Software SDK Developer Guide for DMA

VisionFive 2
Version: 1.0
Date: 2022/11/10
Doc ID: JH7110-DGEN-007

Legal Statements

Important legal notice before reading this documentation.

PROPRIETARY NOTICE

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

Contents

List of Tables

List of Figures

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 DMA module of the StarFive next generation SoC platform - JH7110.

Audience

This document mainly serves the DMA 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.

1. Introduction

Direct Memory Access (DMA) Engine is the DMA driver framework in the Linux kernel. To solve the problem caused by the emergence of different types of DMA devices and drivers, DMA, as a brand new driver framework, is promoted to enable developers to focus on their code instead of worrying about hardware details.

The driver framework provides a standard API structure for code reuse, achieves asynchronous data transmission, and reduces device workload.

1.1. Function Introduction

The JH7110 DMA engine provides unified interfaces for users and assigns dedicated DMA interfaces for different work modes. It releases developers from worrying about hardware interfaces.

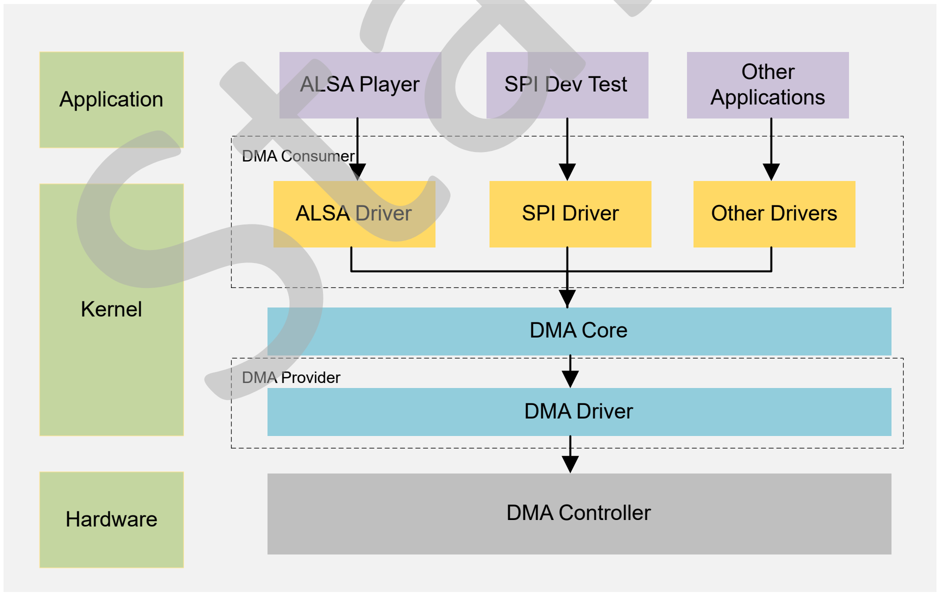
The SGDMA (DMA) of JH7110 has the following features.

- Support general DMA operations and Scatter Gather DMA
- Support up to 4 gather&scatter channels
- Support up to 32 requests
- Support AXI4 bus width of 64-bit
- Support memory to memory, memory to peripheral, peripheral to memory and peripheral to peripheral transfer
- Can generate both interrupts for transmission completion and for transmission failure due to errors
- Support arbitration - fixed and round robin
- Provide status information of each transfer for each channel in the status register

1.2. Block Diagram

The following figure shows the block diagram of the DMA engine.

Figure 1-1 Block Diagram



The DMA framework is mainly divided into the following 3 parts, the DMA provider, the DMA consumer and the DMA core. The following list provides some examples from the DMA consumer's perspective:

- **ALSA Player**: *Advanced Linux Sound Architecture (ALSA)* provides audio and MIDI functionality to the Linux operating system. The ALSA player communicates with the DMA core via the ALSA driver.
- **SPI Dev Test**: The SPI development test application communicates with the DMA core via the SPI driver.
- **Other Applications**. The other applications communicate with the DMA core via their associated driver programs.

From the DMA provider's perspective, the **DMA Core** provides public interfaces for the DMA consumers to access the hardware DMA channel.

