SPI Driver Documentation

Salman, Tayyab, Zawaher

Contents

1 Introduction

This document describes the SPI driver developed for a custom core and its integration into the Linux kernel. The driver is designed to handle SPI communication, enabling efficient interaction between hardware and software through the Linux SPI framework.

The driver includes features such as:

- Read and write operations via a '/proc' file interface.
- Interrupt-based data transmission and reception.
- Memory-mapped I/O for interacting with the SPI core registers.
- Compatibility with the Linux device tree for dynamic hardware configuration.

2 Driver Architecture

The SPI driver is implemented to abstract the hardware-level operations of the SPI core into a user-friendly Linux interface. Below is the block diagram illustrating the architecture:

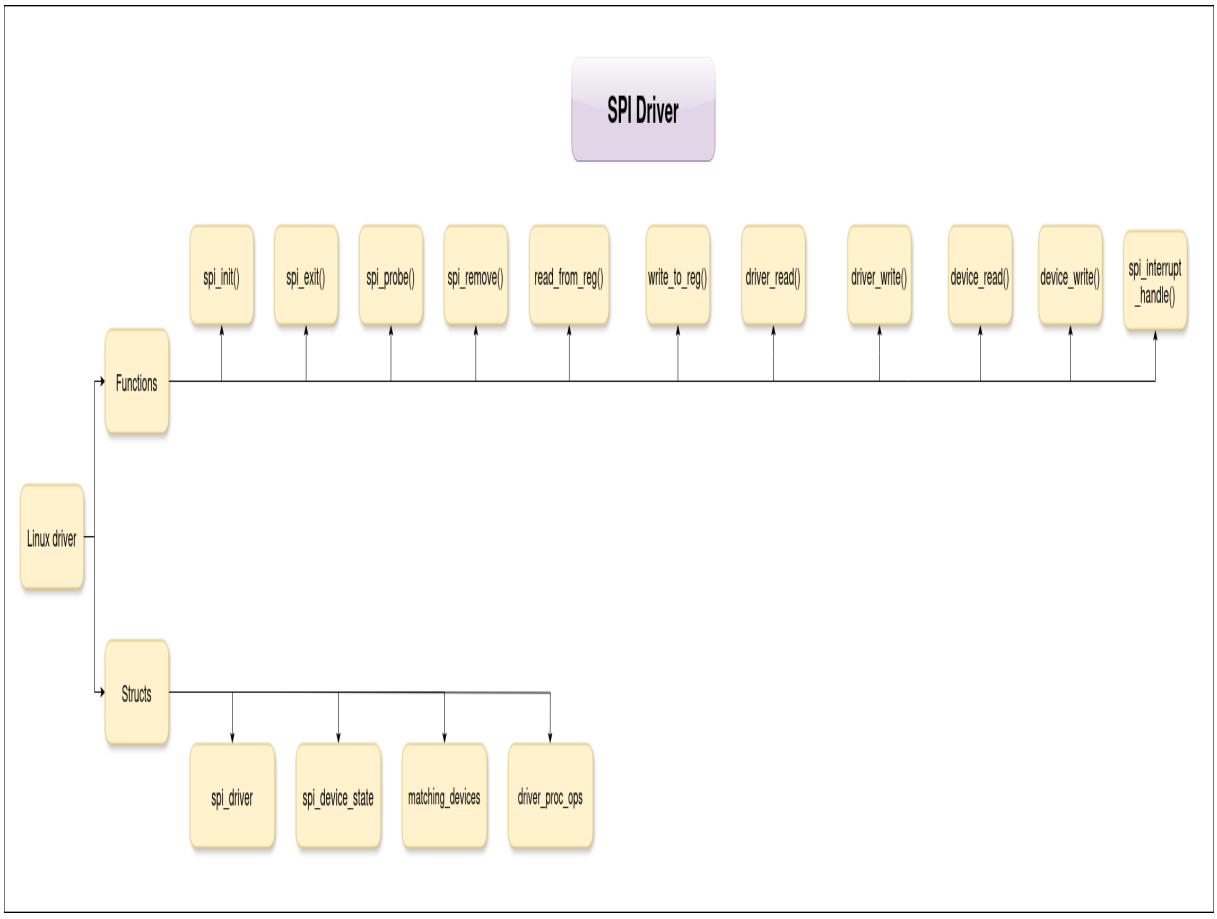

Figure 1: Block Diagram of the SPI Driver Architecture

3 Integration into Linux Kernel

To integrate the SPI driver into Linux, the following steps are performed:

3.1 Device Tree Configuration

The SPI driver uses the Linux device tree for configuration. The relevant entry in the device tree specifies the compatible hardware and memory mappings for the SPI core:

```
spi0: spi@10030000 {
    compatible = "sifive,spi0";
    reg = <0x10030000 0x1000>;
    interrupts = <3>;
};
```

3.2 Kernel Driver Registration

The driver is implemented as a platform driver and registered with the kernel using `platform_driver_register`. This enables the kernel to associate the driver with the hardware specified in the device tree.

3.3 Build and Load

The driver module is compiled as a kernel module using the Linux kernel build system.In order to build the driver use the `make` command.The `"spidriver.ko"` file can be obtained from the build folder. It is loaded using the `insmod` command and interacts with the core through memory-mapped I/O.

3.4 Testing

The SPI driver is tested for:

- Correct handling of data transmission and reception.
- Compatibility with standard Linux SPI tools.
- Performance under different system loads.

4 Structures

This section describes the structures used in the SPI driver.

