



StarFive  
赛昉科技

# 昉·惊鸿7110以太网开发和 移植手册

昉·星光 2

版本：1.0

日期：2022/12/30

Doc ID: JH7110-PGCH-001

# 法律声明

阅读本文件前的重要法律告知。

## 版权注释

版权 ©上海赛昉科技有限公司，2023。版权所有。

本文档中的说明均基于“视为正确”提供，可能包含部分错误。内容可能因产品开发而定期更新或修订。上海赛昉科技有限公司（以下简称“赛昉科技”）保留对本协议中的任何内容进行更改的权利，恕不另行通知。

赛昉科技明确否认任何形式的担保、解释和条件，无论是明示的还是默示的，包括但不限于适销性、特定用途适用性和非侵权的担保或条件。

赛昉科技无需承担因应用或使用任何产品或电路而产生的任何责任，并明确表示无需承担任何及所有连带责任，包括但不限于间接、偶然、特殊、惩戒性或由此造成的损害。

本文件中的所有材料受版权保护，为赛昉科技所有。不得以任何方式修改、编辑或断章取义本文件中的说明，本文件或其任何部分仅限用于内部使用或教育培训。

## 联系我们：

地址：浦东新区盛夏路61弄张润大厦2号楼502，上海市，201203，中国

网站：<http://www.starfivetech.com>

邮箱：

- [sales@starfivetech.com](mailto:sales@starfivetech.com)（销售）
- [support@starfivetech.com](mailto:support@starfivetech.com)（支持）

# 前言

关于本指南和技术支持信息

## 关于本手册

本手册主要为SDK开发者提供赛昉科技新一代SoC平台——昉·惊鸿7110的以太网模块的开发和移植指导。

## 受众

本手册主要服务于与以太网相关驱动程序的开发人员。如果您正在开发和移植其他模块，请与您的销售或支持顾问联系，获取昉·惊鸿7110的完整文档。






## 修订历史

表 0-1 修订历史

Version	发布说明	修订
1.0	2022/12/30	首次发布。

## 注释和注意事项

本指南中可能会出现以下注释和注意事项：

-  **提示：**  
建议如何在某个主题或步骤中应用信息。
-  **注：**  
解释某个特例或阐释一个重要的点。
-  **重要：**  
指出与某个主题或步骤有关的重要信息。
-  **警告：**  
表明某个操作或步骤可能会导致数据丢失、安全问题或性能问题。
-  **警告：**  
表明某个操作或步骤可能导致物理伤害或硬件损坏。

# 目录

表格清单.....	6
插图清单.....	7
法律声明.....	ii
前言.....	iii
<b>1. 简介.....</b>	<b>8</b>
1.1. 设备树概述.....	8
1.2. 设备树代码.....	9
<b>2. 以太网简介.....</b>	<b>11</b>
2.1. 关于以太网.....	11
2.2. 以太网设备框架.....	11
2.3. GMAC源代码结构.....	12
2.4. 配置.....	13
2.4.1. 内核菜单配置.....	13
2.4.2. 设备驱动程序配置.....	17
<b>3. 初始化U-Boot.....</b>	<b>21</b>
3.1. U-Boot源代码结构.....	21
3.2. U-Boot启动流程.....	21
<b>4. 添加一个新的以太网驱动程序.....</b>	<b>24</b>
4.1. 以太网驱动程序结构.....	24
4.2. 添加一个新的PHY.....	25
4.3. 在U-Boot中启用PHY.....	25
4.4. PHY设备初始化.....	27
<b>5. 驱动程序验证.....</b>	<b>31</b>
5.1. 验证环境.....	31
5.2. 验证新驱动程序.....	31
5.3. 通过MIDO命令访问PHY.....	32
5.4. PING - 数字环回.....	32
<b>6. 调试方法.....</b>	<b>34</b>
6.1. 通用调试命令.....	34
6.2. 一般故障排除步骤.....	35
<b>7. 已知问题.....</b>	<b>37</b>
7.1. 以太网GMAC仅支持RGMII.....	37
7.1.1. 仅支持1,000 M.....	37

---

7.1.2. 自动协商..... 37



# 表格清单

表 0-1 修订历史.....	iii
表 2-1 GMAC源代码结构.....	13



# 插图清单

图 1-1 设备树 workflow.....	9
图 2-1 以太网相关层.....	11
图 2-2 以太网设备框架.....	12
图 2-3 Networking support.....	14
图 2-4 Networking options.....	15
图 2-5 Device Drivers.....	16
图 2-6 Ethernet drivers support.....	17
图 3-1 U-Boot源代码结构.....	21
图 3-2 U-Boot启动流程1.....	22
图 3-3 U-Boot启动流程2.....	23
图 4-1 U-Boot PHY 结构示例.....	24
图 4-2 在配置文件中添加PHY.....	26
图 4-3 在设备初始化添加PHY.....	26
图 4-4 定义PHY数据结构.....	27
图 4-5 YT8521 PHY初始化.....	28
图 4-6 YT8531 PHY初始化1.....	28
图 4-7 YT8531 PHY初始化2.....	29
图 5-1 验证以太网驱动程序.....	32
图 5-2 MIDO命令.....	32
图 5-3 Ping命令.....	33
图 7-1 GMAC仅支持1,000 M.....	37
图 7-2 GMAC 10 M/100 M/1,000 M自动协商.....	38

# 1. 简介

与Linux操作系统中的所有其他SoC一样，U-Boot和以太网是开发应用程序和设计移植策略的前两个模块。

本手册主要介绍了移植昉·惊鸿7110 U-Boot和YT8531 PHY到新开发板的步骤。您可以使用本手册的信息作为移植任何其他以太网的参考。

本手册参考的源代码基于以下环境：

- SDK版本： 3.0
- U-Boot版本： 3.0
- Linux内核版本： 5.15



## 注：

对于不同的U-Boot和Linux内核版本，参考信息可能有所差异，在移植前，请与您的销售或支持顾问联系。

## 1.1. 设备树概述

自Linux 3.x以来，系统就引入了设备树作为数据结构和语言来描述硬件配置。设备树是硬件设置的系统可读描述，这样操作系统不必硬编码机器的详细信息。

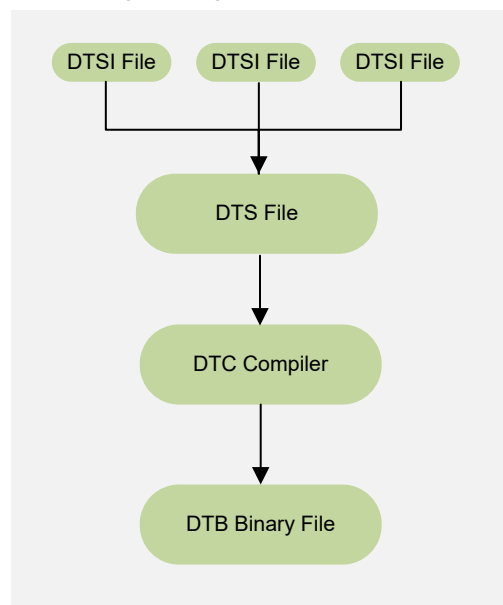
一个设备树主要有以下呈现形式。

- 设备树编译器（DTC）：用于将设备树编译为系统可读的二进制文件的工具。
- 设备树源码（DTS）：人类可读的设备树描述文件。您可以在此文件中找到目标参数并修改硬件配置。
- 设备树源码信息（DTSI）：可包括在设备树描述中的人类可读的头文件。您可以在此文件中找到目标参数并修改硬件配置。
- 设备树块（DTB）：系统可读设备树二进制blob文件，在系统中烧录以供执行。

