Broadcom Starfighter 2 Ethernet switch driver

Broadcom's Starfighter 2 Ethernet switch hardware block is commonly found and deployed in the following products:

- xDSL gateways such as BCM63138
- streaming/multimedia Set Top Box such as BCM7445
- Cable Modem/residential gateways such as BCM7145/BCM3390

The switch is typically deployed in a configuration involving between 5 to 13 ports, offering a range of built-in and customizable interfaces:

- single integrated Gigabit PHY
- quad integrated Gigabit PHY
- quad external Gigabit PHY w/ MDIO multiplexer
- integrated MoCA PHY
- several external MII/RevMII/GMII/RGMII interfaces

The switch also supports specific congestion control features which allow MoCA fail-over not to lose packets during a MoCA role re-election, as well as out of band back-pressure to the host CPU network interface when downstream interfaces are connected at a lower speed.

The switch hardware block is typically interfaced using MMIO accesses and contains a bunch of sub-blocks/registers:

- `SWITCH_CORE`: common switch registers
- `SWITCH_REG`: external interfaces switch register
- `SWITCH_MDIO`: external MDIO bus controller (there is another one in SWITCH_CORE, which is used for indirect PHY accesses)
- `SWITCH_INDIR_RW`: 64-bits wide register helper block
- `SWITCH_INTRL2_0/1`: Level-2 interrupt controllers
- `SWITCH_ACB`: Admission control block
- `SWITCH_FCB`: Fail-over control block

Implementation details

The driver is located in `drivers/net/dsa/bcm_sf2.c` and is implemented as a DSA driver; see `Documentation/networking/dsa/dsa.rst` for details on the subsystem and what it provides.

The SF2 switch is configured to enable a Broadcom specific 4-bytes switch tag which gets inserted by the switch for every packet forwarded to the CPU interface, conversely, the CPU network interface should insert a similar tag for packets entering the CPU port. The tag format is described in `net/dsa/tag_brcm.c`.

Overall, the SF2 driver is a fairly regular DSA driver; there are a few specifics covered below.

Device Tree probing

The DSA platform device driver is probed using a specific compatible string provided in `net/dsa/dsa.c`. The reason for that is because the DSA subsystem gets registered as a platform device driver currently. DSA will provide the needed device_node pointers which are then accessible by the switch driver setup function to setup resources such as register ranges and interrupts. This currently works very well because none of the of_* functions utilized by the driver require a struct device to be bound to a struct device_node, but things may change in the future.

MDIO indirect accesses

Due to a limitation in how Broadcom switches have been designed, external Broadcom switches connected to a SF2 require the use of the DSA slave MDIO bus in order to properly configure them. By default, the SF2 pseudo-PHY address, and an external switch pseudo-PHY address will both be snooping for incoming MDIO transactions, since they are at the same address (30), resulting in some kind of "double" programming. Using DSA, and setting `ds->phys_mii_mask` accordingly, we selectively divert reads and writes towards external Broadcom switches pseudo-PHY addresses. Newer revisions of the SF2 hardware have introduced a configurable pseudo-PHY address which circumvents the initial design limitation.

Multimedia over CoAxial (MoCA) interfaces

MoCA interfaces are fairly specific and require the use of a firmware blob which gets loaded onto the MoCA processor(s) for packet processing. The switch hardware contains logic which will assert/de-assert link states accordingly for the MoCA interface whenever the MoCA coaxial cable gets disconnected or the firmware gets reloaded. The SF2 driver relies on such events to properly set its MoCA interface carrier state and properly report this to the networking stack.

The MoCA interfaces are supported using the PHY library's fixed PHY/emulated PHY device and the switch driver registers a `fixed_link_update` callback for such PHYs which reflects the link state obtained from the interrupt handler.

Power Management

Whenever possible, the SF2 driver tries to minimize the overall switch power consumption by applying a combination of:

- turning off internal buffers/memories
- disabling packet processing logic
- putting integrated PHYs in IDDQ/low-power
- reducing the switch core clock based on the active port count
- enabling and advertising EEE
- turning off RGMII data processing logic when the link goes down

Wake-on-LAN

Wake-on-LAN is currently implemented by utilizing the host processor Ethernet MAC controller wake-on logic. Whenever Wake-on-LAN is requested, an intersection between the user request and the supported host Ethernet interface WoL capabilities is done and the intersection result gets configured. During system-wide suspend/resume, only ports not participating in Wake-on-LAN are disabled.