Phylink and SFP: Going Beyond 1G Copper

Andrew Lunn <andrew@lunn.ch>

October 30, 2018

1 Introduction

The move from Fast Ethernet to 1Gbps Ethernet is well in the past now. It is time for multi-gigabit Ethernet to become the new defacto standard. However, it is not clear cut what exact bandwidth will take the place of 1Gbps. 2.5Gbps has the advantage of working over existing Cat 5e cables which are widely deployed. Cat 6 cables, which are also well deployed, can support 5Gbps. 10Gbps however requires Cat 7 cables. Fiber optics is also starting to become more popular. Fiber to the premises is slowly being deployed in many countries. Embedded applications of fiber are becoming more common, where lighter cables can be an advantage, e.g. in aerospace, or immunity to EMC issues in industrial settings.

Over the last few months the following MAC drivers were submitted to the netdev community. They clearly show the progression towards multi-gigabit:

- Marvell Octeontex2: 2.5G, 5G, 10G, 20G, 25G, 40G, 50G, 100G
- Intel IGC: 2.5G
- Freescale DPAA2: 10G
- Aquantia AQC111 USB dongle: 2.5G, 5G
- DEC TURBOchannel FDDI, 100Mbps.

During the time these drivers were submitted, no 1Gbps drivers were submitted. The FDDI driver is a clear outlier to the pattern. But it also shows that so long as somebody is willing to submit quality code to the netdev community, and provide maintenance, even support for classic hardware will be merged.

Multi-gigabit hardware however brings with it changes to the MAC-PHY link. And these changes cause problems with the existing phylib core infrastructure used by most MAC drivers. And there is no core support for SFP at all.

2 History

Two boards were instrumental in the development of PHYLINK and SFP support in the Linux kernel. The first board is the Solidrun Clearfog, Figure 1. It uses a Marvell SoC, combined with a Marvell Ethernet switch. However the more interesting part was the SFP cage, connected to a 1Gbps Ethernet port of the SoC. Using a suitable module, this allows fiber optical cables to be used, or an RJ-45 copper cables. The second board is the SolidRun MACCHIATObin, Figure 2. This again uses a Marvell SoC. However this SoC has 10G and 2.5G Ethernet ports, and 10G PHYs, as well as SFP cages.

Russell King took on the task of providing mainline support for these two boards. Neither 10G or SFP was new to Linux. However these could of been the first boards where Linux was responsible for controlling the SFP and PHYs. Previous boards have made use of firmware running in a secondary processor. The problems with phylib soon became clear, and Russell decided that a new framework would be needed to properly support these devices, building upon top of the existing phylib and PHY drivers.