1.3. Source Code Structure

The following code block shows the source code structure of the DMA engine.

```
-- drivers
-- dma
-- dw-axi-dmac
   |-- Makefile
   |-- dw-axi-dmac-platform.c
   |-- dw-axi-dmac.h
-- dw-axi-dmac-starfive
   |-- Makefile
   |-- dw-axi-dmac-starfive-misc.c
   |-- starfive_dmaengine_memcpy.c
```

1.4. Device Tree Overview

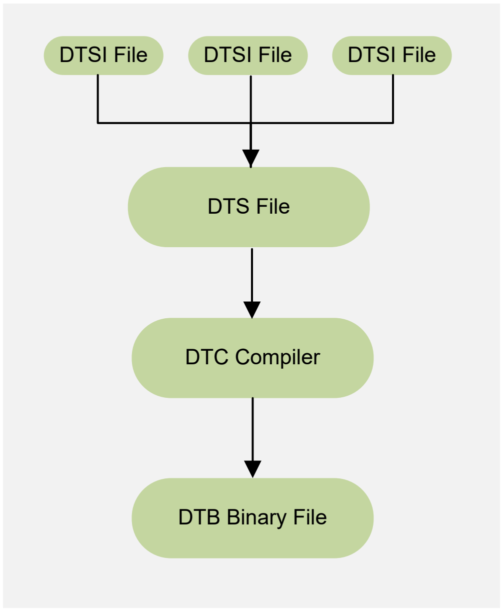
Since Linux 3.x, device tree is introduced as a data structure and language to describe hardware configuration. It is a system-readable description of hardware settings so that the operating system doesn't have to hard code details of the machine.

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 1-2 Device Tree Workflow



1.5. 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

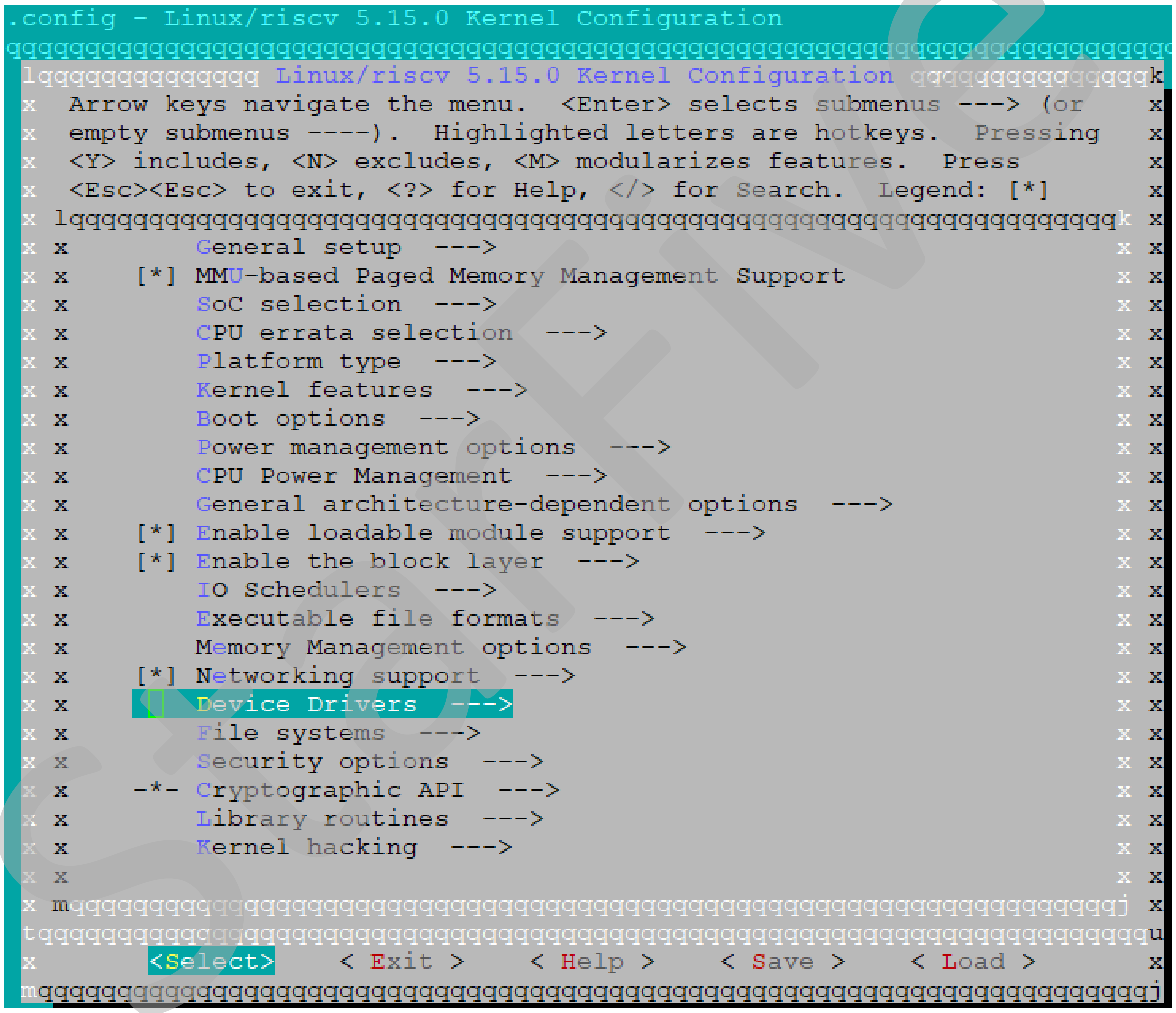
Follow the steps below to enable the kernel configuration for DMA.

