StarFive 赛昉科技

Software SDK Developer Guide for UART

VisionFive 2
Version: 1.0
Date: 2022/11/10
Doc ID: JH7110-DGEN-004

Legal Statements

Important legal notice before reading this documentation.

PROPRIETARY NOTICE

Information in this document is provided "as is," with all faults. Contents may be periodically updated or revised due to product development. Shanghai StarFive Technology Co., Ltd. (hereinafter "StarFive") reserves the right to make changes without further notice to any products herein.

StarFive expressly disclaims all warranties, representations, and conditions of any kind, whether express or implied, including, but not limited to, the implied warranties or conditions of merchantability, fitness for a particular purpose, and non-infringement.

StarFive does not assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation indirect, incidental, special, exemplary, or consequential damages.

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 UART of the StarFive next generation SoC platform - JH7110.

Audience

This document mainly serves the UART 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....5
List of Figures....6
Legal Statements....ii
Preface....iii
1. Introduction....7
1.1. Function Layer....7
1.2. Device Tree Overview....7
1.3. Source Code Structure....8
2. Configuration....9
2.1. Kernel Menu Configuration....9
2.2. Device Tree Source Code....11
2.3. Device Tree Configuration....12
2.4. Board Level Configuration....12
2.5. Set Other UART as Print Console....13
3. Interface Description....14
3.1. Enable or Disable the Serial Port....14
3.2. Configure Serial Port Attributes....14
3.2.1. tcgetattr....14
3.2.2. tcsetattr....14
3.2.3. cfgetispeed....14
3.2.4. cfgetospeed....15
3.2.5. cfsetispeed....15
3.2.6. cfsetospeed....15
3.2.7. cfsetspeed....15
3.2.8. tcflush....15
4. Example Use Case....16

List of Tables

Table 0-1 Revision History.....iii

List of Figures

Figure 1-1 Function Layer..... 7
Figure 1-2 Device Tree Workflow..... 8
Figure 2-1 Device Drivers..... 9
Figure 2-2 Character Devices..... 10
Figure 2-3 Serial Drivers..... 10
Figure 2-4 Support for 8250..... 11

1. Introduction

Universal Asynchronous Receiver-Transmitter (UART) uses a single data transmission line to send data to destination.

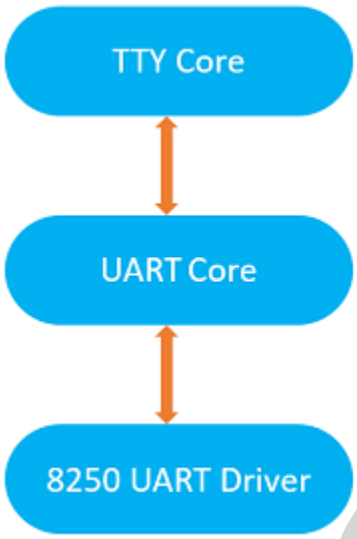
UART can be used not only to output log data for system debug, but also to complete short-distance communication. It is an interface with practical use in embedded systems.

1.1. Function Layer

The UART driver of the JH7110 SoC Platform has the following layers.

- TTY Core: TTY refers to the TeleType and/or the TeleType Writer. It registers and manages all TTY devices in the core.
- UART Core: It provides a set of API for the UART driver for registration of devices and drivers.
- 8250 UART Driver: It serves as a platform for the initialization and data communication for the JH7110 SoC Platform.

Figure 1-1 Function Layer



1.2. Device Tree Overview

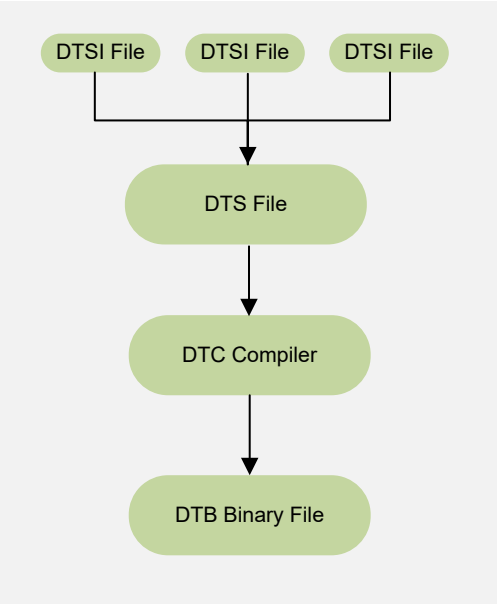
Since Linux 3.x, device tree is introduced as a data structure and language to describe hardware configuration. It is a system-readable description of hardware settings so that the operating system doesn't have to hard code details of the machine.