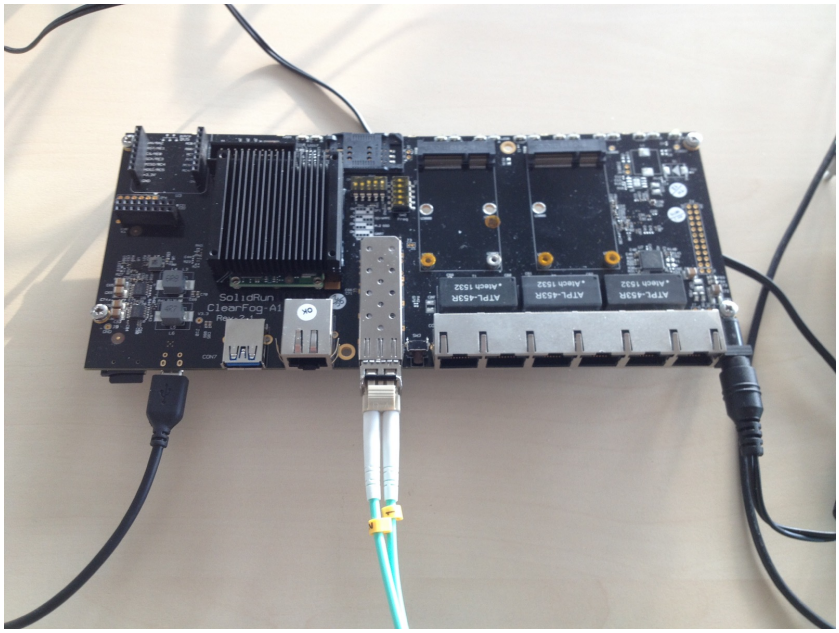

Figure 1: SolidRun Clearfog

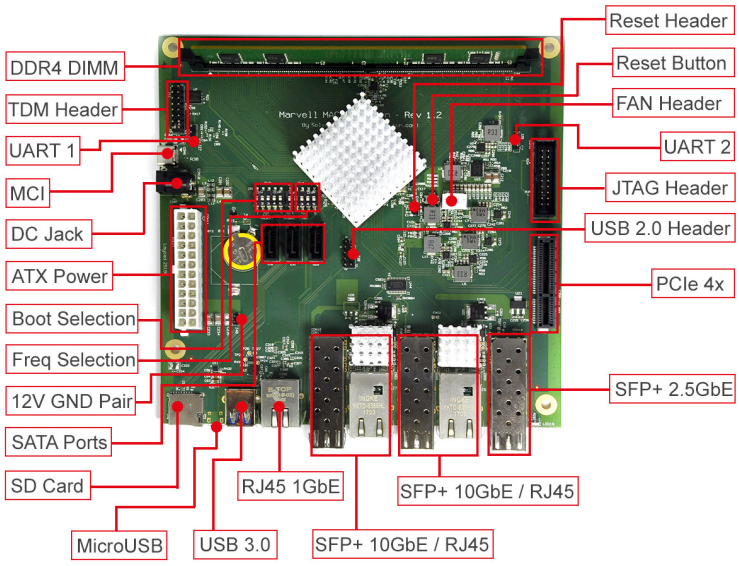

Figure 2: Solidrun MACCHIATObin

Figure 3: SFP Cage and Module

3 Small Form Factor, Pluggable

SFPs have been present in Ethernet switches for many years. It is only recent that they have started to appear in embedded systems. These comprise of two parts, a cage mounted on the board, and a hot-pluggable module, as shown in Figure 3. A second form also exists, where the module is soldered directly onto the board, known as an SFF.

The module uses a pair SERDES lines for the data plane, transporting Ethernet frames. The control plane is multi-format. An I2C bus hosts an AT24 like EEPROM device. However, only part of the memory range is actually EEPROM. Part of the memory contains dynamic information. The fixed part describes the modules capabilities, such as its manufacturer, serial number, wavelength, maximum baud rate, and maximum transmit power. The dynamic part can include the current module temperature, transmit and receive power, and operating voltage. Optionally, it can also have alarm thresholds, and an indicator if these thresholds have been exceeded. If a Copper RJ-45 module is used, as opposed to the more traditional optical fiber, the module needs to contain a Copper PHY. This is then controlled using an MDIO bus encapsulated over I2C. In addition to I2C, a number of signals from the module/cage are connected to GPIO lines of the SoC. These indicate Loss Of Signal, Transmit Error, and module presence. An GPIO output can also be used to disable the transmitter.

An optical SFP module is passive. The SERDES bitstream is converted into pulses of light. The received light is converted back into a SERDES bitstream. The SFP makes no attempt to interpret the bitstream. Also, no auto-negotiation is performed, as is typical of a copper PHY. The LOS is not a good indicator of link. Any light of the correct wavelength is sufficient to deassert LOS.

The passive nature of SFPs leads to one of the problems with the current phylib core. Phylib assumes an active device, and a device connected over MDIO, not I2C. For SFPs, the active elements in determining if a link can be established is actually a part of the MAC. Figure 5 shows the end to end, MAC to MAC link.

These diagrams are based on IEEE 802.3 terms. The MAC itself is connected to a Reconciliation Sublayer. This is just a glue layer to the PCS, the Physical Coding Subsystem. The PCS implements what is often known as the SERDES. It converts the byte stream from the MAC into a differential pair bitstream interface. It also performs encoding, such as 8B/10B. A simple signaling protocol is performed over this encoding, as well as synchronization. It is the PCS block which determines if there is an end-to-end link to the link partner. Phylib has no mechanism to allow the MAC to indicate the link is up, it assumes the PHY will determine this.