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.

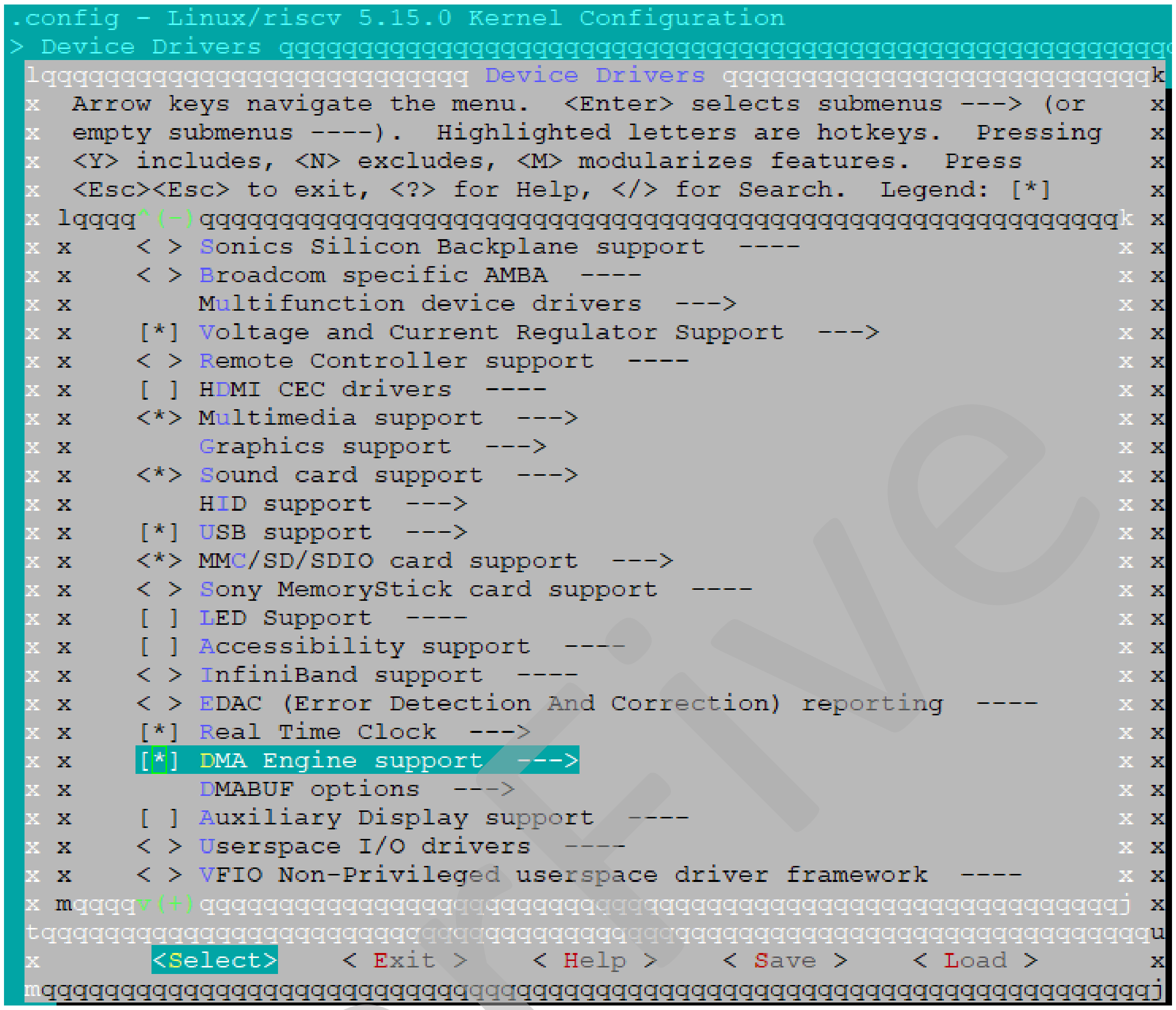
Figure 2-1 Device Drivers



```
.config - Linux/riscv 5.15.0 Kernel Configuration
 lqqqqqqqqqqqqqqq Linux/riscv 5.15.0 Kernel Configuration qqqqqqqqqqqqqqqk
 x  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or     x
 x  empty submenus ----).  Highlighted letters are hotkeys.  Pressing    x
 x  <Y> includes, <N> excludes, <M> modularizes features.  Press         x
 x  <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]       x
 x lqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk x
 x x          General setup  --->                                    x x
 x x      [*] MMU-based Paged Memory Management Support                x x
 x x          SoC selection  --->                                    x x
 x x          CPU errata selection  --->                             x x
 x x          Platform type  --->                                    x x
 x x          Kernel features  --->                                  x x
 x x          Boot options  --->                                     x x
 x x          Power management options  --->                         x x
 x x          CPU Power Management  --->                             x x
 x x          General architecture-dependent options  --->           x x
 x x      [*] Enable loadable module support  --->                   x x
 x x      [*] Enable the block layer  --->                           x x
 x x          IO Schedulers  --->                                    x x
 x x          Executable file formats  --->                          x x
 x x          Memory Management options  --->                        x x
 x x      [*] Networking support  --->                               x x
 x x          Device Drivers  --->                                   x x
 x x          File systems  --->                                     x x
 x x          Security options  --->                                 x x
 x x      -*- Cryptographic API  --->                                x x
 x x          Library routines  --->                                 x x
 x x          Kernel hacking  --->                                   x x
 x x                                                                 x x
 x mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj x
 tqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
 x        <Select>    < Exit >    < Help >    < Save >    < Load >     x
 mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj
```

