>;
                };
        };

        pcie0_power_active: pcie0_power_active {
                power-pins {
                        sf,pins = <PAD_GPIO32>;
                        sf,pinmux = <PAD_GPIO32_FUNC_SEL 0>;
                        sf,pin-ioconfig = <IO(GPIO_IE(1))>;
                        sf,pin-gpio-dout = <GPO_HIGH>;
                        sf,pin-gpio-doen = <OEN_LOW>;
                };
        };
```

```
&pcie0 {
        pinctrl-names = "perst-default", "perst-active", "power-active";
        pinctrl-0 = <&pcie0_perst_default>;
        pinctrl-1 = <&pcie0_perst_active>;
        pinctrl-2 = <&pcie0_power_active>;
        status = "okay";
};
```

In the above example, make sure you understand the use of the following values:

- The parameter value of “`perst-default`” in pinctrl-0 is the pin configuration for normal work mode.
- The parameter value of “`perst-active`” in pinctrl-1 is the pin configuration for active mode.

4.2. Using API to Drive DTS

4.2.1. Obtain Pin Control Resources

Usually by using the `devm_pinctrl_get` interface, you can obtain all the pin resources of a device.

The following code block provides an example.

```
        struct device *dev = &pcie->pdev->dev;

        pcie->pinctrl = devm_pinctrl_get(dev);
        if (IS_ERR_OR_NULL(pcie->pinctrl)) {
                dev_err(dev, "Getting pinctrl handle failed\n");
                return -EINVAL;
        }
```

4.2.2. Get Pin Control Status

By using the `pinctrl_lookup_state` interface, you can obtain all the pin status of a device.

The following code block provides an example.

```
        pcie->perst_state_def
                = pinctrl_lookup_state(pcie->pinctrl, "perst-default");
        if (IS_ERR_OR_NULL(pcie->perst_state_def)) {
                dev_err(dev, "Failed to get the perst-default pinctrl han-dle\n");
                return -EINVAL;
        }

        pcie->perst_state_active
                = pinctrl_lookup_state(pcie->pinctrl, "perst-active");
        if (IS_ERR_OR_NULL(pcie->perst_state_active)) {
                dev_err(dev, "Failed to get the perst-active pinctrl handle\n");
                return -EINVAL;
        }

        pcie->power_state_active
                = pinctrl_lookup_state(pcie->pinctrl, "power-active");
        if (IS_ERR_OR_NULL(pcie->power_state_active)) {
                dev_err(dev, "Failed to get the power-default pinctrl han-dle\n");
                return -EINVAL;
        }
```

4.2.3. Set Pin Control Status

By using the `pinctrl_select_state` interface, you can configure the pin status of a device.

The following code block provides an example.

```
        if (pcie->power_state_active) {
                ret = pinctrl_select_state(pcie->pinctrl, pcie->power_state_active);
                if (ret)
                        dev_err(dev, "Cannot set power pin to high\n");
        }

        if (pcie->perst_state_active) {
                ret = pinctrl_select_state(pcie->pinctrl, pcie->perst_state_active);
                if (ret)
                        dev_err(dev, "Cannot set reset pin to low\n");
        }
```

4.3. Realize Interrupt Functions

You can use the `gpiod_to_irq` interface to load the virtual interrupt signal, and then call the interrupt function.

The following code block provides an example.

```
        if (!(host->caps & MMC_CAP_NEEDS_POLL))
                irq = gpiod_to_irq(ctx->cd_gpio);