下图显示了上述形式的关系（工作流）。



图 1-1 设备树工作流程



## 1.2. 设备树代码

### 总体结构

昉·惊鸿7110的设备树代码如下：

```

linux
├── arch
│   ├── riscv
│   │   ├── boot
│   │   │   ├── dts
│   │   │   │   └── starfive
│   │   │   │       ├── codecs
│   │   │   │       │   ├── sf_pdm.dtsi
│   │   │   │       │   ├── sf_pwm dac.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平台

昉·惊鸿7110 SoC平台的设备树源代码在以下路径：

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

## 昉·星光 2

昉·星光 2 单板计算机 (SBC) 的设备树源代码在以下路径：

```

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

```

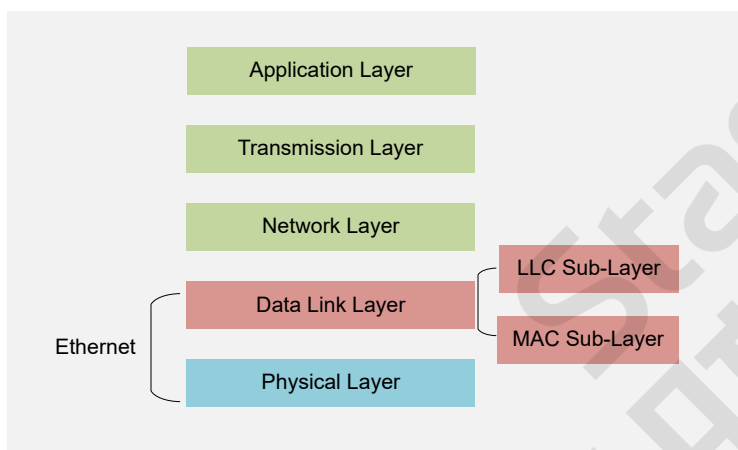
## 2. 以太网简介

本章介绍了配置现有以太网驱动程序的方法。

### 2.1. 关于以太网

以太网是一种基于局域网的网络通信技术，遵循IEEE802.3协议标准，包括10 M、100 M和1,000 M的以太网速度范围。在TCP/IP协议中，以太网位于以下层中。

图 2-1 以太网相关层

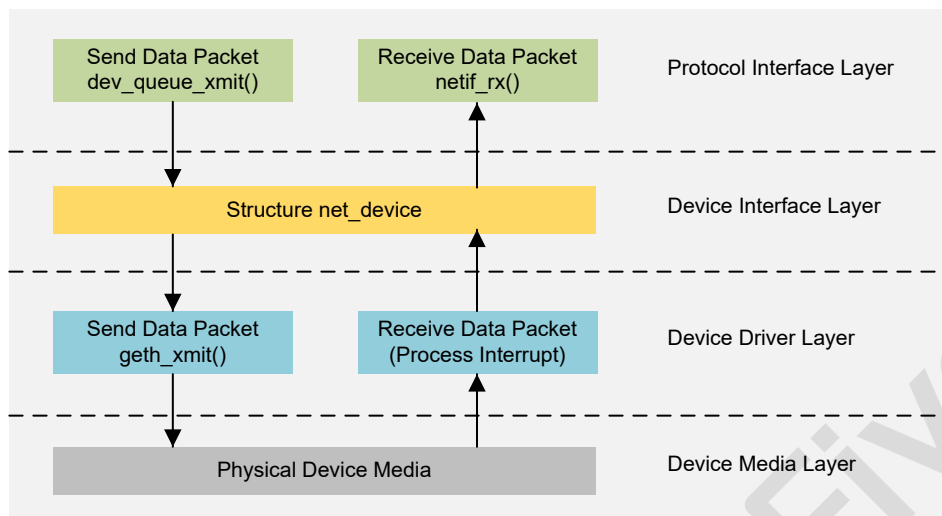


以太网与TCP/IP层中的物理层（L1: Physical Layer）和数据链路层（L2: Data Link Layer）相关。数据链路层包含逻辑链路控制（LLC）子层和多媒体访问控制（MAC）子层。

### 2.2. 以太网设备框架

下图为linux内核中网络设备框架，该框架具有以下层。

图 2-2 以太网设备框架



- **协议接口层 (Protocol Interface Layer)**：该层提供了统一的数据发送和接收接口。`dev_queue_xmit()`接口用于发送数据，`netif_rx()`接口用于接收数据。
- **设备接口层 (Device Interface Layer)**：该层提供了`net_device`的统一结构，用于描述网络设备的属性和操作细节。该结构可以作为设备驱动程序层中所有功能的容器来工作。
- **设备驱动程序层 (Device Driver Layer)**：该层实现了在`net_device`结构中定义的功能操作指针，然后将这些操作移交给硬件驱动程序进行执行。
- **设备媒体层 (Device Media Layer)**：该层包含作为完成数据包发送和接收任务的物理元素，包括网络传输适配器和用于传输的介质。

## 2.3. GMAC源代码结构

GMAC的源代码结构在以下路径：

```
Drivers/net/ethernet/stmicro/stmmac
```

下面的代码块为GMAC源代码的示例。

```
1 Drivers/net/ethernet/stmicro/stmmac
2
3 |— stmmac.h
4 |— dwmac-starfive-plat.c
5 |— stmmac_main.c
```

表 2-1 GMAC源代码结构

文件	说明
stmmac.h	DWMAC平台的GMAC驱动程序头文件。在此文件中，定义了一些宏、数据结构和内部接口。
dwmac-starfive-plat.c	赛昉科技DWMAC平台的GMAC驱动程序特定配置选项。
stmmac_main.c	DWMAC平台上GMAC驱动程序公共接口。

## 2.4. 配置

### 2.4.1. 内核菜单配置

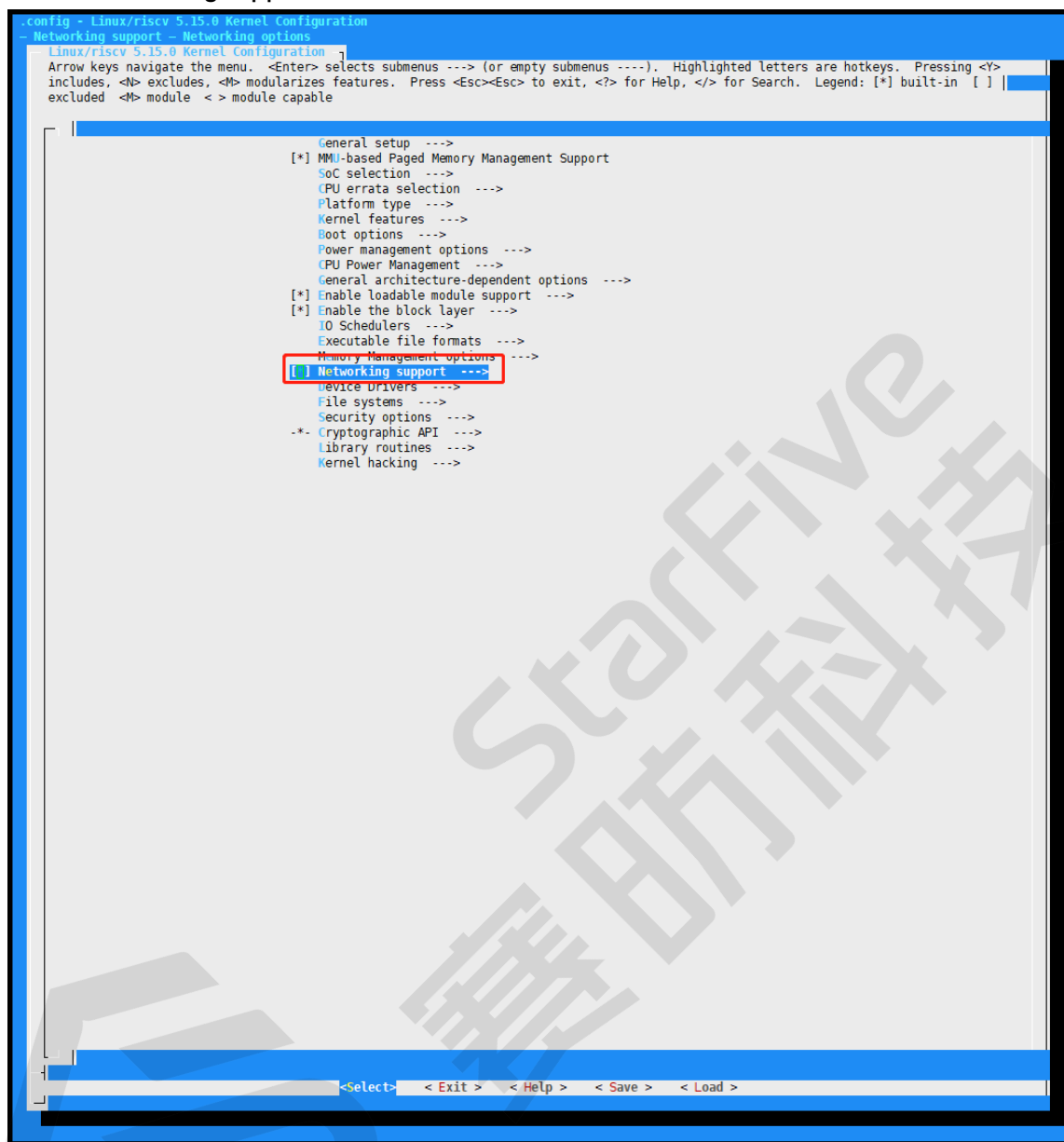
按照以下步骤，在内核菜单对话框中启用GMAC支持。

1. 在freelight-u-sdk的根目录下，输入以下命令以进入内核菜单配置GUI。