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



1.3. Source Code Structure

The following code block shows the source code tree structure of the UART driver.

```
linux 5.15
|-- include
|    |-- linux
|    |    |-- serial_8250.h
|    |    |-- serial_core.h
|-- driver
|    |-- tty
|    |    |-- serial
|    |    |    |-- serial_core.c
|    |    |    |-- 8250
|    |    |    |    |-- 8250_dw.c
|    |    |    |    |-- 8250_core.c
|    |    |    |    |-- 8250.h
```

2. Configuration

2.1. Kernel Menu Configuration

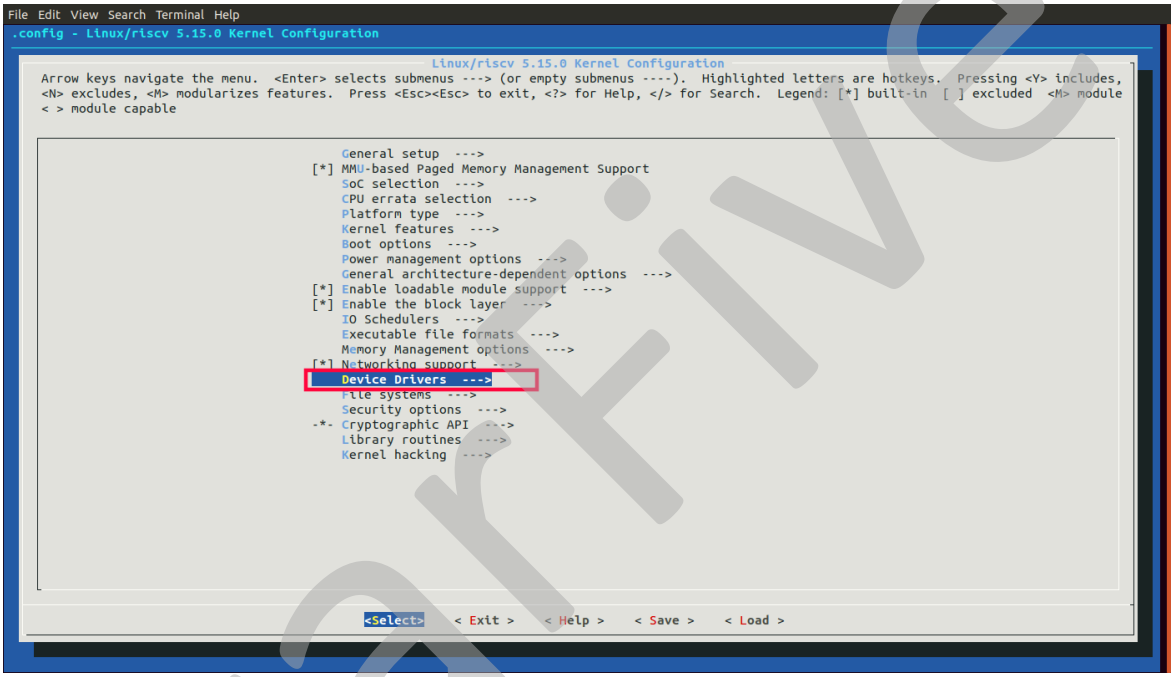
Follow the steps below to enable the kernel configuration for UART.

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

```
make linux-menuconfig
```