3. Select the **DMA Engine support** option and enter the next level.

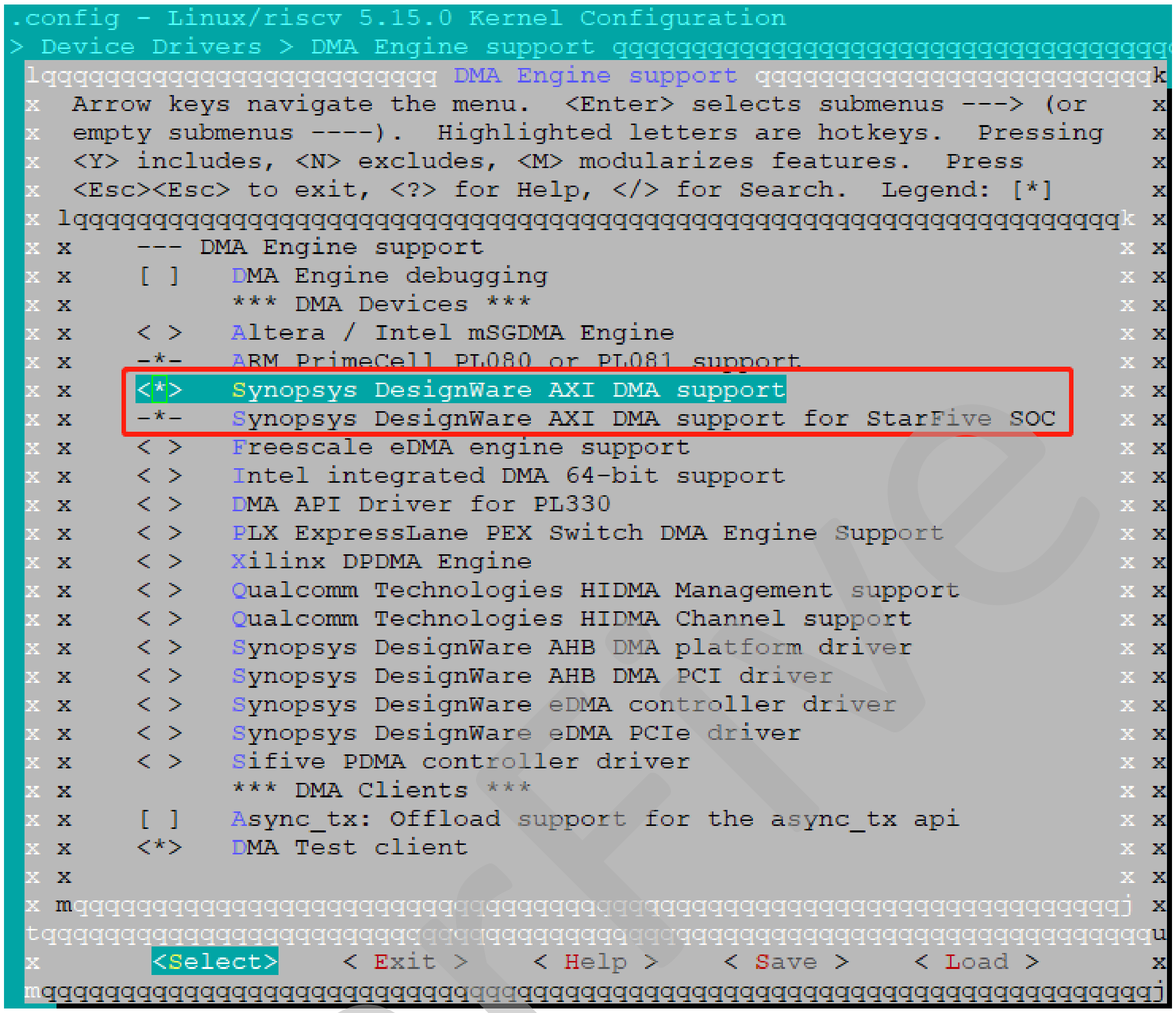
Figure 2-2 DMA Engine Support



```
.config - Linux/riscv 5.15.0 Kernel Configuration
> Device Drivers qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
 lqqqqqqqqqqqqqqqqqqqqqqqqqqq Device Drivers qqqqqqqqqqqqqqqqqqqqqqqqqqqk
 x  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or   x
 x  empty submenus ----).  Highlighted letters are hotkeys.  Pressing  x
 x  <Y> includes, <N> excludes, <M> modularizes features.  Press       x
 x  <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]     x
 x lqqqq^(-)qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk x
 x x     < > Sonics Silicon Backplane support  ----                  x x
 x x     < > Broadcom specific AMBA  ----                            x x
 x x         Multifunction device drivers  --->                      x x
 x x     [*] Voltage and Current Regulator Support  --->             x x
 x x     < > Remote Controller support  ----                         x x
 x x     [ ] HDMI CEC drivers  ----                                  x x
 x x     <*> Multimedia support  --->                                x x
 x x         Graphics support  --->                                  x x
 x x     <*> Sound card support  --->                                x x
 x x         HID support  --->                                       x x
 x x     [*] USB support  --->                                       x x
 x x     <*> MMC/SD/SDIO card support  --->                          x x
 x x     < > Sony MemoryStick card support  ----                     x x
 x x     [ ] LED Support  ----                                       x x
 x x     [ ] Accessibility support  ----                             x x
 x x     < > InfiniBand support  ----                                x x
 x x     < > EDAC (Error Detection And Correction) reporting  ----   x x
 x x     [*] Real Time Clock  --->                                   x x
 x x     [*] DMA Engine support  --->                                x x
 x x         DMABUF options  --->                                    x x
 x x     [ ] Auxiliary Display support  ----                         x x
 x x     < > Userspace I/O drivers  ----                             x x
 x x     < > VFIO Non-Privileged userspace driver framework  ----    x x
 x mqqqqv(+)qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj x
 tqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
 x        <Select>    < Exit >    < Help >    < Save >    < Load >     x
 mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj
```