```
make linux-menuconfig
```

2. 进入**Networking support**菜单。

图 2-3 Networking support



3. 进入Networking options 菜单， 并选择supported network protocols选项。

图 2-4 Networking options

```

config - Linux/riscv 5.15.0 Kernel Configuration
Networking support Networking options
Networking options
  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 <#> module <> module capable

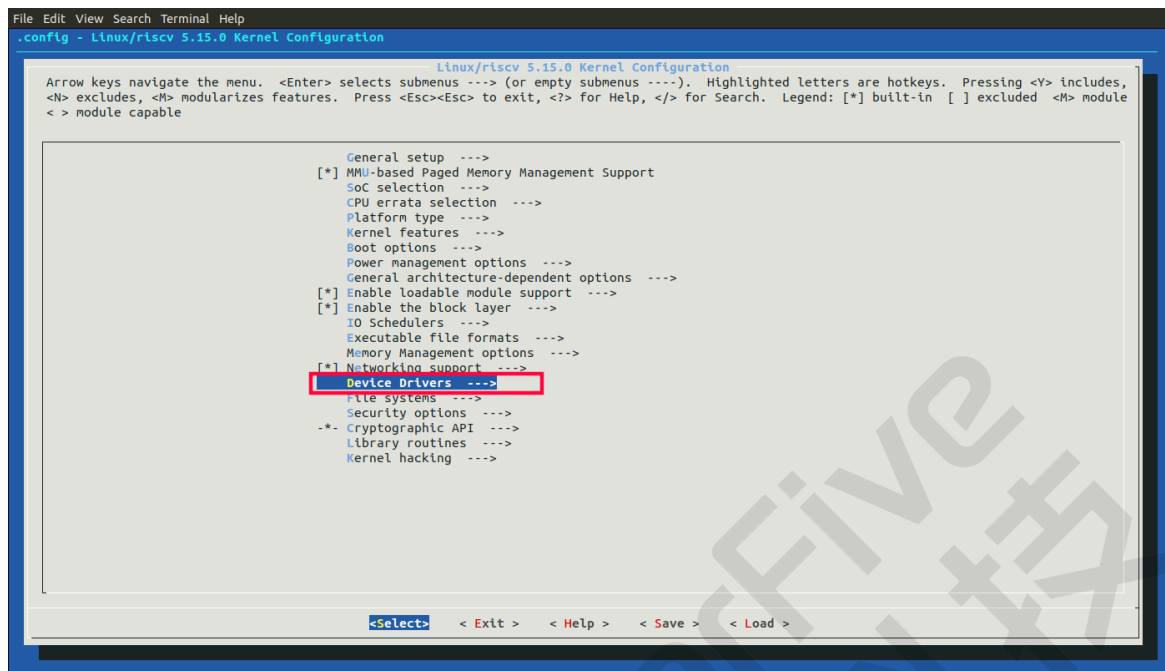
  <+> Packet socket
  <> Packet: sockets monitoring interface
  <#> Unix domain sockets
  <> UNIX: socket monitoring interface
  <> Transport Layer Security support
  <> Transformation user configuration interface
  <> PF_KEY sockets
  [ ] UDP sockets
  [*] TCP/IP networking
  [*] IP: multicasting
  [*] IP: advanced router
  [ ] IP: IB TRIE statistics
  [ ] IP: policy routing
  [ ] IP: equal cost multipath
  [ ] IP: verbose route monitoring
  [*] IP: kernel level autoconfiguration
  [*] IP: DHCP support
  [*] IP: BOOTP support
  [*] IP: RARP support
  <> IP: tunneling
  <> IP: GRE demultiplexer
  [ ] IP: multicast routing
  [ ] IP: TCP syncookie support
  <> Virtual (secure) IP: tunneling
  <> IP: Foo (IP protocols) over UDP
  <> IP: AH transformation
  <> IP: ESP transformation
  <> IP: IPComp transformation
  <#> INET: socket monitoring interface
  <> UDP: socket monitoring interface
  <> RAW: socket monitoring interface
  [ ] INET: allow privileged process to administratively close sockets
  [ ] TCP: advanced congestion control ----
  [ ] TCP: MD5 Signature Option support (RFC2385)
  <> The IPv6 protocol ----
  [ ] MPTCP: Multipath TCP
  [ ] Security Marking
  [ ] Timestamping in PHY devices
  [*] Network packet filtering framework (Netfilter) --->
  [ ] IPF based packet filtering framework (BPFILTER) ----
  <> The DCCP Protocol ----
  <> The SCTP Protocol ----
  <> The Reliable Datagram Sockets Protocol
  <> The TIPC Protocol ----
  <> Asynchronous Transfer Mode (ATM)
  <> Layer Two Tunneling Protocol (L2TP) ----
  <> 802.1d Ethernet Bridging
  <> Distributed Switch Architecture ----
  <> 802.1Q/802.1ad VLAN Support
  <> IECnet Support
  <> ANSI/IEEE 802.2 LLC type 2 Support
  <> Appletalk protocol support
  <> CCITT X.25 Packet Layer
  <> LAPB Data Link Driver
  <> Phonet protocols family
  <> IEEE Std 802.15.4 Low-Rate Wireless Personal Area Networks support ----
  [ ] QoS and/or fair queueing ----
  [ ] Data Center Bridging support
  <#> DNS Resolver support
  <> B.A.T.M.A.N. Advanced Meshing Protocol