4.1 spi_device_state

Purpose: Stores the state and resources of the SPI device, including buffer data and configuration settings.

Fields:

- `void __iomem *base_address`: Base address of the SPI device's memory-mapped registers.
- `char data_tx_available`: Flag indicating if data is available for transmission.
- `char data_rx_available`: Flag indicating if data is available for reception.
- `uint rx_index`: Index for managing the receive buffer.
- `uint tx_index`: Index for managing the transmit buffer.
- `uint user_rx_index`: Index for user access to the receive buffer.
- `uint user_tx_index`: Index for user access to the transmit buffer.
- `char rx_data_buffer[MSG_BUFFER_SIZE]`: Buffer for received data.
- `char tx_data_buffer[MSG_BUFFER_SIZE]`: Buffer for transmitted data.

—

4.2 proc_ops

Purpose: Defines operations for the ‘/proc‘ file entry for the SPI driver.

Fields:

- `proc_read`: Function pointer for handling read operations from the ‘/proc‘ file.
- `proc_write`: Function pointer for handling write operations to the ‘/proc‘ file.

Working: This structure connects the SPI driver's read and write handlers (`driver_read` and `driver_write`) with the ‘/proc‘ filesystem interface.

—

4.3 matching_devices

Purpose: Defines a list of hardware devices compatible with this driver.

Fields:

- `.compatible`: A string specifying compatible devices, e.g., `"sifive,spi0"`.

Working: This structure enables the Linux kernel to match the driver with the appropriate hardware based on the "compatible" property in the device tree.

—

4.4 spi_driver

Purpose: Represents the SPI driver and its associated functions for initialization and cleanup.

Fields:

- `probe`: Points to the `spi_probe` function, called when a device is initialized.
- `remove`: Points to the `spi_remove` function, called when a device is removed.
- `driver`: Contains metadata about the driver:
  - `.name`: Name of the driver, e.g., `"Salman-Tayyab-Zawaher's_driver"`.
  - `.of_match_table`: Points to the `matching_devices` structure.

Working: Associates the driver's operational functions with compatible devices detected in the system.

5 Functions

This section describes the functions implemented in the SPI driver.

5.1 spi_init

Purpose: Initializes the SPI driver by registering it with the platform driver framework.

Working:

- Prints a message to indicate initialization.
- Registers the SPI driver using `platform_driver_register`.

—

5.2 spi_exit

Purpose: Cleans up the SPI driver during removal.

Working:

- Unregisters the SPI driver using `platform_driver_unregister`.
- Prints a message to indicate driver removal.

—

5.3 spi_probe

Purpose: Allocates resources, maps memory, and sets up interrupts for the SPI device.

Working:

- Allocates memory for the `spi_device_state` structure.
- Maps the device's memory region.

- Registers an interrupt handler using `request_irq`.
- Initializes device-specific settings.

—

5.4 spi_remove

Purpose: Cleans up resources allocated during the probe phase.
Working:

- Releases allocated memory and resources.
- Unregisters the device from the driver.

—

5.5 write_to_reg

Purpose: Writes data to a specified SPI register.
Working: Uses `iowrite32` to write data to the given memory-mapped address.
—

5.6 read_from_reg

Purpose: Reads data from a specified SPI register.
Working: Uses `ioread32` to read data from the given memory-mapped address.
—

5.7 driver_read

Purpose: Handles read operations from the driver's proc file.
Working:

- Copies data from the receive buffer to the user space.
- Updates the user's buffer index.

—

5.8 driver_write

Purpose: Handles write operations to the driver's proc file.
Working:

- Copies data from the user space to the transmit buffer.
- Sets the `data_tx_available` flag.

—

5.9 device_write

Purpose: Transfers data from the transmit buffer to the SPI TX FIFO.
Working:

- Writes characters to the TX FIFO until it is full or the buffer is empty.
- Clears the `data_tx_available` flag when all data is written.

—

5.10 device_read

Purpose: Transfers data from the SPI RX FIFO to the receive buffer.
Working:

- Reads characters from the RX FIFO until it is empty.
- Stores the data in the receive buffer.

—

5.11 spi_interrupt_handler

Purpose: Handles SPI device interrupts.
Working:

- Reads interrupt status and disables interrupts.
- Calls `device_write` if the TX FIFO is ready to transmit and data is available.
- Calls `device_read` if data is available in the RX FIFO.

6 NON-Interrupt Driver Struct and Functions

driver_open

- **Purpose**: Configures the SPI controller to select the appropriate chip select line (CS) based on the device file accessed.
- **Working**:
  1. Extracts the minor number of the device file (`/dev/spi0` or `/dev/spi1`) using `iminor(inode)`.
  2. For `/dev/spi0` (minor 0), writes `1` to the `SPI_CS_ID_R` register, selecting CS0.
  3. For `/dev/spi1` (minor 1), writes `2` to the `SPI_CS_ID_R` register, selecting CS1.

driver_close

- **Purpose**: Resets the SPI controller by deselecting all chip select lines when the device file is closed.
- **Working**:
    1. Writes `1` to the `SPI_CS_ID_R` register, ensuring no active CS line remains selected.

driver_dev_ops Structure

Purpose

Links file operations (`open`, `read`, `write`, `release`) for the SPI driver with the respective kernel-space functions.

Working

- `open`: Calls `driver_open` to select the appropriate CS line.
- `read`: Reads data from the SPI RX FIFO.
- `write`: Writes data to the SPI TX FIFO.
- `release`: Calls `driver_close` to reset the CS lines.