An additional problem is correctly configuring the link between the MAC and the SFP. The EEPROM in the SFP contains the maximum baud rate. It does not however indicate a list of link mode. The new SFP subsystem will determine all the link modes which can operate with the available maximum baud rate. e.g. a 4200MBd device, typical for Fiber Channel, can support 2500Base-X and 1000Base-X. The Clearfog MAC however only

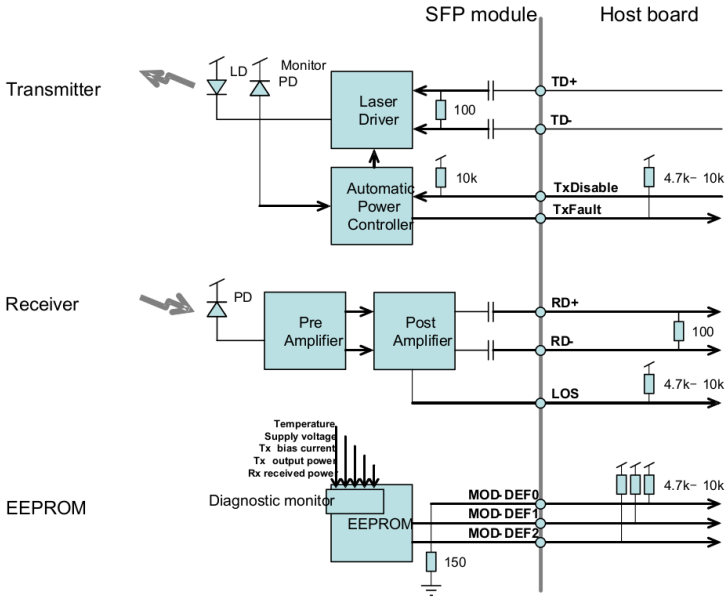

Figure 4: Block diagram of an Optical Fiber SFP module

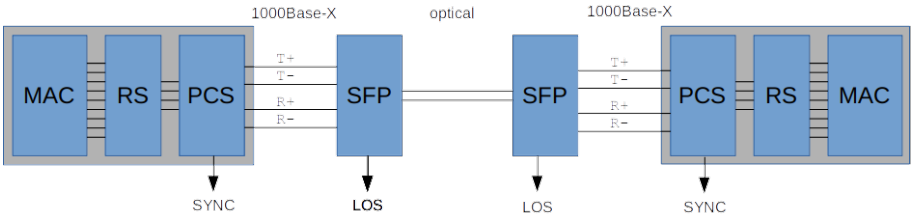

Figure 5: End-to-End SFP Link

supports 1000Base-X Thus there needs to be a negotiation performed to limit the choice to only what both the MAC and the SPF supports. It should be noted that fiber PCS cannot perform auto-negotiation. By default, the fastest supported link mode will be use. The user has to configure the MAC to a slower mode if the link partner is not compatible with this. When a copper RJ-45 module is used, a different choice of link modes is needed. Typically the copper PHY embedded inside the module only supports SGMII. The MAC needs to be configured to enable its SGMII PCS. Phylib is poorly suited to this task.

4 Multi-gigabit PHYs

1G MAC and PHY combinations traditionally make use of GMII, or RGMII between the MAC and the PHY. This is a simple clocked parallel bus structure. No additional protocol is run over this bus, just the Ethernet frame data. The configuration is fixed, and the bus is available as soon as PLL clocks are locked. Multi-gigabit PHYs however are not so simple. At these speeds, parallel buses don't work so well. Instead differential pair serial lines are used, with some form of encoding. The PCS layer is reused to implement the SERDES link between the MAC and the PHY, as shown in Figure 6. The PHY uses a PCS to implement its end of the SERDES interface. A second PCS block then converts the bytestream as required for the on the wire format. The Physical Medium Attachment, PMA, and the Physical Medium Dependent, PMD, then connect the bitstream to the wire.

Auto-negotiation is used with copper PHYs, unlike fiber SFPs. Auto-negotiation is performed by the PCS in the PHY.