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

```

## 4. 进入Device Drivers菜单。

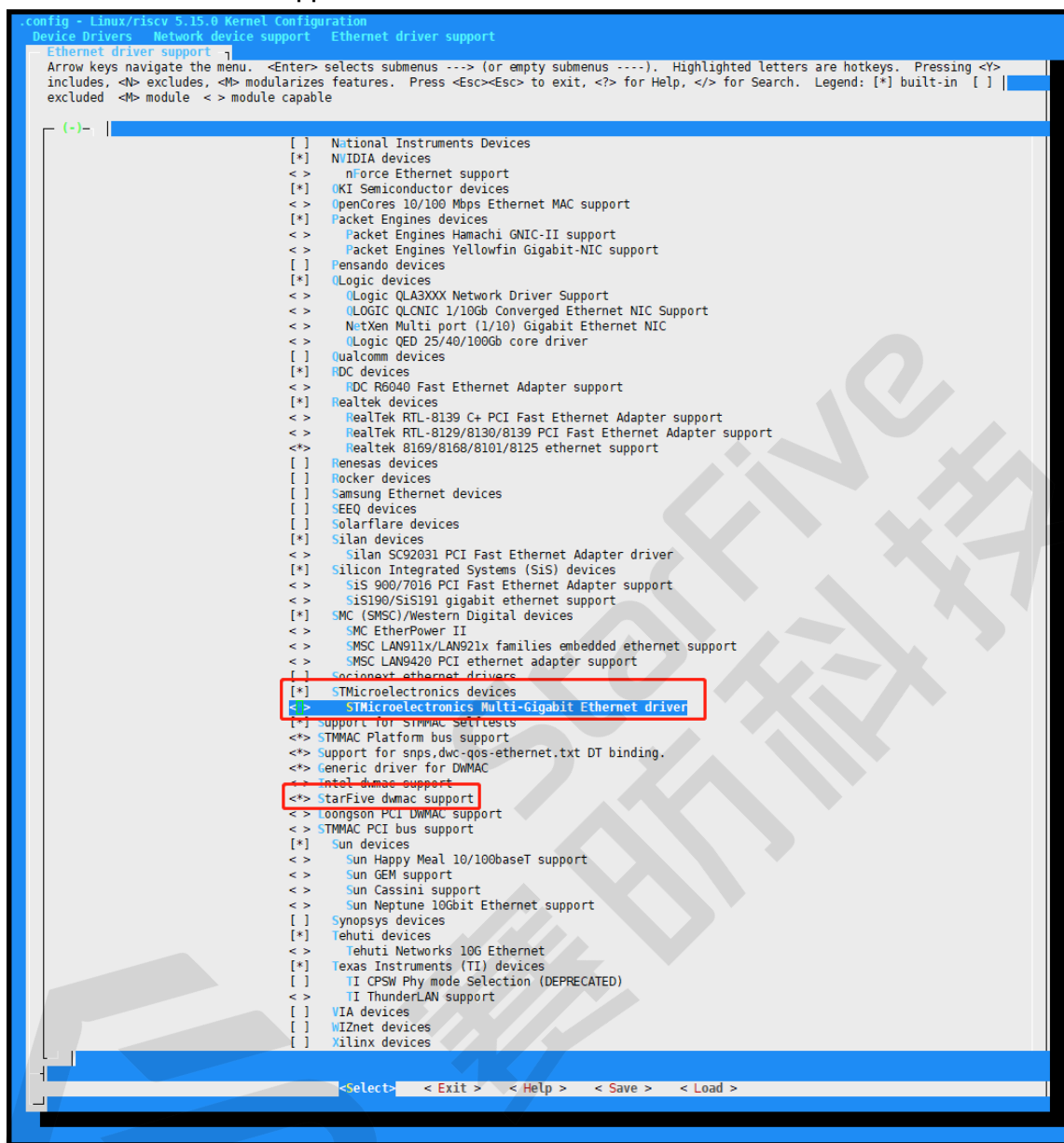
图 2-5 Device Drivers



5. 进入Network device support > Ethernet drivers support菜单，并选择您希望系统支持的GMAC驱动设备。



图 2-6 Ethernet drivers support



6. 保存更改，并退出内核配置对话框。

## 2.4.2. 设备驱动程序配置

DTS/DTSI文件用于存储所有设备树配置。

以太网的设备树源代码在以下路径：

```
linux-5.10/arch/riscv/boot/dts/starfive/
```

以下代码块为以太网的DTS文件结构。

```
linux-5.15.0
L-- arch
```

```

L-- | -- riscv
| -- | -- | -- boot
| -- | -- | -- dts
| -- | -- | -- | -- starfive
| -- | -- | -- | -- | -- jh7110-common.dtsi
| -- | -- | -- | -- | -- | -- jh7110.dts

```

以下代码块为jh7110.dts文件中“gmac0”的设备树源代码示例。

```

gmac0: ethernet@16030000 {
    compatible = "starfive,dwmac", "snps,dwmac-5.10a";
    reg = <0x0 0x16030000 0x0 0x10000>;
    clock-names = "gtx",
        "tx",
        "ptp_ref",
        "stmmaceth",
        "pclk",
        "gtxc",
        "rmii_rtx";
    clocks = <&clkgen JH7110_GMAC0_GTXCLK>,
        <&clkgen JH7110_U0_GMAC5_CLK_TX>,
        <&clkgen JH7110_GMAC0_PTP>,
        <&clkgen JH7110_U0_GMAC5_CLK_AHB>,
        <&clkgen JH7110_U0_GMAC5_CLK_AXI>,
        <&clkgen JH7110_GMAC0_GTXC>,
        <&clkgen JH7110_GMAC0_RMII_RTX>;
    resets = <&rstgen RSTN_U0_DW_GMAC5_AXI64_AHB>,
        <&rstgen RSTN_U0_DW_GMAC5_AXI64_AXI>;
    reset-names = "ahb", "stmmaceth";
    interrupts = <7>, <6>, <5> ;
    interrupt-names = "macirq", "eth_wake_irq", "eth_lpi";
    max-frame-size = <9000>;
    phy-mode = "rgmii-id";
    snps,multicast-filter-bins = <64>;
    snps,perfect-filter-entries = <128>;
    rx-fifo-depth = <2048>;
    tx-fifo-depth = <2048>;
    snps,fixed-burst;
    snps,no-pbl-x8;
    snps,force_thresh_dma_mode;
    snps,axi-config = <&stmmac_axi_setup>;
    snps,tso;
    snps,en-tx-lpi-clockgating;
    snps,en-lpi;
    snps,write-requests = <4>;
    snps,read-requests = <4>;
    snps,burst-map = <0x7>;
    snps,txpbl = <16>;
    snps,rxpbl = <16>;
    status = "disabled";

```

```
};
```

以下提供了对上述代码块中的参数说明。

- **compatible**: 兼容性信息，用于连接驱动程序和目标设备。
- **reg**: 寄存器基本地址“0x16030000”和范围“0x10000”。
- **clocks**: 以太网模块使用到的时钟。
- **clock-names**: 上述时钟的名称。
- **resets**: 以太网模块使用到的复位信号。
- **reset-names**: 上述复位信号的名称。
- **interrupts**: 硬件中断ID。
- **interrupt-names**: 上述中断的名称。
- **phy-mode**: 以太网PHY模式，例如，“rgmii”或“rmii”。
- **snps**: 有关PHY特定参数，请参阅概要文件。
- **status**: 以太网工作状态，“启用”或“禁用”。

以下代码块为 jh7110-common.dtsi 文件中“gmac0”的设备树源代码示例。

```
&gmac0 {
    status = "okay";
    #address-cells = <1>;
    #size-cells = <0>;
    phy0: ethernet-phy@0 {
        rxc_dly_en = <1>;
        rx_delay_sel = <0>;
        tx_delay_sel_fe = <5>;
        tx_delay_sel = <0xa>;
        tx_inverted_10 = <0x1>;
        tx_inverted_100 = <0x1>;
        tx_inverted_1000 = <0x1>;
    };
};

&gmac1 {
    #address-cells = <1>;
    #size-cells = <0>;
    status = "okay";
    phy1: ethernet-phy@1 {
        tx_delay_sel_fe = <5>;
        tx_delay_sel = <0>;
        rxc_dly_en = <0>;
        rx_delay_sel = <0>;
        tx_inverted_10 = <0x1>;
        tx_inverted_100 = <0x1>;
    };
};
```

