Raspberry Pi 5 Driver Development - Quick Setup Guide

Target: Raspberry Pi 5 with Linux Kernel 6.6
Time: ~2-3 hours total
Result: Ready-to-use driver development environment

What You Need

- Raspberry Pi 5
- 16GB+ microSD card
- Ethernet cable (for initial setup)
- Ubuntu 22.04/24.04 PC
- USB-C power supply (5V/5A)

Part 1: PC Setup (45-90 minutes)

Install Tools

```
# Update system
sudo apt update && sudo apt upgrade -y

# Install build tools
sudo apt install -y git bc bison flex libssl-dev make \
  libc6-dev libncurses5-dev build-essential

# Install cross-compiler for 64-bit ARM
sudo apt install -y crossbuild-essential-arm64

# Verify
aarch64-linux-gnu-gcc --version
```

Get Kernel Source

```
# Create workspace
mkdir -p ~/rpi-driver-dev
cd ~/rpi-driver-dev

# Clone kernel (takes 5-10 min)
git clone --depth=1 -b rpi-6.6.y https://github.com/raspberrypi/linux
cd linux
```

Configure Kernel

```
# Set kernel name
export KERNEL=kernel_2712

# Load default config
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- bcm2711_defconfig

# Customize (optional but recommended)
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- menuconfig
```

In menuconfig (use arrow keys, Space to select `<M>`): - Device Drivers → SPI support → `<M>` User mode SPI device driver - Device Drivers → `<M>` Userspace I/O drivers - Device Drivers → `<M>` Industrial I/O support - Device Drivers → LED Support → `<M>` LED Support for GPIO

Save: ESC ESC → Yes

Build Kernel (30-90 min depending on CPU)

```
make -j$(nproc) ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- Image modules dtbs
```

Get coffee. This takes a while.

Verify Build

```
# Check kernel image exists (~30MB)
ls -lh arch/arm64/boot/Image

# Check device trees exist
ls arch/arm64/boot/dts/broadcom/bcm2712*.dtb

# Save kernel version
make kernelrelease
# Remember this version! (e.g., 6.6.78-v8-16k+)
```

Part 2: SD Card Setup (20 minutes)

Flash Raspberry Pi OS

```
# Install imager
sudo snap install rpi-imager

# Launch
rpi-imager
```

In the imager: 1. Choose Device: **Raspberry Pi 5** 2. Choose OS: **Raspberry Pi OS (64-bit)** 3. Choose Storage: **Your SD card** 4. Click **Next** → **EDIT SETTINGS**

Configure: - Hostname: `raspberrypi5` - Username: `pi` / Password: `[your choice]` - WiFi: Your network details (optional) - Enable SSH (password authentication)

Save → **YES** → **Flash** (takes 5-15 min)

Enable SSH Manually (if imager customization fails)

```
# After flashing, re-insert SD card
# Create SSH file
sudo touch /media/$USER/bootfs/ssh

# Create user (if login fails)
echo 'raspberry' | openssl passwd -6 -stdin
# Copy the hash output, then:
echo 'pi:PASTE_HASH_HERE' | sudo tee /media/$USER/bootfs/userconf.txt

# Enable password auth
sudo sed -i 's/#PasswordAuthentication yes/PasswordAuthentication yes/' \
  /media/$USER/rootfs/etc/ssh/sshd_config

# Fix permissions
sudo chown -R 1000:1000 /media/$USER/rootfs/home/pi
```

Copy Custom Kernel

```
cd ~/rpi-driver-dev/linux

# Install modules (2-5 min)
sudo env PATH=$PATH make ARCH=arm64 \
  CROSS_COMPILE=aarch64-linux-gnu- \
  INSTALL_MOD_PATH=/media/$USER/rootfs \
  modules_install

# Backup stock kernel
sudo cp /media/$USER/bootfs/kernel_2712.img \
  /media/$USER/bootfs/kernel_2712.img.backup

# Copy your kernel
sudo cp arch/arm64/boot/Image /media/$USER/bootfs/kernel_2712.img

# Copy device trees
sudo cp arch/arm64/boot/dts/broadcom/bcm2712*.dtb /media/$USER/bootfs/
```

```
# Copy overlays
sudo cp arch/arm64/boot/dts/overlays/*.dtb* /media/$USER/bootfs/overlays/
sudo cp arch/arm64/boot/dts/overlays/README /media/$USER/bootfs/overlays/

# Safely unmount
sync && sync && sync
sudo umount /media/$USER/bootfs
sudo umount /media/$USER/rootfs
```

Part 3: Boot & Connect (10 minutes)

First Boot

1. Insert SD card into Pi 5
2. Connect Ethernet cable (Pi → Router)
3. Power on
4. Wait 2 minutes

Find Pi's IP

Check your router's connected devices or:

```
# Scan network
sudo apt install nmap
nmap -sn 192.168.1.0/24 | grep -B 2 "Raspberry"
```

SSH In

```
ssh pi@192.168.1.XXX  # Use your Pi's IP
# Or try:
ssh pi@raspberrypi5.local
```

First time: - Type `yes` when asked - Enter your password

Verify Custom Kernel

```
uname -r
# Should show: 6.6.78-v8-16k+ (your version)

uname -m
# Should show: aarch64
```

If version matches - SUCCESS!