        if (irq >= 0) {
                if (!ctx->cd_gpio_isr)
                        ctx->cd_gpio_isr = mmc_gpio_cd_irqt;
                ret = devm_request_threaded_irq(host->parent, irq,
                        NULL, ctx->cd_gpio_isr,
                        IRQF_TRIGGER_RISING | IRQF_TRIGGER_FALLING | IRQF_ONESHOT,
                        ctx->cd_label, host);
                if (ret < 0)
                        irq = ret;
        }
```

5. FAQ

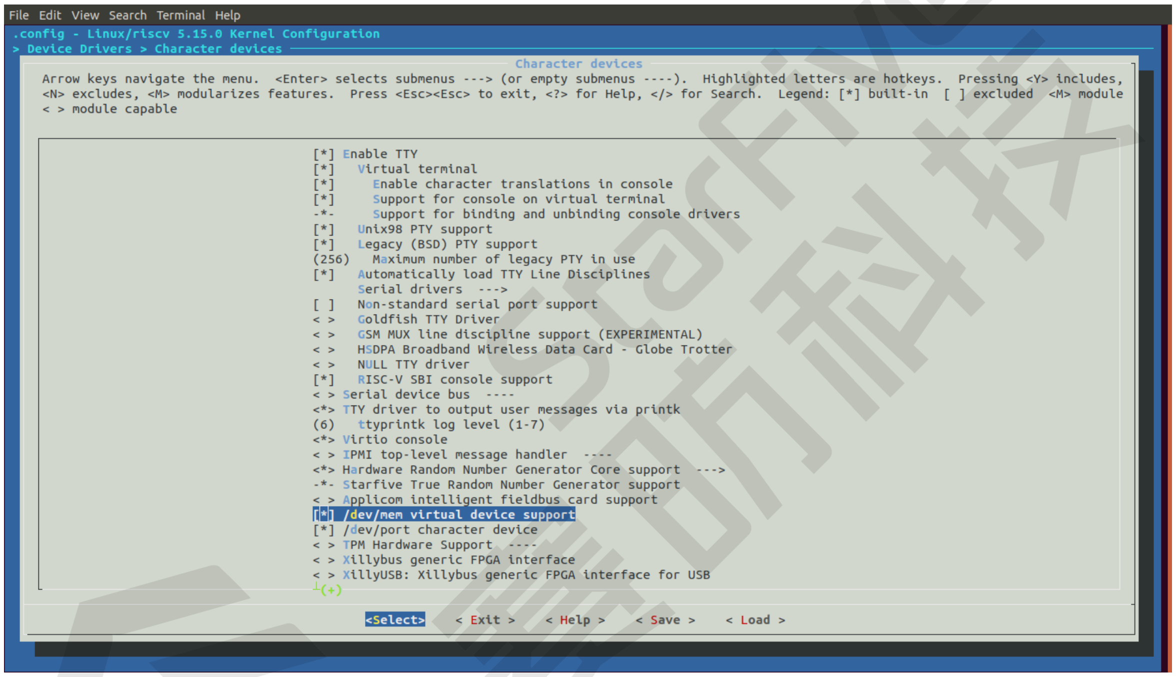
5.1. Common Debug Methods

5.1.1. Reading and Writing Registers via devmem

Follow the procedure below to perform the debug.

1. Enter the **/dev/mem virtual device support** menu.

Figure 5-1 Reading and Writing Registers via devmem



```
.config - Linux/riscv 5.15.0 Kernel Configuration
> Device Drivers > Character devices
                                  Character devices
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> includes,
  <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module
  < > module capable

                    [*] Enable TTY
                    [*]   Virtual terminal
                    [*]     Enable character translations in console
                    [*]     Support for console on virtual terminal
                    -*-     Support for binding and unbinding console drivers
                    [*]   Unix98 PTY support
                    [*]   Legacy (BSD) PTY support
                    (256)   Maximum number of legacy PTY in use
                    [*]   Automatically load TTY Line Disciplines
                          Serial drivers  --->
                    [ ]   Non-standard serial port support
                    < >   Goldfish TTY Driver
                    < >   GSM MUX line discipline support (EXPERIMENTAL)
                    < >   HSDPA Broadband Wireless Data Card - Globe Trotter
                    < >   NULL TTY driver
                    [*]   RISC-V SBI console support
                    < > Serial device bus  ----
                    <*> TTY driver to output user messages via printk
                    (6)   ttyprintk log level (1-7)
                    <*> Virtio console
                    < > IPMI top-level message handler  ----
                    <*> Hardware Random Number Generator Core support  --->
                    -*- Starfive True Random Number Generator support
                    < > Applicom intelligent fieldbus card support
                    [*] /dev/mem virtual device support
                    [*] /dev/port character device
                    < > TPM Hardware Support  ----
                    < > Xillybus generic FPGA interface
                    < > XillyUSB: Xillybus generic FPGA interface for USB
                    ┴(+)