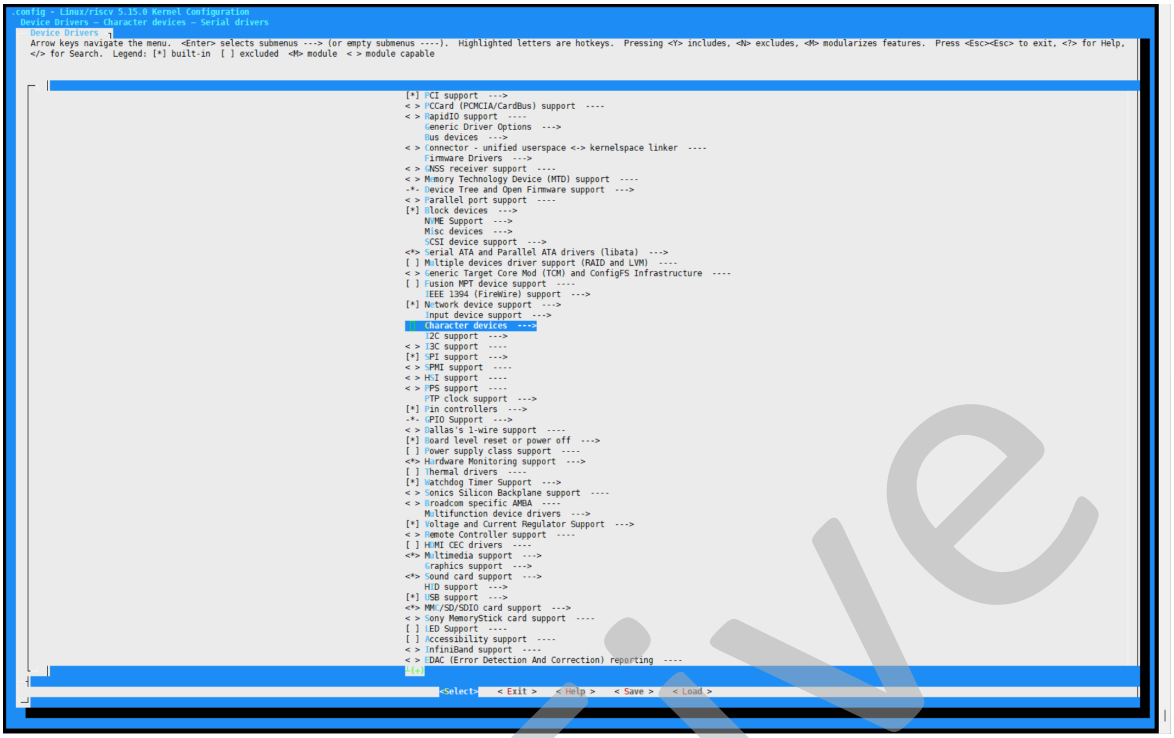
2. Enter the **Device Drivers** menu.

Figure 2-1 Device Drivers



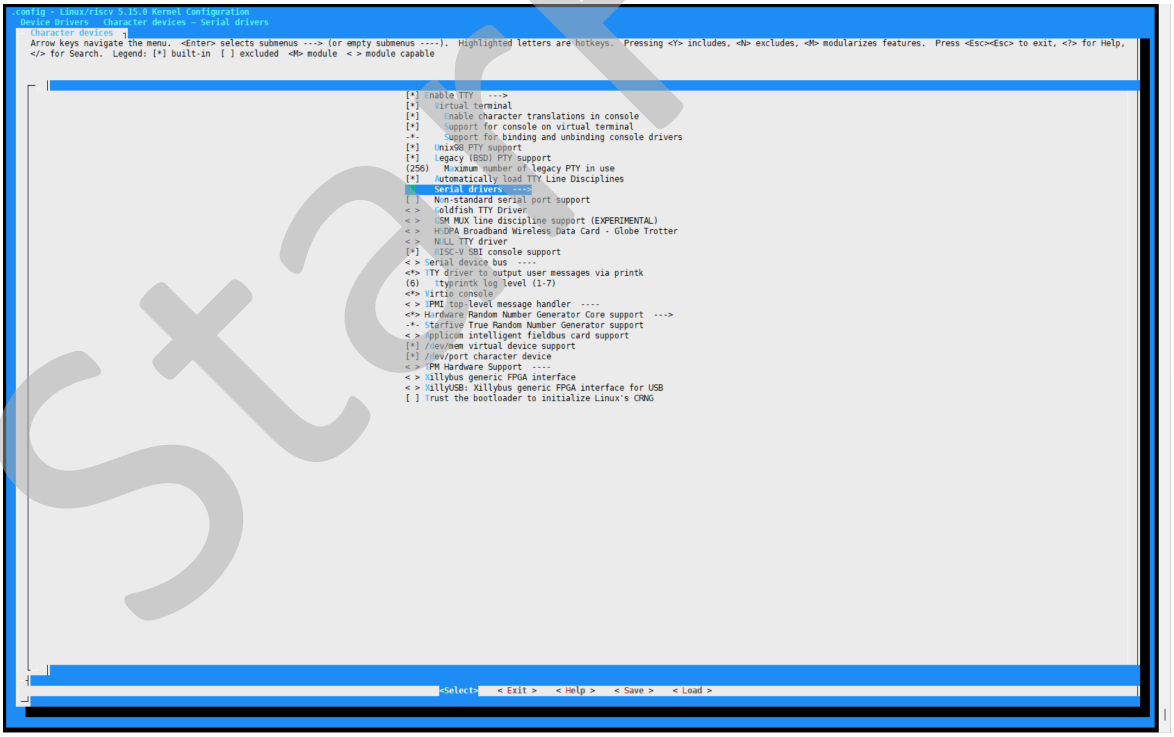
3. Enter the **Character devices** menu.