4. In the **DMA Engine support** menu, select the **DesignWare DMA Support** and the **DesignWare DMA Support for StarFive SoC** options.

Figure 2-3 DesignWare DMA Support



```
.config - Linux/riscv 5.15.0 Kernel Configuration
> Device Drivers > DMA Engine support qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
 lqqqqqqqqqqqqqqqqqqqqqqqqqq DMA Engine support qqqqqqqqqqqqqqqqqqqqqqqqqk
 x  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or    x
 x  empty submenus ----).  Highlighted letters are hotkeys.  Pressing   x
 x  <Y> includes, <N> excludes, <M> modularizes features.  Press        x
 x  <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]      x
 x lqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk x
 x x     --- DMA Engine support                                      x x
 x x     [ ]   DMA Engine debugging                                  x x
 x x           *** DMA Devices ***                                   x x
 x x     < >   Altera / Intel mSGDMA Engine                          x x
 x x     -*-   ARM PrimeCell PL080 or PL081 support                  x x
 x x     <*>   Synopsys DesignWare AXI DMA support                   x x
 x x     -*-   Synopsys DesignWare AXI DMA support for StarFive SOC  x x
 x x     < >   Freescale eDMA engine support                         x x
 x x     < >   Intel integrated DMA 64-bit support                   x x
 x x     < >   DMA API Driver for PL330                              x x
 x x     < >   PLX ExpressLane PEX Switch DMA Engine Support         x x
 x x     < >   Xilinx DPDMA Engine                                   x x
 x x     < >   Qualcomm Technologies HIDMA Management support        x x
 x x     < >   Qualcomm Technologies HIDMA Channel support           x x
 x x     < >   Synopsys DesignWare AHB DMA platform driver           x x
 x x     < >   Synopsys DesignWare AHB DMA PCI driver                x x
 x x     < >   Synopsys DesignWare eDMA controller driver            x x
 x x     < >   Synopsys DesignWare eDMA PCIe driver                  x x
 x x     < >   Sifive PDMA controller driver                         x x
 x x           *** DMA Clients ***                                   x x
 x x     [ ]   Async_tx: Offload support for the async_tx api        x x
 x x     <*>   DMA Test client                                       x x
 x x                                                                 x x
 x mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj x
 tqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
 x        <Select>    < Exit >    < Help >    < Save >    < Load >      x
 mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj
```

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

2.2. Device Tree Configuration

In the DTSI file, the system provides general configuration information for DMA controller, see the following code block for an example.