```
tx_inverted_1000 = <0x0>;  
};  
};};22 };
```

以下提供了对上述代码块中的参数说明。

- **rx\_c\_dly\_en**: 此字段用于设置是否在RGMII模式下启用接收器的2ns时延。1: 启用; 0: 禁用。
- **rx\_delay\_sel**: 此字段用于配置接收器时钟延迟, 每步长150 ps, 可接受范围: 0x0到0xf。
- **tx\_delay\_sel\_fe**: 此字段用于配置10 M和100 M模式下发送器时钟延迟, 每步长150 ps, 可接受范围: 0x0到0xf。
- **tx\_delay\_sel**: 此字段用于配置1,000 M模式下发送器时钟延迟, 每步长150 ps, 可接受范围: 0x0到0xf。
- **tx\_inverted\_10**: 此字段用于设置是否在10 M模式下启用发送器时钟反转。1: 启用; 0: 禁用。
- **tx\_inverted\_100**: 此字段用于设置是否在100 M模式下启用发送器时钟反转。1: 启用; 0: 禁用。
- **tx\_inverted\_1000**: 此字段用于设置是否在1,000 M模式下启用发送器时钟反转。1: 启用; 0: 禁用。

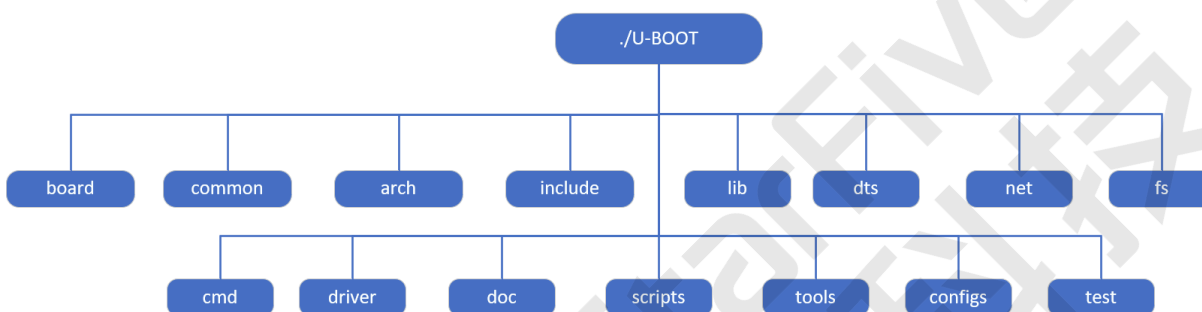
## 3. 初始化U-Boot

本章介绍初始化U-Boot的方法，作为添加新设备驱动程序所做的准备。

### 3.1. U-Boot源代码结构

下图为昉·惊鸿7110的U-Boot源代码文件目录。

图 3-1 U-Boot源代码结构



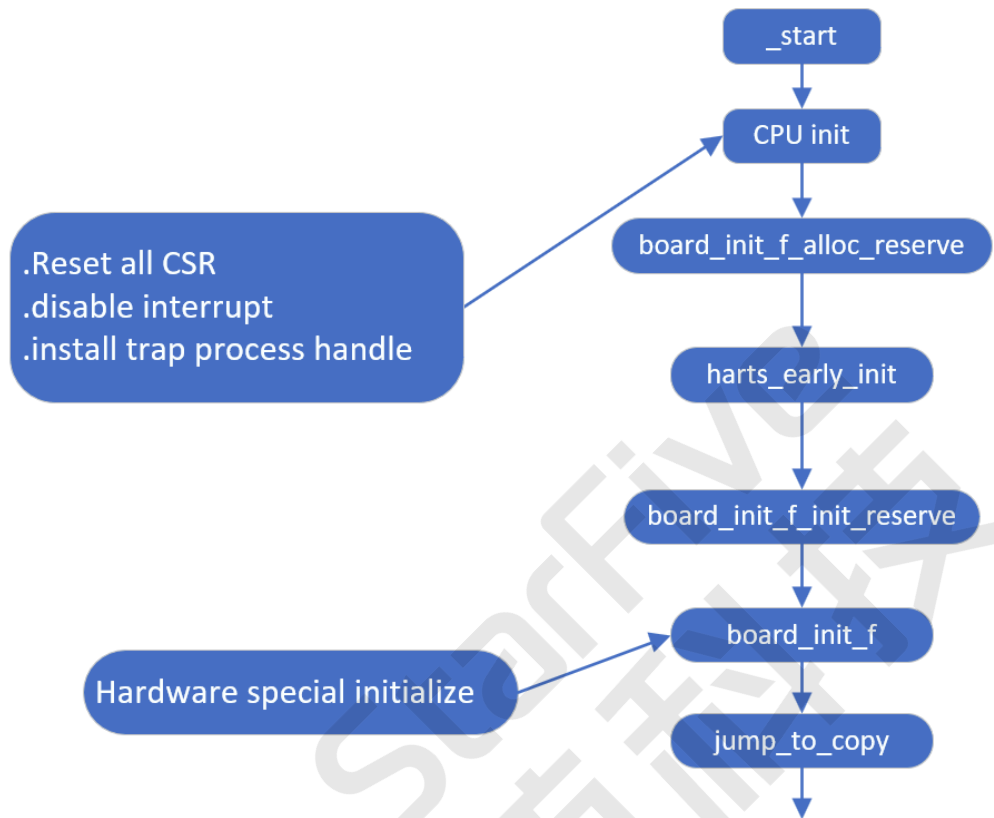
以下为对上图中文件夹的介绍。

- **board**: board文件夹包含所有指定板的文件，包含赛昉科技昉·惊鸿7110和昉·星光 2的文件等。
- **arch**: 指定核心文件夹，包含所有核心初始化文件。这些文件不独立于开发板；因此，您不需要修改此文件夹中的任何内容。
- **driver**: 该文件夹包含U-Boot支持的所有驱动程序，包括以太网驱动程序、PHY驱动程序、USB驱动程序等。
- **net**: 该文件夹包含U-Boot中支持的所有上层协议，包括ping、tftp、icmp和其他协议。
- **cmd**: 该文件夹包含U-Boot支持的所有命令。
- **configs**: 该文件夹包含所有去配置文件，每个文件关联一个特殊开发板。
- **scripts**: 该文件夹包含用于编译的规则文件。

### 3.2. U-Boot启动流程

下图为U-Boot启动流程框图。

图 3-2 U-Boot启动流程1



以下为上图中每个过程的描述。

- **\_start**: 每个相同架构的开发板都有相同的start.s文件，文件位于arch目录下。**\_start**是系统在核心通电时将使用的第一个指令。
- **CPU init**: CPU初始化步骤，可设置所有CPU相关和指定的寄存器。如上图所示，该步骤还可设置RISC-V核心指定的寄存器。



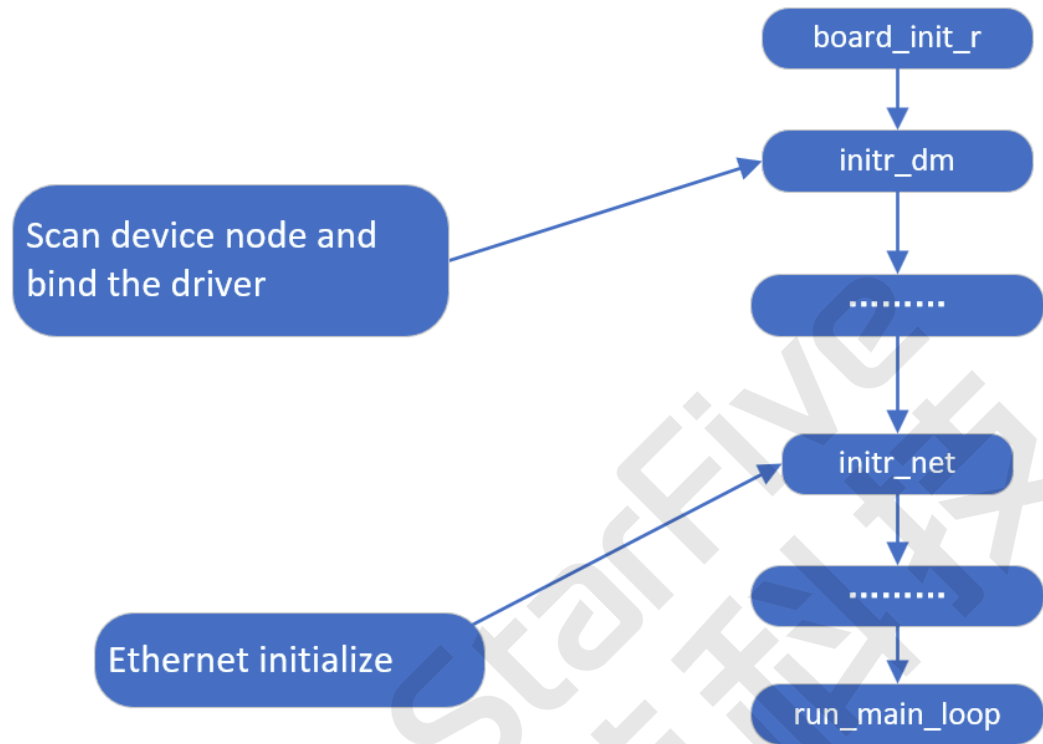
注：

U-Boot的启动仅占用一个核心，其他核心均被设置为空闲模式。多数情况下，U-boot在Linux启动之前不会使用辅助核心。

- **board\_init\_f\_alloc\_reserve**: 保留早期的**malloc** arena和全球数据**struct** arena。
- **harts\_early\_init**: 配置harts的专有设置和自定义CSR。
- **board\_init\_f\_init\_reserve**: 初始化保留空间。
- **board\_init\_f**: 在重新定位符号之前，先初始化基本的硬件和运行环境，如CPU、计时器、控制台和设备树等。
- **jump\_to\_copy**: 将全局数据**struct**复制到高地址空间，并重新定位监视器代码。

重新定位符号和监视器代码之后，系统将开始以下启动过程。

图 3-3 U-Boot启动流程2



以下为上图中每个过程的描述。

- **board\_init\_r**: 开发板初始化文件。上图所示的所有与开发板相关的初始化过程都将逐一执行。
- **initr\_dm**: 扫描设备节点，并与正确的驱动程序保持关联。
- **initr\_net**: 以太网初始化文件。该文件将初始化您希望包含在开发板上的所有以太网接口。
- **main\_loop**: 屏幕上弹出U-Boot之前的最后一个初始化步骤。

**结果**: 所有过程完成后，U-boot完成启动并准备使用。

在**initr\_net**进程中，位于drivers/net/phy/phy.c文件夹下的**phy\_init**函数将用于初始化以太网PHY。参见[PHY设备初始化 \(第 27页\)](#)获取更多信息。

## 4. 添加一个新的以太网驱动程序

如果U-Boot中不支持使用的以太网PHY，您可以按照以下步骤为新设备添加PHY驱动程序代码。

### 4.1. 以太网驱动程序结构

以下代码块为高度概括了以太网PHY结构。