Figure 2-2 Character Devices



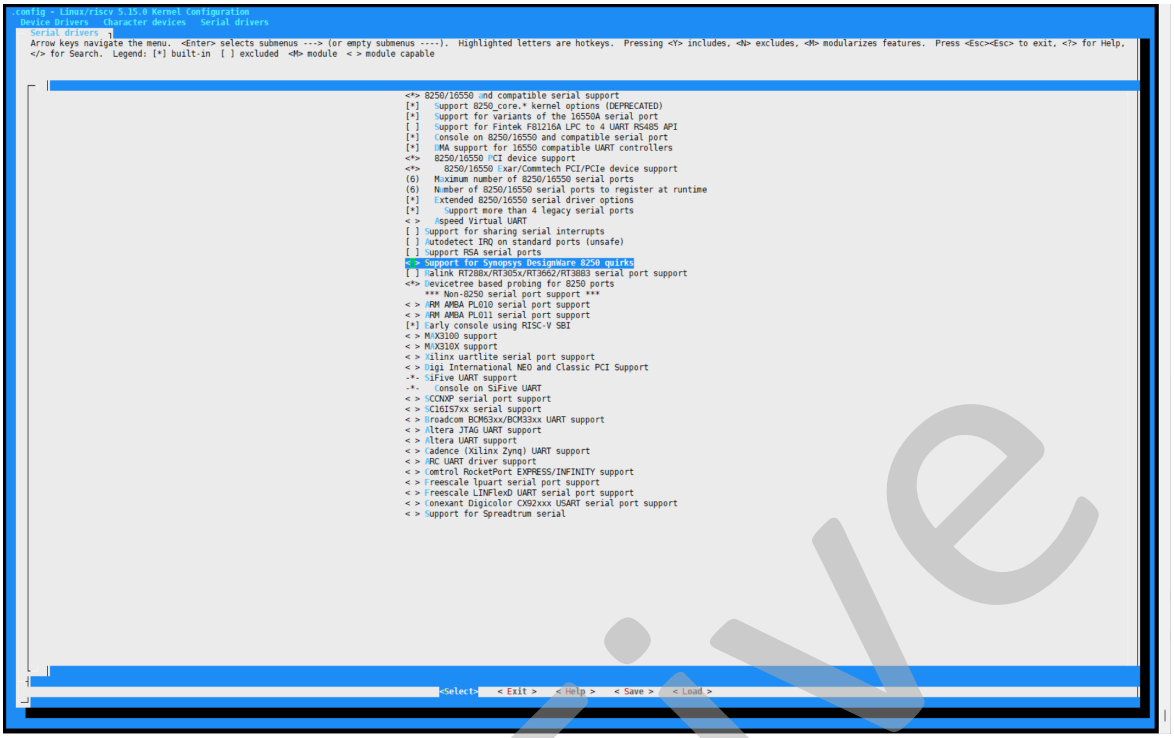
4. Enter the **Serial drivers** menu.

Figure 2-3 Serial Drivers



5. Select the **Support for Synopsys Designware 8250 quirks** option.

Figure 2-4 Support for 8250



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

2.2. 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
```

2.3. Device Tree Configuration

For Linux 5.15, UART has the following configuration for a general UART Controller:

```
uart0: serial@10000000 {
        compatible = "snps,dw-apb-uart";
        reg = <0x0 0x10000000 0x0 0xl0000>;
        reg-io-width = <4>;
        reg-shift = <2>;
        clocks = <&clkgen JH7110_UART0_CLK_CORE>,
                 <&clkgen JH7110_UART0_CLK_APB>;
        clock-names = "baudclk", ”apb_pclk";
        resets = <&rstgen RSTN_U0_DW_UART_APB>,
                 <&rstgen RSTN_U0_DW_UART_CORE>;
        interrupts = <32>;
        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 "`0x10000000`" and range "`0x10000`". Make sure you do not change the 2 bits after it, **reg-io-width** and **reg-shift**.
- **clocks**: The clocks used by the UART module.
- **clock-names**: The names of the above clocks.
- **resets**: The reset signals used by the UART module.
- **interrupts**: Hardware interrupt ID.
- **status**: The work status of the UART module. To enable the module, set this bit as "`okay`" or to disable the module, set this bit as "`disabled`".

