Step 1: Know about the Hardware

Knowing about the hardware involves ..

- What is the Bus interface this hardware uses to communicate with the host (USB, PCIe, I2C etc..), where the Device Driver Software resides.
- How to detect the hardware. In almost all the cases the hardware bus mandates that the Hardware Manufacturer assign an ID. How the ID is assigned might vary based on the hardware Bus.
- Register Definition knowledge. Typically the device datasheet can provide this information.
- If the main device is a SoC (System on Chip) , does your driver need to interact with Firmware running in the SoC. Know, what is the Command Protocol defined by the Firmware
- Knowing how the Data Transfer take place, Is it polling or interrupt driven?

Step 2: Say Hello to your hardware (In other words, talk to your hardware)

Once the basic information is known about the hardware, then it's time to talk to the hardware. At this point you should at least have a prototype hardware or a finalized version of the hardware available.

Start writing your Kernel Mode Driver. The Driver Development Documentation for the selected OS (Windows, Linux , RTOS etc..) should have ton of information.

Usually the the first functions that you develop for your Device Driver are Load and Unload functions (Note: the function names may be different for each OS), Which are invoked when the Operation System Starts and Shuts down respectively.

One of the primary duty of the Load/Unload functions is to detect the hardware plugged into the system or not. If the Hardware is plugged in then Load function is successful if not call unload. Typical ways to detect the hardware are by Device ID specified by the specific Bus (e.g USB, PCIe etc..)

Step 3: Initialize your hardware

Once you are able to detect your hardware, the next is to initialize your hardware. The kind of initialization that is needed may vary for each hardware. It may range from just writing to couple of device registers to all the way, downloading a microcode onto the device and communicating using proprietary command protocol.

As an example , if you are developing a Device Driver for Audio Playback Device, then you might need to download the firmware code for the Audio DSP (Digital Signal Processor) , Send commands to initialize the type of Audio Decoding that needs to be done (MP3, Dolby AC3 etc..). In short prepare the hardware so that it's ready to operate.

Step 4 : Control your hardware

The next step is controlling your hardware. Until Step 3, the operation might be one time operation when the OS loads, Step 4 might be needed to be performed multiple times after the Operating System is up and running.

Taking our example of Audio Playback Device, Let's say you want to play a MP3 audio file, the high level application software might read the file types and characteristics, like the audio stream's Sampling Rate, Bit rate etc. and send the corresponding parameters to the Kernel Device Driver.The Device Driver in turn translates those parameters as commands to the Audio DSP. This operation is repeated for each file that is played back. In some cases the Step 3 and 4 might be merged into single step and that's fine.

Step 5: Data Communication to your hardware

More often than not your device might deal with some form of Data (Audio, Video, Network Packet etc..). Once the device is initialized and ready to operate then a steady stream of data needs to be supplied to the hardware.

Your Device Driver acts as a pipe between the higher level application and the lower level Hardware/Firmware for transferring the Data. Again, find out what is the protocol that was designed for data communication. If you are developing a driver from the scratch , you should have hashed out a protocol with the Hardware/Firmware folks on how to do data transfer.

Typically the data transfer might be interrupt driven or polling. Either way , your OS should provide facilities (Interrupt Service Routine , Messaging etc..). You will be using those facilities to do data transfer.

Start with single packet of data transfer and make sure the whole process from initializing to , hardware control to transferring of that packet works fine. The definition of single packet may vary depending upon your application, say for example , in Audio Device it might a one 64K audio data.

Step 6 : Start and Stop data communication

The next step is control the Data transfer . As in our Audio Playback example , sending the stop interrupt command to the firmware after the audio stream file is done. Typical issues faced here may be that the stopping of the stream does not occur at the right time resulting buffer overflow or underflow.

Step 7 : Fine Tune and Debug Your Driver based on Testing.

By this time you almost know in and out of the code and you will be the authority of your device driver. At this point your driver maybe ready for black box testing, since all the development is done.

The above steps may not match exactly to your situation but should be useful to give you an overall idea on how to go about in designing and developing a device driver or at the least, should be useful in answering the interview question " How do you Develop a Device Driver ?".

Happy Device Driving .. :)