Note:
You are not recommended to change the following configuration. If you need to change, make sure you contact your system maintenance engineer in advance.

```
dma: dma-controller@16050000 {
compatible = "starfive,jh7110-dma", "snps,axi-dma-1.01a";
reg = <0x0 0x16050000 0x0 0x10000>;
clocks = <&clkgen JH7110_DMA1P_CLK_AXI>,
        <&clkgen JH7110_DMA1P_CLK_AHB>;
clock-names = "core-clk", "cfgr-clk";
resets = <&rstgen RSTN_U0_DW_DMA1P_AXI>,
          <&rstgen RSTN_U0_DW_DMA1P_AHB>;
reset-names = "rst-axi", "rst-ahb";
interrupts = <73>;
dma-cells = <2>;
dma-channels = <4>;
snps,dma-masters = <1>;
snps,data-width = <3>;
snps,num-hs-if = <56>;
snps,block-size = <65536 65536 65536 65536>;
snps,priority = <0 1 2 3>;
snps,axi-max-burst-len = <16>;
status = "disabled";
};
```

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 "`0x16050000`" and range "`0x10000`".
- **clocks**: The clocks used by the DMA module.
- **clock-names**: The names of the above clocks.
- **resets**: The reset signals used by the DMA module.
- **reset-names**: The names of the above reset signals.
- **interrupts**: Hardware interrupt ID.
- **dma-cells**: The field is used to configure DMA cells via DTS configuration.
- **dma-channels**: The supported DMA channels.
- **dma-masters**: The supported AXI master interfaces.
- **data-width**: The field is used to configure data bus width (Unit: Byte). The default value is 3.
- **num-hs-if**: The total count of hardware handshakes, maximum 56 handshakes.
- **priority**: The priorities of the 4 channels mentioned in the **dma-channels** field.
- **axi-max-burst-len**: The maximum data length allowed in one single transmission.
- **status**: The work status of the DMA module. To enable the module, set this bit as "`okay`" or to disable the module, set this bit as "`disabled`".

2.3. DMA Requester Device Tree Configuration

In the DTSI file, the DMA controller requester needs to include the general configuration information of the used DMA module, see the following code block for an example.

```
tdm: tdm@10090000 {
......
dmas = <&dma 20 1>, <&dma 21 1>;
dma-names = "rx", "tx";
......
};
```

The following list provides explanations for the parameters.

- **dmas**: The field contains 3 parameters:
    - **&dma**: The referenced DMA node.
    - **20/21**: The DMA channel ID.
    - **1**: The burst transaction length, "`1`" indicates to transfer 4 words at a time.

  For example, in the code `<&dma 20 1>`, `20` indicates channel number; `1` indicates burst length is 4.
- **dma-names**: The names of the DMA channels, including "rx" for the receiver channel and "tx" for the transceiver channel. The names can also be found in the DMA driver.

2.3.1. Using DMA in TDM

The following code block provides an example of the TDM device tree in the case when DMA is used in the TDM driver.

```
tdm: tdm@10090000 {
                        compatible = "starfive,jh7110-tdm";
                        reg = <0x0 0x10090000 0x0 0x1000>;
                        reg-names = "tdm";
                        clocks = <&clkgen JH7110_AHB0>,
                                 <&clkgen JH7110_TDM_CLK_AHB>,
                                 <&clkgen JH7110_APB0>,
                                 <&clkgen JH7110_TDM_CLK_APB>,
                                 <&clkgen JH7110_TDM_INTERNAL>,
                                 <&tdm_ext>,
                                 <&clkgen JH7110_TDM_CLK_TDM>,
                                 <&clkgen JH7110_MCLK_INNER>;
                        clock-names = "clk_ahb0", "clk_tdm_ahb",
                                      "clk_apb0", "clk_tdm_apb",
                                      "clk_tdm_internal", "clk_tdm_ext",
                                      "clk_tdm", "mclk_inner";
                        resets = <&rstgen RSTN_U0_TDM16SLOT_AHB>,
                                 <&rstgen RSTN_U0_TDM16SLOT_APB>,
                                 <&rstgen RSTN_U0_TDM16SLOT_TDM>;
                        reset-names = "tdm_ahb", "tdm_apb", "tdm_rst";
                        dmas = <&dma 20 1>, <&dma 21 1>;
                        dma-names = "rx","tx";
                        #sound-dai-cells = <0>;
                        status = "disabled";
            };
```

3. Interface Description

JH7110 provides the following application interfaces for DMA engine.

3.1. dma_request_channel

The interface has the following parameters.