Part 4: Test Module (15 minutes)

Create Test Module (on PC)

```
mkdir -p ~/rpi-driver-dev/test_module
cd ~/rpi-driver-dev/test_module

# Create hello.c
cat > hello.c << 'EOF'
#include <linux/module.h>
#include <linux/kernel.h>

static int __init hello_init(void) {
    pr_info("Hello from Pi 5!\n");
    return 0;
}

static void __exit hello_exit(void) {
    pr_info("Goodbye from Pi 5!\n");
}

module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your Name");
MODULE_DESCRIPTION("Test module");
EOF

# Create Makefile
cat > Makefile << 'EOF'
KERNEL_SRC := $(HOME)/rpi-driver-dev/linux
obj-m += hello.o

all:
    make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- \
      -C $(KERNEL_SRC) M=$(PWD) modules

clean:
    make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- \
      -C $(KERNEL_SRC) M=$(PWD) clean
EOF

# Build
make
```

Deploy to Pi

```
# Copy module
scp hello.ko pi@192.168.1.XXX:~/

# SSH to Pi
ssh pi@192.168.1.XXX

# Load module
sudo insmod hello.ko

# Check kernel log
dmesg | tail -5
# Should show: "Hello from Pi 5!"

# Unload module
sudo rmmod hello

# Check again
dmesg | tail -5
# Should show: "Goodbye from Pi 5!"
```

If you see both messages - YOUR SETUP IS COMPLETE!

Part 5: WiFi Setup (Optional, 10 minutes)

On the Pi (via SSH):

```
# Unblock WiFi
sudo rfkill unblock wifi

# Enable WiFi radio
sudo nmcli radio wifi on

# Scan for networks
sudo nmcli device wifi rescan
sudo nmcli device wifi list

# Connect (replace with your details)
sudo nmcli device wifi connect "YourWiFiName" password "YourPassword"

# Check connection
ip a | grep wlan0 -A 3
# Should show IP address

# Test
```

```
ping -c 4 google.com
```

Now unplug Ethernet and SSH via WiFi!

```
ssh pi@192.168.1.XXX  # Use WiFi IP
```

Your Development Workflow

Build Module (on PC)

```
cd ~/rpi-driver-dev/your_project
# Edit your .c file
make
```

Deploy & Test (on Pi)

```
# Copy from PC
scp module.ko pi@raspberrypi5.local:~/

# SSH to Pi
ssh pi@raspberrypi5.local

# Test
sudo insmod module.ko
dmesg | tail
sudo rmmod module
```

Useful Commands

On PC

```
# Clean build
make clean

# Check module info
modinfo module.ko

# Rebuild kernel (if config changed)
cd ~/rpi-driver-dev/linux
make -j$(nproc) ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- modules
```

On Pi

```
# View kernel logs
dmesg | tail -20
```

```
# List loaded modules
lsmod

# Module info
modinfo module_name

# Kernel version
uname -r

# Live log monitoring
sudo dmesg -w
```

Troubleshooting

SSH Won't Connect

Create SSH file on SD card:

```
sudo touch /media/$USER/bootfs/ssh
```

Wrong Kernel Running

Check version:

```
uname -r
```

If not 6.6.x, re-copy kernel to SD card (see Part 2)

Module Won't Load

Check version match:

```
# On Pi
uname -r

# On PC
modinfo module.ko | grep vermagic
```

Must match exactly. Rebuild if different.

WiFi Not Working

```
# Unblock
sudo rfkill unblock all

# Enable
sudo nmcli radio wifi on
```

```
# Rescan
sudo nmcli device wifi rescan
sudo nmcli device wifi list
```

Key Differences from Pi 3/4

Item	Pi 3/4	Pi 5
Compiler	arm-linux-gnueabihf-	**aarch64-linux-gnu-**
Config	bcm2709_defconfig	**bcm2711_defconfig**
Kernel	kernel8.img	**kernel_2712.img**
Device Tree	bcm2710*.dtb	**bcm2712*.dtb**

What You Built

Custom Linux kernel 6.6.78 for Pi 5
Cross-compilation environment
WiFi connectivity
Working test module
Complete development workflow

You're ready for Linux driver development!

Next Steps

- Study Linux driver books (adapt for kernel 6.6)
- Experiment with GPIO drivers
- Try I2C/SPI device drivers
- Build custom hardware interfaces

Happy coding!

Quick Reference Card

```
# Build on PC
cd ~/rpi-driver-dev/project
make

# Deploy
```

```
scp module.ko pi@raspberrypi5.local:~/

# Test on Pi
sudo insmod module.ko
dmesg | tail
sudo rmmod module

# Kernel version
uname -r  # Should be 6.6.x-v8-16k+
```

Total Setup Time: ~2-3 hours
Kernel: Linux 6.6.78-v8-16k+
Architecture: aarch64 (64-bit ARM)
Status: Production Ready