You can configure each UART controller in the device tree. A UART node represents a UART controller. You need to specify an alias for the UART node so that you can identify it from the others.

2.4. 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";
};
```

2.5. Set Other UART as Print Console

Follow the procedure below to complete the setting.

1. Find the target UART port in the `board.dts` file, and make sure the port has been enabled.

   ```
   &uart3 {
           pinctrl-names = "default";
           pinctrl-0 = <&uart3_pins>;
           status = "okay";
   };
   ```

2. Modify the Kernel command line parameter which passed by the front boot step to use the target UART port as the printing console.

   ```
   earlyprintk console=ttyS3,115200 debug rootwait earlycon=sbi
   Note:
   ttyS0 <=====> uart0
   ttySl <=====> uartl
   ttyS2 <=====> uart2
   ttyS3 <=====> uart3
   ```

3. Interface Description

UART driver automatically registers and generates the following device for serial communication: `/dev/ttySx`.

When working on UART relevant applications, Linux application developers are recommended to follow the standard Linux coding practices.

3.1. Enable or Disable the Serial Port

When enabling or disabling the serial port, make sure you have followed the rules below:

- Include all the following header files:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
```

- Make sure you use standard functions to open and close the files you need.

```
int open(const char *pathname, int flags);
int close (int fd)
```

3.2. Configure Serial Port Attributes

Make sure you have included the following file before setting attributes.

```
#include <termios.h>
```

Commonly, a serial port has the following attributes.

- Baud rate
- Data bit
- Stop bit
- Verification bit
- Traffic control

The following interfaces are used to configure the attributes of a serial port.

3.2.1. tcgetattr

`tcgetattr` is a Linux standard interface to get parameters from the terminal.

See *Library Functions* on the Linux man-pages project for more information.

3.2.2. tcsetattr

`tcsetattr` is a Linux standard interface to set parameters to the terminal.

See *Library Functions* on the Linux man-pages project for more information.

3.2.3. cfgetispeed

`cfgetispeed` is a Linux standard interface to get input baud rate.

See *Library Functions* on the Linux man-pages project for more information.

3.2.4. cfgetospeed

`cfgetispeed` is a Linux standard interface to get output baud rate.

See *Library Functions* on the Linux man-pages project for more information.

3.2.5. cfsetispeed

`cfsetispeed` is a Linux standard interface to set input baud rate.

See *Library Functions* on the Linux man-pages project for more information.

3.2.6. cfsetospeed

`cfsetospeed` is a Linux standard interface to set output baud rate.

See *Library Functions* on the Linux man-pages project for more information.

3.2.7. cfsetspeed

`cfsetospeed` is a Linux standard interface to set input and output speed.

See *Library Functions* on the Linux man-pages project for more information.

3.2.8. tcflush

`tcflush` is a Linux standard interface to discard data written to the object .

See *Library Functions* on the Linux man-pages project for more information.

4. Example Use Case

The following demo program including the complete process of opening a UART device, listening on it, and printing when readable data is detected.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>  /*File control definition*/
#include <termios.h>  /*PPSIX Terminal operating system definition*/
#include <stdio.h>  /*Standard input and output definition*/
#include <unistd.h> /*UNIX standard function definition*/

#define BAUDRATE        115200
#define UART_DEVICE     "/dev/ttyS3"