                     <Select>    < Exit >    < Help >    < Save >    < Load >
```

2. Use the following commands to run your read or write operations.

   ◦ To read a register value from physical address "0x1304006c", execute the following command:

   ```
   devmem 0x1304006c
   ```

   ◦ To write a register value to physical address "0x1304006c", for example, with the lower 8-bit as 0x5D, execute the following command:

   ```
   devmem 0x1304006c 32 0x005b5c5d
   ```

   Then you can perform the above read command again to verify if your write operation is successful. The following code block shows the commands and returns.

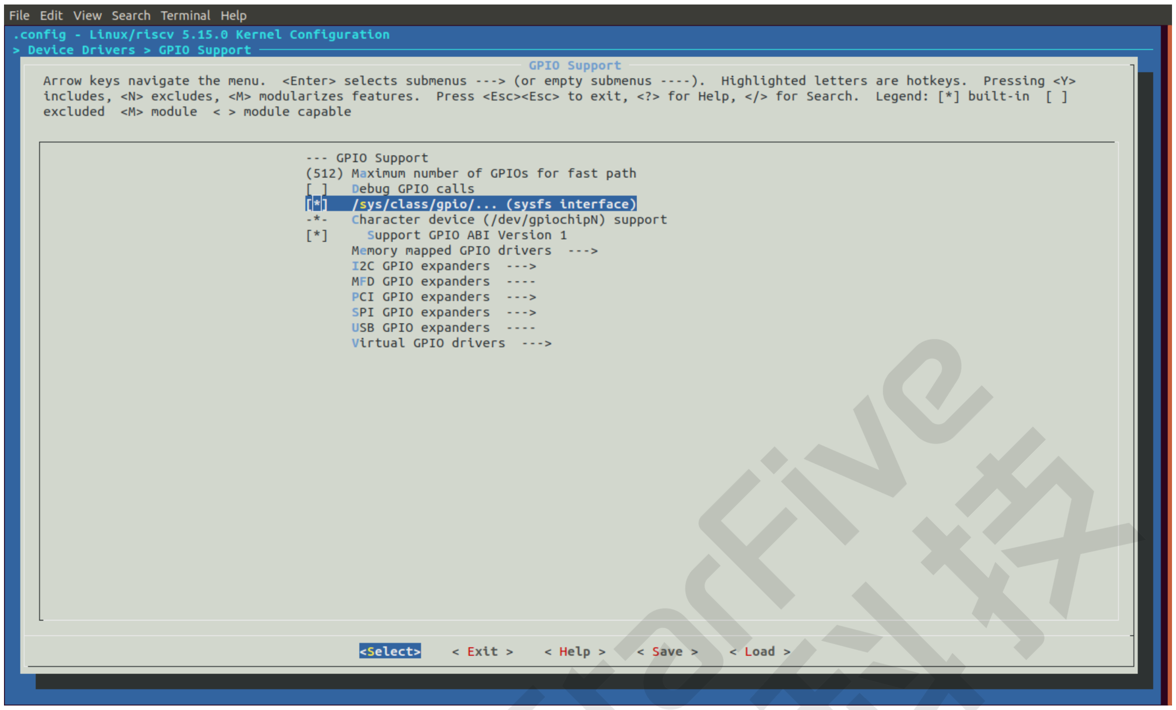
   ```
   #    devmem 0x1304006c
   0x00SBSC00
   #    devmem 0x1304006c 32 0x005b5c5d
   #    devmem 0x1304006c
   0X005B5C5D
   ```

5.1.2. Using sysfs to Debug

Follow the procedure below to perform the debug.

1. Enter the **/sys/class/gpio/...(sysfs interface)** menu.

Figure 5-2 Using sysfs to Debug



```
File  Edit  View  Search  Terminal  Help
.config - Linux/riscv 5.15.0 Kernel Configuration
> Device Drivers > GPIO Support
                                        GPIO Support
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]
  excluded  <M> module  < > module capable

                                  --- GPIO Support
                                  (512) Maximum number of GPIOs for fast path
                                  [ ]   Debug GPIO calls
                                  [*]   /sys/class/gpio/... (sysfs interface)
                                  -*-   Character device (/dev/gpiochipN) support
                                  [*]     Support GPIO ABI Version 1
                                        Memory mapped GPIO drivers  --->
                                        I2C GPIO expanders  --->
                                        MFD GPIO expanders  ----
                                        PCI GPIO expanders  --->
                                        SPI GPIO expanders  --->
                                        USB GPIO expanders  ----
                                        Virtual GPIO drivers  --->

                        <Select>    < Exit >    < Help >    < Save >    < Load >
```

2. Use the following commands to run the debug.

```
# cd /sys/class/gpio/
# ls
export gpiochip0 gpiochip64 unexport
# echo 44 > export
# ls
export gpio44 gpiochip0 gpiochip64 unexport
# cd gpio44/
# ls
active_low direction subsystem value
device edge uevent
# cat direction
in
# cat value
1
```