- **Synopsis**:

```
struct dma_channel *dma_request_channel(struct device *dev, const char *name)
```

- **Description**: The interface is used to request an available DMA channel.
- **Parameter**:
  - **dev**: The DMA applicant device.
  - **name**: Name of the DMA channel. The value corresponds with the value of **dma-names** in .DMA Requester Device Tree Configuration *(on page 14)*
- **Return**:
  - **Success**: A pointer to the handle of DMA channel.
  - **Failure**: NULL.

3.2. dma_release_channel

The interface has the following parameters.

- **Synopsis**:

```
void dma_release_channel(struct dma_chan *chan)
```

- **Description**: The interface is used to release a specified DMA channel.
- **Parameter**:
  - **chan**: The pointer to the handle of DMA channel to be released.
- **Return**: Void.

3.3. dmaengine_slave_config

The interface has the following parameters.

- **Synopsis**:

```
int dmaengine_slave_config(struct dma_chan *chan, struct dma_slave_config *config)
```

- **Description**: The interface is used to configure the device (slave) information of a DMA channel.
- **Parameter**:
  - **chan**: The pointer to the handle of the relevant DMA channel.
  - **config**: Parameters of the slave in the DMA channel.
- **Return**:
  - **Success**: 0.
  - **Failure**: Error code.

3.4. dmaengine_prep_dma_cyclic

The interface has the following parameters.

- **Synopsis**:

```
static inline struct dma_async_tx_descriptor *
        dmaengine_prep_dma_cyclic(struct dma_chan *chan,
                        dma_addr_t buf_addr,
                        size_t buf_len,
                        size_t period_len,
                        enum dma_transfer_direction dir,
                        unsigned long flags)
```

- **Description**: The interface is used to prepare a transmission of the ring buffer. The interface is usually used in audio scenarios. In a DMA transmission with a certain length, every time when a certain amount of bytes (defined in **period_len**) has been transmitted, the system will request the callback function of transmission completion.
- **Parameter**:
    - **chan**: The relevant DMA channel.
    - **buf_addr**: Destination address.
    - **buf_len**: The length of the transmission.
    - **period_len**: When a certain number of data bytes are transferred, the callback function that completes the transfer will be executed.

      Note:
      buf_len must be integral multiple of **period_len**.

    - **dir**: Transmission direction. For example:
        - **DMA_MEM_TO_DEV**: From memory to device.
        - **DMA_DEV_TO_MEM**: From device to memory.
    - **flags**: The macro definitions started with "DMA_PREP_", which provides additional information for the *DMA Controller (DMAC)* driver. For example:
        - **DMA_PREP_INTERRUPT**: The flag is used to inform DMAC that an interrupt will be generated after the transmission, so that DMAC will use the callback function from the client.
        - **DMA_PREP_FENCE**: The flag is used to inform DMAC that all the later transmissions rely on the success of the current transmission, so that DMAC will prioritize the DMA transmissions accordingly.
- **Return**:
    - **Success**: A pointer to the transmission descriptor.
    - **Failure**: NULL.

3.5. dmaengine_prep_slave_sg

The interface has the following parameters.

- **Synopsis**:

```
static inline struct dma_async_tx_descriptor *
        dmaengine_prep_slave_sg(struct dma_chan *chan,
                                struct scatterlist *sgl,
                                unsigned int sg_len,
                                enum dma_transfer_direction dir,
                                unsigned long flags)
```

- **Description**: The interface is used to prepare a DMA transmission for the scattered buffer.

- **Parameter**:
    - **chan**: The relevant DMA channel.
    - **sgl**: The address of the scattered transmission list. The list should be created before a scattered transmission.
    - **dir**: Transmission direction. For example:
        - **DMA_MEM_TO_DEV**: From memory to device.
        - **DMA_DEV_TO_MEM**: From device to memory.
    - **flags**: The macro definitions started with "DMA_PREP_", which provides additional information for the *DMA Controller (DMAC)* driver. For example:
        - **DMA_PREP_INTERRUPT**: The flag is used to inform DMAC that an interrupt will be generated after the transmission, so that DMAC will use the callback function from the client.
        - **DMA_PREP_FENCE**: The flag is used to inform DMAC that all the later transmissions rely on the success of the current transmission, so that DMAC will prioritize the DMA transmissions accordingly.
- **Return**:
    - **Success**: A pointer to the transmission descriptor.
    - **Failure**: NULL.

3.6. dmaengine_submit

The interface has the following parameters.

- **Synopsis**:

```
dma_cookie_t dmaengine_submit(struct dma_async_tx_descriptor *desc)
```

- **Description**: The interface is used to submit a prepared transmission.
- **Parameter**:
    - **desc**: The pointer to the description of the transmission.
- **Return**:
    - **Success**: A cookie greater than 0.
    - **Failure**: Error code.

3.7. dma_async_issue_pending

The interface has the following parameters.