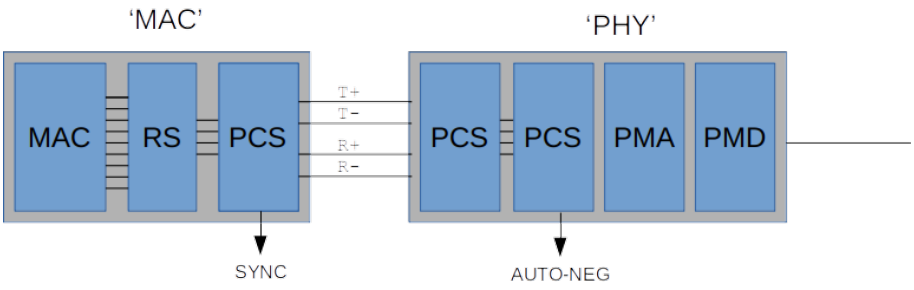

Figure 6: Multi-Gigabit PHY

1. The local and link partner PCS blocks determine what link mode should be used over the cable. This is then signaled over MDIO to the PHY driver.

2. The PHY configures its side of the SERDES link to the MAC with the selected link mode.

3. The MAC then must configure the MACs PCS block with the selected link mode.

4. After a short time, the MACs PCS will report the MAC-PHY link is up.

At this point, the end-to-end link will is declared up, and frames can start flowing. Phylib cannot perform this multi-stage link up processing. The choice of link modes can also be complex. Often SGMII is used for up to 1G links. 2.5G can either be implemented using 2500SGMII, which is SGMII clocked at a higher rate, or using 2500Base-X. Higher speeds then use the corresponding Base-X link mode between the MAC and the PHY and Base-T between the PHY and the link partner PHY. Some negotiation is request between the MAC and PHY to make this work. This is particularly true when the MAC only supports a subset of link modes of what the PHY supports, for example some Marvell MACs in there SOHO Ethernet switches support 2.5G and 10G, but not 5G. However Marvells multi-gigabit PHYs do support 5G.

5 Phylib API and its Limitations

Phylib is the classic API used by MAC drivers to interface with PHYs. A `struct phy_device` describes a PHY. The MAC driver needs to connect itself to a PHY, using `phy_connect()`, `of_phy_connect()` or a variant. When the MAC driver is removed, or suspended, it should disconnect from the PHY, using `phy_disconnect()`. When the interface is configured up, `phy_start()` is used to trigger configuration of the PHY, and probably start auto-negotiation. When the interface is configured down, `phy_stop()` is used to power down the PHY. Lastly, when the PHY experiences a change in state, e.g. auto-negotiation is complete, or the link goes down, the MACs `adjust_link()` callback is called, so it can take the appropriate action.

This classic API works well for 10/100/1G Half/Full duplex copper PHYs. It also works well for managing the internal state machine of multi-gigabit Ethernets, auto-negotiation, etc. It however is not sufficient to make a good MAC API for multi-gigabit Ethernets, nor can it support SFP devices. The details of API make the assumption that the signaling to the PHY is performed over MDIO. PHYs are cold plug devices. When the MDIO bus driver is instantiated, all PHY devices on the bus are found, and a `struct phy_device` created for each PHY. The MAC-PHY interface is configured once during the `phy_connect()` call, and never changed. The MAC is not really involved in determining if the link is up. It is assumed the PHY will determine this, and via the `adjust_link()` callback inform the MAC.

6 Phylink API

SFP have much more dynamic behavior than traditional PHYs. They are hot-pluggable, so they come and go. They can be passive fiber optical devices, or contain an active traditional copper PHY, requiring signaling. The MAC-PHY link is dynamic, depending on what baud rates the SFP supports. A multi-gigabit PHY also has a dynamic MAC-PHY link, with the configuration depending on the result of auto-negotiation to the link partner, and the common link modes supported by the MAC and the PHY. All this dynamic behavior is what resulted in the development of the Phylink API. It should be noted that Phylink does not replace phylib and the existing PHY drivers. Phylink will internally make use of the phylib state machine and PHY drivers when appropriate to driver a copper PHY.

