Raspberry Pi 4 UART Kernel Module Implementation Guide

Overview

This guide shows how to implement a UART kernel module for Raspberry Pi 4 from scratch. The module provides direct access to the hardware UART controller (PL011) with interrupt-driven I/O.

Prerequisites

1. **Raspberry Pi 4** with Raspbian/Debian
2. **Kernel headers** installed:
   ```bash
   sudo apt update
   sudo apt install raspberrypi-kernel-headers build-essential
   ```
3. **Root access** for module loading/unloading

Hardware Details

Raspberry Pi 4 UART Configuration

- **UART0 (PL011)**: Primary UART, full featured
- **Base Address**: 0xFE201000 (BCM2711)
- **GPIO Pins**:
  - GPIO 14 (TXD) - Pin 8
  - GPIO 15 (RXD) - Pin 10
- **IRQ**: 29

Memory Layout

The BCM2711 (RPi4) uses different base addresses than earlier Pi models:

- **Physical**: 0xFE201000
- **Size**: 4KB (0x1000)

Building the Module

1. **Save the source files**:
   - `rpi4_uart.c` - Main module source
   - `Makefile` - Build configuration

2. **Build the module**:

```bash
make clean
make
```

3. **Check build output**:

```bash
ls -la *.ko
```

Module Installation

Load the Module

```bash
sudo insmod rpi4_uart.ko
```

Verify Loading

```bash
lsmod | grep rpi4_uart
dmesg | tail -10
```

Check Device Creation

```bash
ls -la /dev/rpi4_uart
```

If the device node doesn't exist automatically:

```bash
sudo mknod /dev/rpi4_uart c $(cat /proc/devices | grep rpi4_uart | awk '{print $1}') 0
sudo chmod 666 /dev/rpi4_uart
```

Testing the Module

Compile Test Program

```bash
gcc -o uart_test uart_test.c
```

Test Modes

1. **Send Text**:
   ```bash
   ./uart_test -t "Hello UART!"
   ```
2. **Send File**:
   ```bash
   echo "Test message" > test.txt
   ./uart_test -f test.txt
   ```
3. **Interactive Mode**:
   ```bash
   ./uart_test -i
   ```
4. **Read-Only Monitor**:
   ```bash
   ./uart_test -r
   ```

Hardware Connections

Internal UART (UART0)

- Accessible via GPIO pins 14 and 15
- No external connections needed for testing
- Can connect to external devices via pins 8 and 10

External Device Connections

```
RPi4 Pin | GPIO | Function | External Device
---------|------|----------|----------------
```

```
Pin 8    | 14   | TXD      | RXD of target device
Pin 10   | 15   | RXD      | TXD of target device
Pin 6    | GND  | Ground   | Ground of target device
```

Loopback Testing

For testing without external hardware, connect:

- Pin 8 (TXD) to Pin 10 (RXD) with a jumper wire

Module Architecture

Key Components

1. **Hardware Layer**
   - Direct register access to PL011 UART controller
   - GPIO configuration for UART pins
   - Interrupt handling for RX/TX
2. **Buffer Management**
   - Circular buffers for RX and TX
   - Thread-safe operations with spinlocks
   - Wait queues for blocking I/O
3. **Character Device Interface**
   - Standard file operations (open, close, read, write)
   - Device node creation via udev
   - User-space accessibility

Register Map (PL011 UART)

```
Offset | Register | Description
-------|----------|------------------------
0x00   | DR       | Data Register
0x04   | RSR      | Receive Status Register
0x18   | FR       | Flag Register
0x24   | IBRD     | Integer Baud Rate Divisor
0x28   | FBRD     | Fractional Baud Rate Divisor
0x2C   | LCRH     | Line Control Register
0x30   | CR       | Control Register
```

```
0x38  | IMSC    | Interrupt Mask Set/Clear
0x44  | ICR     | Interrupt Clear Register
```

Configuration Options

Baud Rate Configuration

The module defaults to 115200 baud. To change:

1. **Modify source code**:

   ```c
   // For 9600 baud (48MHz clock):
   uart_write_reg(UART_IBRD, 312);  // 48000000/(16*9600) = 312.5
   uart_write_reg(UART_FBRD, 32);   // 0.5 * 64 = 32
   ```

2. **Common baud rates**:

   ```
   Baud Rate | IBRD | FBRD
   ----------|------|-----
   9600      | 312  | 32
   19200     | 156  | 16
   38400     | 78   | 8
   57600     | 52   | 5
   115200    | 26   | 3
   ```