- **Synopsis**:

```
void dma_async_issue_pending(struct dma_chan *chan)
```

- **Description**: The interface is used to start the transmission in a certain channel.
- **Parameter**:
    - **chan**: The pointer to the handle of the relevant DMA channel.
- **Return**: Void.

3.8. dmaengine_terminate_all

The interface has the following parameters.

- **Synopsis**:

```
void dma_async_issue_pending(struct dma_chan *chan)
```

- **Description**: The interface is used to stop the DMA transmission in a certain channel.
- **Parameter**:
    - **chan**: The pointer to the handle of the relevant DMA channel.
- **Return**:
    - **Success**: 0.
    - **Failure**: Error code.

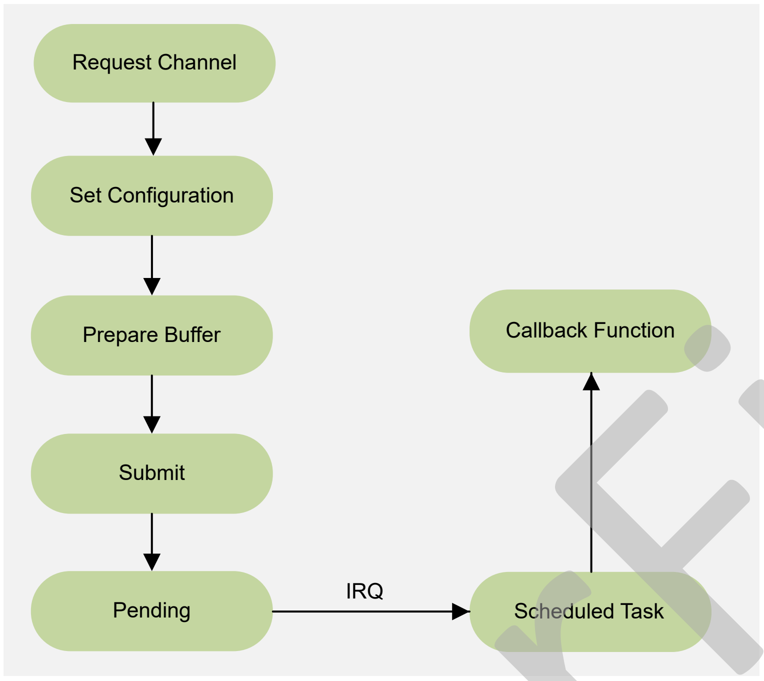
4. Work Process

The section provides the general work procedure and notes for using the DMA Engine.

4.1. General Work Process

The following figure describes the general procedure of using the DMA Engine.

Figure 4-1 General Work Process



4.2. Work Notices

Make sure you know the following before using the DMA Engine.

- The callback function does not support sleep and scheduling.
- The time for the callback function should not be too long.
- Pending means the transmission will hold until a soft interrupt, except for the cyclic mode.

5. Use Example

The following code block provides a DMA use case.

```
/* Request a callback function for transmission complete  */
static void dma_complete_func(void *arg)
{
    struct completion *done = (struct completion *)arg;
        complete(done);
}

struct dma_chan *chan;
dma_cap_mask_t mask;
dma_cookie_t cookie;
struct dma_slave_config config;
struct dma_tx_state state;
struct dma_async_tx_descriptor *tx = NULL;
void *src_buf;
dma_addr_t src_dma;

    dma_cap_zero(mask);
    dma_cap_set(DMA_SLAVE, mask);
    dma_cap_set(DMA_CYCLIC, mask);

/* Request an available channel */
chan = dma_request_channel(dt->mask, NULL, NULL);
    if (!chan){
        return -EINVAL;
    }

src_buf = kmalloc(1024*4, GFP_KERNEL);
    if (!src_buf) {
        dma_release_channel(chan);
        return -EINVAL;
    }

/* Use DMA to visit the mapping address */
src_dma = dma_map_single(NULL, src_buf, 1024*4, DMA_TO_DEVICE);

    config.direction = DMA_MEM_TO_DEV;
    config.src_addr = src_dma;
    config.dst_addr = 0x01c;
    config.src_addr_width = DMA_SLAVE_BUSWIDTH_2_BYTES;
    config.dst_addr_width = DMA_SLAVE_BUSWIDTH_2_BYTES;
    config.src_maxburst = 1;
    config.dst_maxburst = 1;

    dmaengine_slave_config(chan, &config);

tx = dmaengine_pre_dma_cyclic(chan, scr_dma, 1024*4, 1024, DMA_MEM_TO_DEV, DMA_PREP_INTERRUPT | DMA_CTRL_ACK);

/* Set the callback function for transmission complete */
tx->callback = dma_complete_func;

/* submit and start the transmission */
cookie = dmaengine_submit(tx);
dma_async_issue_pending(chan);
```