```
phy_yutai_init(void)
{
    phy_register(&YT8512_driver);
    phy_register(&YT8521_driver);
    phy_register(&YT8531_driver);

    return 0;
}
```

以上文件包含了U-Boot中默认支持的所有以太网PHY支持（自适应）。

正如本文件中所述，系统将逐个初始化所提到的PHY。



注：

如果您发现启动流程所用时间较长，通过检查每个PHY的使用，您可以移除一些未使用的PHY，只留下所需的PHY。

下面以一个特定的U-Boot PHY结构为例。

图 4-1 U-Boot PHY 结构示例

```
static struct phy_driver YT8521_driver = {
    .name = "YuTai YT8521",
    .uid = 0x0000011a,
    .mask = 0x00000fff,
    .features = PHY_GBIT_FEATURES,
    .config = &yt8521_config,
    .startup = &ytphy_startup,
    .shutdown = &genphy_shutdown,
};
```

以下为上述参数的描述：

- **.name**：您希望支持的以太网PHY的名称，您可以输入一个随机名称，但建议输入一个特定设备的名称，以便将来进行维护。
- **.uid**：制造商ID以及以太网PHY设备ID，可以在PHY制造商的手册中找到。



- **.mask**: 以太网PHY的掩码，在示例“0x0000ffff”中，数字“f”的位置是UID号。在实践中，可以省略这个数字，以简化输入。
- **.feature**: PHY的千兆位特征。例如，以太网PHY是否为千兆位PHY。
- **.config**: 介绍如何初始化以太网PHY的函数调用。对于大多数PHY，是不需要配置的。对于具有QSGMI和RMII的复杂PHY，则需要通过配置来指定PHY的角色。

## 4.2. 添加一个新的PHY

例如，如果您希望在Motorcomm中添加一个新的名为YT8531的PHY，您需要找到drivers/net/phy/motorcomm.c文件，并执行以下操作。

- 按照已有的数据结构创建一个新的结构。文件中的数据结构由U-Boot定义，若要添加新的PHY支持，您必须准确地遵循数据结构和格式。
- 重用现有的启动和关闭功能。只有在您的设备有特殊要求时才会修改它们。
- 通过将**phy\_register ()**的函数调用添加为新条目，以确保您已经注册了新的PHY，例如：

```
phy_register(&YT8531_driver)
```



**注：**

如果要从其他供应商添加PHY，请确保找到用C语言编写的用于PHY注册的正确文件，例如，对于Broadcom PHY，使用文件broadcom.c。

## 4.3. 在U-Boot中启用PHY

执行以下步骤，在U-Boot上启用新的PHY：

1. 要为U-Boot启用新的PHY，首先需要在开发板特定的头文件中定义宏定义。

下面的代码块提供了一个在昉·星光 2头文件/configs/starfive-visionfive.h.h中添加YT8531 PHY的示例。

```
#define DWC_NET_PHYADDR
```



**注：**

请确保您在头文件中定义的PHY地址是正确的，否则，系统必须枚举所有可用的PHY地址。

2. 然后，您需要在配置文件中添加定义宏定义。

下图显示了在配置文件中添加YT8531 PHY的示例。

图 4-2 在配置文件中添加PHY

```

int phy_init(void)
{
#ifdef CONFIG_B53_SWITCH
    phy_b53_init();
#endif
#ifdef CONFIG_MV88E61XX_SWITCH
    phy_mv88e61xx_init();
#endif
#ifdef CONFIG_PHY_AQUANTIA
    phy_aquantia_init();
#endif
#ifdef CONFIG_PHY_ATHEROS
    phy_atheros_init();
#endif
    .....
    .....

#ifdef CONFIG_PHY_NCSI
    phy_ncsi_init();
#endif
#ifdef CONFIG_PHY_XILINX_GMII2RGMII
    phy_xilinx_gmii2rgmii_init();
#endif
#ifdef CONFIG_PHY_YUTAI
    phy_yutai_init();
#endif
    genphy_init();

    return 0;
}

```

3. 然后，您就可以为PHY设备的初始化添加一个新的条目。

下图提供了一个在文件驱动程序/net/phy/mowercomm.c中添加YT8531 PHY的示例。

图 4-3 在设备初始化添加PHY

```

int phy_yutai_init(void)
{
    phy_register(&YT8512_driver);
    phy_register(&YT8521_driver);
    phy_register(&YT8531_driver);

    return 0;
}

```

4. 然后你需要定义驱动程序结构。

下图提供了一个在文件驱动程序drivers/net/phy/motorcomm.c中定义YT8531 PHY的示例。

图 4-4 定义PHY数据结构

