How to Get Your Patch Accepted Into the Hwmon Subsystem

This text is a collection of suggestions for people writing patches or drivers for the hwmon subsystem. Following these suggestions will greatly increase the chances of your change being accepted.

1. General

- It should be unnecessary to mention, but please read and follow:
    - Documentation/process/submit-checklist.rst
    - Documentation/process/submitting-drivers.rst
    - Documentation/process/submitting-patches.rst
    - Documentation/process/coding-style.rst
- Please run your patch through 'checkpatch --strict'. There should be no errors, no warnings, and few if any check messages. If there are any messages, please be prepared to explain.
- Please use the standard multi-line comment style. Do not mix C and C++ style comments in a single driver (with the exception of the SPDX license identifier).
- If your patch generates checkpatch errors, warnings, or check messages, please refrain from explanations such as "I prefer that coding style". Keep in mind that each unnecessary message helps hiding a real problem, and a consistent coding style makes it easier for others to understand and review the code.
- Please test your patch thoroughly. We are not your test group. Sometimes a patch can not or not completely be tested because of missing hardware. In such cases, you should test-build the code on at least one architecture. If run-time testing was not achieved, it should be written explicitly below the patch header.
- If your patch (or the driver) is affected by configuration options such as CONFIG_SMP, make sure it compiles for all configuration variants.

2. Adding functionality to existing drivers

- Make sure the documentation in Documentation/hwmon/<driver_name>.rst is up to date.
- Make sure the information in Kconfig is up to date.
- If the added functionality requires some cleanup or structural changes, split your patch into a cleanup part and the actual addition. This makes it easier to review your changes, and to bisect any resulting problems.
- Never mix bug fixes, cleanup, and functional enhancements in a single patch.

3. New drivers

- Running your patch or driver file(s) through checkpatch does not mean its formatting is clean. If unsure about formatting in your new driver, run it through Lindent. Lindent is not perfect, and you may have to do some minor cleanup, but it is a good start.
- Consider adding yourself to MAINTAINERS.
- Document the driver in Documentation/hwmon/<driver_name>.rst.
- Add the driver to Kconfig and Makefile in alphabetical order.
- Make sure that all dependencies are listed in Kconfig.
- Please list include files in alphabetic order.
- Please align continuation lines with '(' on the previous line.
- Avoid forward declarations if you can. Rearrange the code if necessary.
- Avoid macros to generate groups of sensor attributes. It not only confuses checkpatch, but also makes it more difficult to review the code.
- Avoid calculations in macros and macro-generated functions. While such macros may save a line or so in the source, it obfuscates the code and makes code review more difficult. It may also result in code which is more complicated than necessary. Use inline functions or just regular functions instead.
- Limit the number of kernel log messages. In general, your driver should not generate an error message just because a runtime operation failed. Report errors to user space instead, using an appropriate error code. Keep in mind that kernel error log messages not only fill up the kernel log, but also are printed synchronously, most likely with interrupt disabled, often to a serial console. Excessive logging can seriously affect system performance.

- Use devres functions whenever possible to allocate resources. For rationale and supported functions, please see Documentation/driver-api/driver-model/devres.rst. If a function is not supported by devres, consider using devm_add_action().
- If the driver has a detect function, make sure it is silent. Debug messages and messages printed after a successful detection are acceptable, but it must not print messages such as "Chip XXX not found/supported".

  Keep in mind that the detect function will run for all drivers supporting an address if a chip is detected on that address. Unnecessary messages will just pollute the kernel log and not provide any value.
- Provide a detect function if and only if a chip can be detected reliably.
- Only the following I2C addresses shall be probed: 0x18-0x1f, 0x28-0x2f, 0x48-0x4f, 0x58, 0x5c, 0x73 and 0x77. Probing other addresses is strongly discouraged as it is known to cause trouble with other (non-hwmon) I2C chips. If your chip lives at an address which can't be probed then the device will have to be instantiated explicitly (which is always better anyway.)
- Avoid writing to chip registers in the detect function. If you have to write, only do it after you have already gathered enough data to be certain that the detection is going to be successful.

  Keep in mind that the chip might not be what your driver believes it is, and writing to it might cause a bad misconfiguration.
- Make sure there are no race conditions in the probe function. Specifically, completely initialize your chip and your driver first, then register with the hwmon subsystem.
- Use devm_hwmon_device_register_with_info() or, if your driver needs a remove function, hwmon_device_register_with_info() to register your driver with the hwmon subsystem. Try using devm_add_action() instead of a remove function if possible. Do not use hwmon_device_register().
- Your driver should be buildable as module. If not, please be prepared to explain why it has to be built into the kernel.
- Do not provide support for deprecated sysfs attributes.
- Do not create non-standard attributes unless really needed. If you have to use non-standard attributes, or you believe you do, discuss it on the mailing list first. Either case, provide a detailed explanation why you need the non-standard attribute(s). Standard attributes are specified in Documentation/hwmon/sysfs-interface.rst.
- When deciding which sysfs attributes to support, look at the chip's capabilities. While we do not expect your driver to support everything the chip may offer, it should at least support all limits and alarms.
- Last but not least, please check if a driver for your chip already exists before starting to write a new driver. Especially for temperature sensors, new chips are often variants of previously released chips. In some cases, a presumably new chip may simply have been relabeled.