Buffer Size

Default buffer size is 1024 bytes. Modify `BUFFER_SIZE` in source:

```c
#define BUFFER_SIZE 2048  // Increase buffer size
```

Debugging and Troubleshooting

Common Issues

1. **Module fails to load**:

   ```bash
   ```

```bash
dmesg | grep -i error
# Check kernel version compatibility
uname -r
```

2. **Device node not created**:

```bash
# Manual creation
sudo mknod /dev/rpi4_uart c $(cat /proc/devices | grep rpi4_uart | awk '{print $1}') 0
sudo chmod 666 /dev/rpi4_uart
```

3. **Permission denied**:

```bash
sudo chmod 666 /dev/rpi4_uart
# Or add user to dialout group
sudo usermod -a -G dialout $USER
```

4. **GPIO conflicts**:

```bash
# Check GPIO usage
cat /sys/kernel/debug/gpio
# Disable other UART services
sudo systemctl disable serial-getty@ttyS0.service
```

Debug Commands

```bash
```

```bash
# Check module status
lsmod | grep rpi4_uart

# View kernel messages
dmesg | tail -20

# Check device information
cat /proc/devices | grep rpi4_uart

# Monitor interrupts
watch -n 1 'cat /proc/interrupts | grep uart'

# Check GPIO status
cat /sys/kernel/debug/gpio | grep -A2 -B2 "gpio-1[45]"
```

Enable Additional Debugging

Add to module source for verbose debugging:

```c
#define DEBUG 1
// Add debug prints
pr_debug("UART: %s\n", __func__);
```

Compile with debug info:

```bash
make EXTRA_CFLAGS=-DDEBUG
```

Performance Considerations

Interrupt Load

- RX interrupts trigger on FIFO threshold
- TX interrupts only when buffer has data
- Use `UART_IFLS` register to adjust FIFO levels

Buffer Tuning

```c
```

```c
// Adjust FIFO interrupt levels
#define UART_IFLS_RX_1_8  (0 << 3)  // RX FIFO 1/8 full
#define UART_IFLS_RX_1_4  (1 << 3)  // RX FIFO 1/4 full
#define UART_IFLS_RX_1_2  (2 << 3)  // RX FIFO 1/2 full
#define UART_IFLS_TX_1_8  (0 << 0)  // TX FIFO 1/8 full
#define UART_IFLS_TX_1_4  (1 << 0)  // TX FIFO 1/4 full
```

Memory Usage

- Static buffers: 2KB (1KB RX + 1KB TX)
- Can be made dynamic with kmalloc/kfree
- Consider using kfifo for better buffer management

Advanced Features

Adding Flow Control

```c
// Hardware flow control (RTS/CTS)
#define GPIO_UART_RTS 16
#define GPIO_UART_CTS 17

// In uart_hw_init():
gpio_request(GPIO_UART_RTS, "uart_rts");
gpio_request(GPIO_UART_CTS, "uart_cts");
```

Multiple UART Support

The RPi4 has multiple UARTs:

- UART0 (PL011): Full-featured, this implementation
- UART1 (Mini UART): Simplified, different registers
- UART2-5: Additional PL011 instances

DMA Support

For high-speed transfers, implement DMA:

```c
```

```c
#include <linux/dmaengine.h>
// Request DMA channels
// Configure scatter-gather lists
// Implement DMA callbacks
```

Unloading the Module

```bash
# Remove module
sudo rmmod rpi4_uart

# Verify removal
lsmod | grep rpi4_uart

# Clean up device node if needed
sudo rm -f /dev/rpi4_uart
```

Production Considerations

1. **Error Handling**: Add comprehensive error checking
2. **Power Management**: Implement suspend/resume callbacks
3. **Device Tree**: Use device tree for hardware configuration
4. **Sysfs Interface**: Add sysfs attributes for runtime configuration
5. **Multiple Instances**: Support multiple UART instances
6. **Flow Control**: Implement RTS/CTS hardware flow control

Example Applications

Serial Communication

```c
```

```c
// Simple echo server
int fd = open("/dev/rpi4_uart", O_RDWR);
char buffer[256];
while (1) {
    int bytes = read(fd, buffer, sizeof(buffer));
    if (bytes > 0) {
        write(fd, buffer, bytes); // Echo back
    }
}
```

GPS Module Interface

```bash
# Connect GPS module and read NMEA sentences
./uart_test -r | grep '$GPGGA'
```

Sensor Data Collection

```bash
# Log sensor data to file
./uart_test -r > sensor_data.log
```

This implementation provides a solid foundation for UART communication on Raspberry Pi 4, with room for extension and customization based on specific requirements.