```
static struct phy_driver YT8531_driver = {  
    .name           = "YT8531 Gigabit Ethernet",  
    .uid            = PHY_ID_YT8531,  
    .mask           = MOTORCOMM_PHY_ID_MASK,  
    .features       = PHY_GBIT_FEATURES,  
    .config         = &yt8531_config,  
    .startup        = &ytphy_startup,  
    .shutdown       = &genphy_shutdown,  
};
```

## 4.4. PHY设备初始化

下图显示了YT8521 PHY设备初始化代码的示例。

图 4-5 YT8521 PHY初始化

```

static int yt8521_config(struct phy_device *phydev)
{
    int ret, val;

    ret = 0;
    genphy_config_aneg(phydev);

    /* disable auto sleep */
    val = ytphy_read_ext(phydev, EXTREG_SLEEP_CONTROL);
    if (val < 0)
        return val;

    val &= ~(1 << YT8521_EN_SLEEP_SW_BIT);
    ret = ytphy_write_ext(phydev, EXTREG_SLEEP_CONTROL, val);
    if (ret < 0)
        return ret;

    /*set delay config*/
    ret = ytphy_of_config(phydev);
    if (ret < 0)
        return ret;

    val = ytphy_read_ext(phydev, YT8521_EXT_CLK_GATE);
    if (val < 0)
        return val;

    val &= ~(1 << 12);
    ret = ytphy_write_ext(phydev, YT8521_EXT_CLK_GATE, val);
    if (ret < 0)
        return ret;

    return 0;
}

```

下图显示了YT8531 PHY设备初始化代码的示例。

图 4-6 YT8531 PHY初始化1

```

static int yt8531_config(struct phy_device *phydev)
{
    int ret;

    ret = 0;
    genphy_config_aneg(phydev);

    /* set delay config */
    ret = ytphy_of_config(phydev);
    if (ret < 0)
        return ret;
    return 0;
}

```

图 4-7 YT8531 PHY初始化2

```

static int ytphy_of_config(struct phy_device *phydev)
{
    ofnode node;
    u32 val;
    u32 cfg;
    int i;

    node = phydev->node;
    if (!ofnode_valid(node)) {
        /* Look for a PHY node under the Ethernet node */
        node = dev_read_subnode(phydev->dev, "ethernet-phy");
    }

    if (!ofnode_valid(node)) /* No node found*/
        return 0;

    /*read rxc_dly_en config*/
    cfg = ofnode_read_u32_default(node, ytphy_rxden_grp[0].name, ~0);
    if (cfg != -1) {

        val = ytphy_read_ext(phydev, YTPHY_EXTREG_CHIP_CONFIG);

        /*check the cfg overflow or not*/
        cfg = (cfg > ((1 << ytphy_rxden_grp[0].size) - 1)) ?
            ((1 << ytphy_rxden_grp[0].size) - 1) : cfg;

        val = bitfield_replace(val, ytphy_rxden_grp[0].off,
            ytphy_rxden_grp[0].size, cfg);
        ytphy_write_ext(phydev, YTPHY_EXTREG_CHIP_CONFIG, val);
    }

    /* set drive strenght of rxd/rx_ctl rgmii pad */
    val = ytphy_read_ext(phydev, YTPHY_PAD_DRIVES_STRENGTH_CFG);
    val |= YTPHY_RGMII_SW_DR_MASK;
    ytphy_write_ext(phydev, YTPHY_PAD_DRIVES_STRENGTH_CFG, val);

    val = ytphy_read_ext(phydev, YTPHY_EXTREG_RGMII_CONFIG1);
    for (i = 0; i < ARRAY_SIZE(ytphy_rxtxd_grp); i++) {

        cfg = ofnode_read_u32_default(node,
            ytphy_rxtxd_grp[i].name, ~0);
        cfg = (cfg != -1) ? cfg : ytphy_rxtxd_grp[i].dflt;

        /*check the cfg overflow or not*/
        cfg = (cfg > ((1 << ytphy_rxtxd_grp[i].size) - 1)) ?
            ((1 << ytphy_rxtxd_grp[i].size) - 1) : cfg;

        val = bitfield_replace(val, ytphy_rxtxd_grp[i].off,
            ytphy_rxtxd_grp[i].size, cfg);
    }

    return ytphy_write_ext(phydev, YTPHY_EXTREG_RGMII_CONFIG1, val);
}

```

上面的函数调用指定了如何初始化以太网PHY。您必须使用MDIO总线来访问PHY控制寄存器，因此，在配置之前，请务必确认已正确配置MDIO接口。



## 5. 驱动程序验证

### 5.1. 验证环境

在您开始验证新的以太网驱动设备前，您需要为以下条目定义环境变量。

- U-Boot
- 开发板IP地址（通过设置`ipaddr`变量）
- 主动式以太网接口（通过设置`ethact`变量）
- MAC接口地址（通过设置`ethaddr`变量）

作为单过程操作系统，Linux一次只能操作一个以太网驱动程序（接口）。因此您需要在上述参数中指定，以便在使用之前通知U-Boot哪个接口是活动的。

以下代码为一个示例：

```
====>print
baudrate=115200
bootargs=console=ttyS0,115200 debug rootwait earlycon=sbi
bootcmd=run load_vf2_env;run importbootenv;run boot2;run distro_bootcmd
bootcmd_mmc0=devnum=0; run mmc_boot
bootdelay=2
bootdir=/boot
eth0addr=6c:cf:39:7c:4e:22
eth1addr=6c:cf:39:7c:3e:53
ethact=ethernet@16030000
ethaddr=6c:cf:39:7c:4e:22
ipaddr=192.168.120.230
netmask=255.255.255.0
stderr=serial@10000000
stdin=serial@10000000
stdout=serial@10000000
```

### 5.2. 验证新驱动程序

添加一个新的以太网驱动程序后，您将在第二次访问U-Boot时看到以下界面。

图 5-1 验证以太网驱动程序

```

U-Boot 2021.10-dirty (Nov 23 2022 - 15:24:46 +0800)

CPU:   rv64imacu
Model: StarFive VisionFive V2
DRAM:  8 GiB
MMC:   sdio0@16010000: 0, sdio1@16020000: 1
Loading Environment from SPIFlash... SF: Detected gd25lq128 with page size 256 Bytes, erase size 4 KiB, total 16 MiB
*** Warning - bad CRC, using default environment

StarFive EEPROM format v2

-----EEPROM INFO-----
Vendor : StarFive Technology Co., Ltd.
Product full SN: VF7110B1-2228-D008E032-00000001
data version: 0x2
PCB revision: 0x1
BOM revision: B
Ethernet MAC0 address: 6c:cf:39:7c:4e:22
Ethernet MAC1 address: 6c:cf:39:7c:3e:53
-----EEPROM INFO-----

In:    serial@10000000
Out:   serial@10000000
Err:   serial@10000000
Model: StarFive VisionFive V2
Net:   eth0: ethernet@16030000, eth1: ethernet@16040000
Switch to partitions #0, OK
mmc1 is current device
found device 1
bootmode flash device 1
Failed to load 'uEnv.txt'
Can't set block device
Hit any key to stop autoboot:  0
StarFive #
StarFive #

```

上图红框中的信息表示对该接口的SoC支持已经准备好使用，但是，如果数据在PHY中被阻塞，我们仍需要验证数据通信。

### 5.3. 通过MIDO命令访问PHY

您需要使用MDIO命令访问以太网PHY。

下图为以太网PHY列表，每个PHY都有对应的命令来访问。

图 5-2 MIDO命令

```

StarFive #
StarFive # mdio list
ethernet@16030000:
ethernet@16040000:
StarFive #
StarFive #

```

您可以使用以上命令来检查开发板上的PHY是否已准备好进行数据通信。

### 5.4. PING - 数字环回

在确认对PHY的访问已准备就绪之后，您可以使用PING命令启动数字环回，以发送和接收PING数据包。

要启动测试，请运行命令 `ping $ipaddr`。

下图显示了执行该命令的返回示例。



图 5-3 Ping命令