#define FALSE  -1
#define TRUE   0

/**
*@brief  Set communication speed for UART
*@param  fd     Type int  Open UART file
*@param  speed  Type int  UART speed
*@return  void
*/
int speed_arr[] = {B115200, B38400, B19200, B9600, B4800, B2400, B1200, B300,
              B115200, B38400, B19200, B9600, B4800, B2400, B1200, B300, };
int name_arr[] = {115200, 38400, 19200, 9600, 4800, 2400, 1200,  300,
        115200, 38400, 19200, 9600, 4800, 2400, 1200,  300, };
void set_speed(int fd, int speed){
  int   i;
  int   status;
  struct termios   Opt;
  tcgetattr(fd, &Opt);
  for ( i= 0;  i < sizeof(speed_arr) / sizeof(int);  i++) {
    if  (speed == name_arr[i]) {
      tcflush(fd, TCIOFLUSH);
      cfsetispeed(&Opt, speed_arr[i]);
      cfsetospeed(&Opt, speed_arr[i]);
      status = tcsetattr(fd, TCSANOW, &Opt);
      if  (status != 0) {
        perror("tcsetattr fd1");
        return;
      }
      tcflush(fd,TCIOFLUSH);
    }
  }
}

/**
*@brief   Set data bit, stop bit and parity check bit
*@param  fd     Type  int  OPEN UART file
*@param  databits Type  int data bit   Value 7 or 8
*@param  stopbits Type  int stop bit   Value 1 or 2
*@param  parity  Type  int  parity check Type Value N,E,O,,S
*/
int set_Parity(int fd,int databits,int stopbits,int parity)
{
 struct termios options;
 if  ( tcgetattr( fd,&options)  !=  0) {
  perror("SetupSerial 1");
  return(FALSE);
 }
 options.c_cflag &= ~CSIZE;
 switch (databits) /*Set dataa bit count*/
 {
 case 7:
  options.c_cflag |= CS7;
  break;
 case 8:
  options.c_cflag |= CS8;
```

```
  break;
 default:
  fprintf(stderr,"Unsupported data size\n"); return (FALSE);
 }
 switch (parity)
 {
  case 'n':
  case 'N':
   options.c_cflag &= ~PARENB;   /* Clear parity enable */
   options.c_iflag &= ~INPCK;     /* Enable parity checking */
   break;
  case 'o':
  case 'O':
   options.c_cflag |= (PARODD | PARENB); /* Set as odd parity check*/
   options.c_iflag |= INPCK;             /* Disable parity check */
   break;
  case 'e':
  case 'E':
   options.c_cflag |= PARENB;     /* Enable parity */
   options.c_cflag &= ~PARODD;   /* Set as even parity check*/
   options.c_iflag |= INPCK;       /* Disable parity check */
   break;
  case 'S':
  case 's':  /*as no parity*/
   options.c_cflag &= ~PARENB;
   options.c_cflag &= ~CSTOPB;break;
  default:
   fprintf(stderr,"Unsupported parity\n");
   return (FALSE);
  }
 /* Set stop bit*/
 switch (stopbits)
 {
  case 1:
   options.c_cflag &= ~CSTOPB;
   break;
  case 2:
   options.c_cflag |= CSTOPB;
     break;
  default:
    fprintf(stderr,"Unsupported stop bits\n");
    return (FALSE);
 }
 /* Set input parity option */
 if (parity != 'n')
  options.c_iflag |= INPCK;
 tcflush(fd,TCIFLUSH);
 options.c_cc[VTIME] = 150; /* Set to more than 15 seconds*/
 options.c_cc[VMIN] = 0; /* Update the options and do it NOW */
 if (tcsetattr(fd,TCSANOW,&options) != 0)
 {
  perror("SetupSerial 3");
  return (FALSE);
 }
 options.c_lflag  &= ~(ICANON | ECHO | ECHOE | ISIG);  /*Input*/
 options.c_oflag  &= ~OPOST;   /*Output*/
 return (TRUE);
}

int main(int argc, char *argv[])
{

    int    fd, c=0, res;
    char *dev;

    char  buf[256];

    printf("Start...\n");
    if (argc == 2)
     dev = argv[1];
    else
     dev = UART_DEVICE;
```

```
    fd = open(dev, O_RDWR);

    if (fd < 0) {
        perror(UART_DEVICE);
        exit(1);
    }

    printf("Open...\n");
 printf("bandrate %d...\n",BAUDRATE);
    set_speed(fd,BAUDRATE);
 if (set_Parity(fd,8,1,'N') == FALSE)  {
  printf("Set Parity Error\n");
  exit (0);
 }

    printf("Reading...\n");
    while(1) {
        res = read(fd, buf, 255);

        if(res==0)
            continue;
        buf[res]=0;

        printf("%s", buf);

        if (buf[0] == 0x0d)
         printf("\n");
 write(fd,buf,res);

        if (buf[0] == '@') break;
    }

    printf("Close...\n");
    close(fd);

    return 0;
}
```