```
StarFive #  
StarFive # ping 192.168.120.72  
ethernet@16030000 waiting for PHY auto negotiation to complete.... done  
Using ethernet@16030000 device  
host 192.168.120.72 is alive  
StarFive # ping 192.168.120.72  
Using ethernet@16030000 device  
host 192.168.120.72 is alive  
StarFive #
```



## 6. 调试方法

### 6.1. 通用调试命令

以下提供了通常用于调试以太网连接的命令示例。

- 检查以太网设备信息：

- 检查适配器状态：

```
ifconfig eth0
```

- 检查数据包发送和接收统计信息：

```
cat /proc/net/dev
```

- 检查当前速度：

```
cat /sys/class/net/eth0/speed
```

- 启用或禁用以太网设备。

- 启用：

```
ifconfig eth0 up
```

- 禁用：

```
ifconfig eth0 down
```

- 配置以太网设备。

- 配置动态IP地址：

```
ifconfig eth0 192.168.1.101
```

- 配置MAC地址：

```
ifconfig eth0 hw ether 00:11:22:aa:bb:cc
```

- 自动获取IP地址：

```
udhcpc -i eth0
```

- 设置PHY模式：（设置100 M速度，启用全双工和自动协商）

```
ethtool -s eth0 speed 100 duplex full autoneg on
```

- 通用测试命令：

◦ 连接测试：

```
ping 192.168.1.101
```

◦ 吞吐量测试：



**注：**

在测试前，请确保您已经启用了内核菜单中的iperf工具。

▪ TCP吞吐量测试

服务器端：

```
iperf3 -s -i 1
```

客户端：

```
iperf3 -c 192.168.1.101 -i 1 -t 60 -P 4
```

▪ UDP吞吐量测试

服务器端：

```
iperf3 -s -u -i 1
```

客户端：

```
iperf3 -c 192.168.1.101 -u -b 100M -i 1 -t 60 -P 4
```

## 6.2. 一般故障排除步骤

本节介绍了一些处理一般故障的步骤。

### 软件故障排除

以下列表显示了软件问题的一般故障排除步骤。

1. 验证PHY模式是否配置正确。
2. 验证时钟设置是否配置正确。
3. 验证GPIO设置是否配置正确，例如，IO MUX（多路复用）功能、驱动能力、上拉和下拉设置等。
4. 验证PHY复位设置是否配置正确。
5. 使用以下命令来验证在“eth0”上发送和接收数据包的状态。

```
cat /proc/net/dev
```

## 硬件故障排除

以下列表显示了硬件问题的一般故障排除步骤。

1. 验证PHY电源vcc-ephy是否正常工作。
2. 验证时钟波形是否良好。



## 7. 已知问题

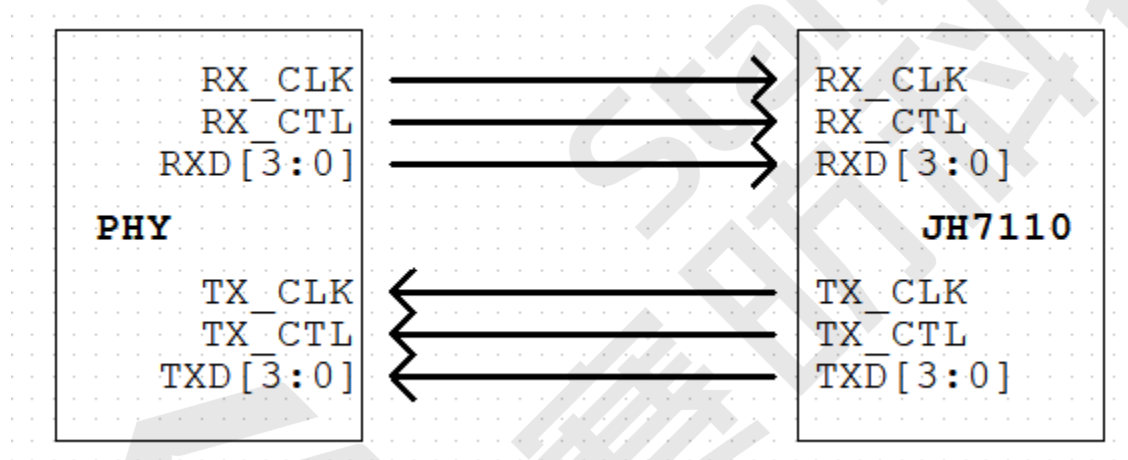
### 7.1. 以太网GMAC仅支持RGMII

昉·惊鸿7110仅支持以太网GMAC连接的RGMII模式。由于此限制，昉·惊鸿7110有以下布局要求。

#### 7.1.1. 仅支持1,000 M

如果您只需支持1,000 M模式，可按照以下要求设计布局。

图 7-1 GMAC仅支持1,000 M



布局要求：

- RX/TX的线长不能超过6,000 mil。
- 匹配RXD[3:0]信号组与RX\_CTL和RX\_CLK信号的线长在100 mil以内。匹配TXD[3:0]信号组与TX\_CTL和TX\_CLK信号的线长在100 mil以内。
- 数据和时钟通道的布线应保持一个完整的参考平面。

#### 7.1.2. 自动协商

如果您需要支持10/100/1,000 M模式的自动协商，您需要了解以下限制，然后按照以下要求设计布局。



**重要：**

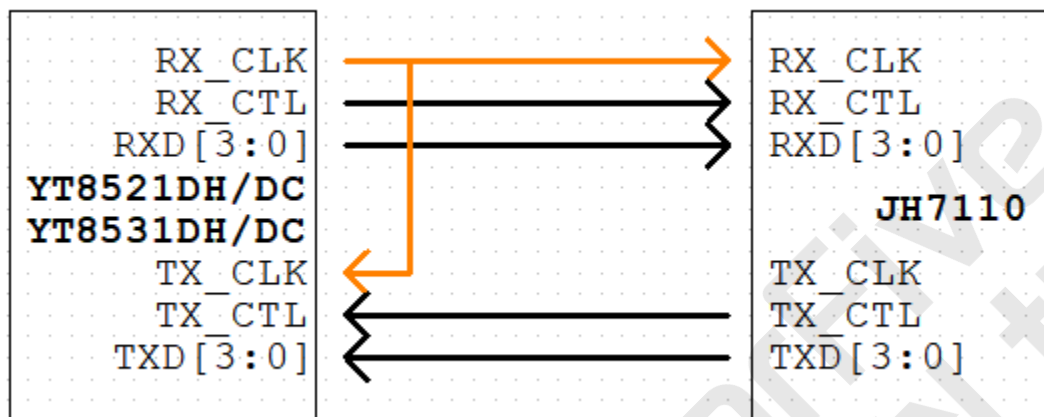
在自动协商模式下，仅支持以下PHY模式：



- YT8521DH/DC
- YT8531DH/DC

另外，您需要将PHY的RX\_CLK连接到其TX\_CLK，如下图中的橙色线所示。

图 7-2 GMAC 10 M/100 M/1,000 M自动协商



GMAC0布局要求:

- TX\_CLK到RX\_CLK的线长不能超过500 mil。
- RX和TX的线长不能超过4,300 mil。
- 匹配RXD[3:0]信号组与RX\_CTL和RX\_CLK信号的线长在100 mil以内。
- 匹配RXD[3:0]信号组与RX\_CTL和RX\_CLK信号的线长在100 mil以内。
- 数据和时钟通道的布线应保持一个完整的参考平面。

GMAC1布局要求:

- TX\_CLK到RX\_CLK的线长不能超过500 mil。
- RX\_CLK的线长不能超过4,000 mil。匹配RXD[3:0]信号组与RX\_CTL和RX\_CLK信号的线长在100 mil以内。
- TX\_CLK的线长比RX\_CLK的长2,000 mil。匹配RXD[3:0]信号组与RX\_CTL和RX\_CLK信号的线长在100 mil以内。
- 数据和时钟通道的布线应保持一个完整的参考平面。