In many ways, the phylink API is very similar to the phylib API. This is intentional, making the transition simple. A `struct phylink` is used to represent the abstract PHY and its link to the MAC. The MAC needs to create its phylink representer by calling `phylink_create()` during probe, and destroy it during remove using `phylink_destroy()`. As part of creating the phylink, the device tree properties will be parsed to determine if an SFP is being used, a fixed-link, and if in-band signaling between the MAC and PHY is to be used. If an SFP is not being used, `phylink_connect_phy()` or `phylink_of_phy_connect()` is used to connect the MAC to a PHY. Phylink will instantiate a phylib state machine to drive the PHY.

When the interface is configured up, `phylink_start()` is called. For a copper PHY, either direct, or via an SFP, the PHY is configuration and auto-negotiation probably started. For a fiber SFP, the TX disable will be deasserted. And if the cage is empty, not a lot happens except the state is tracked for when a module is inserted. `phylink_stop()` is called when the interface is configured down.

There is one important additional API call for the MAC, `phylink_mac_change()`. The MAC uses this when it determines its SERDES device has established or lost link. This could be all the way to the link partner for an SFP, or just to the local multi-gigabit PHY.

The single phylib `adjust_link()` callback has been replaced by a number of callbacks.

`validate()` is called by phylink to allow the MAC driver to remove link modes and other configurations it does not support. This is the primary mechanism to ensure the MAC-PHY link operates in a mode both ends support.

`mac_link_state()` is called by phylink to determine how the MAC sees the link, in terms of speed, duplex, up/down. This is particularly important when in-band signaling is used, e.g. SGMII.

`mac_an_restart()` is used to trigger 802.3z BaseX auto-negotiation.

`mac_config()` is used to configure the MAC and the MAC-PHY link. This is very similar to phylib's `adjust_link()` when a PHY is being used.

`mac_link_down()` is used to force the link down. This is often called when the SFP is ejected, or the PHY indicates the link is down.

`mac_link_up()` is used to allow the link to come up, e.g. an SFP has been inserted, or the PHY indicates it has link.

7 Examples and Documentation

There are a few MAC drivers which have made the transition from phylib to phylink. These provide good examples of how to make use of phylink.

The Marvell mvneta, the driver for the MAC interfaces on Clearfog, naturally leads the way. It makes full use of all the features of Phylink and SFPs, including many not mentioned in this overview.

The DSA core, used for implementing control of Ethernet switches, has also swapped to using phylink. Two of the switch drivers, mv88e6xxx and bcm_sf2, export the necessary interfaces towards the DSA core to make use of phylink.

The Marvell mvpp2 driver, as used by the MACCHIATObin is a bit behind in this transition.

Phylink and SFP are well documented using kerneldoc, which can be found at: `https://www.kernel.org/doc/html/latest/networking/kapi.html?highlight=sfp#phylink` and `https://www.kernel.org/doc/html/latest/networking/kapi.html?highlight=sfp#sfp-support`

8 SFP Freebies

Making use of central infrastructure not only results in using well tested and supported code, but often results in a few freebies. Things that just work, which individual drivers doing their own thing don't have, or need to implement and debug themselves. The SFP core provides a few of these.

8.1 Ethtool Support

`ethtool(1)` has the option `--module-info` option to dump the EEPROM contents of the SFP module. It can also decode the various fields into somewhat human readable text. It also understands the different formats used by different SFP standard revisions. There is currently more than 5 implementations of the kernel side code needed to wire up the MAC driver to provide this information. And the different implementations have subtle differences. The core SFP support also provides an implementation, and makes it usable without the MAC driver needing to do anything.

8.2 HWMON Support

An SFP contains a number of sensors, to measure the temperature of the module, the supply voltage, the current transmit power and the receive power. These values are part of the EEPROM contents and can be dumped using `ethtool(1)`. However, the Linux kernel has a subsystem to export sensor values, HWMON. The SFP driver has gained code to export these sensors via HWMON. The SFP EEPROM can optionally contain warn and alarm thresholds. Standard user space tools will then see these sensors. SFPs can consume a lot of power, and hence generate heat. For a box containing a number of SFPs, e.g. an industrial Ethernet switch, these SFPs can be the main source of heat. Having temperature sensors available using the standard HWMON interface then allows the thermal zone subsystem to be used, to control a fan, etc.

9 Conclusion

Over the last couple of years, there has been a trend of using firmware to control PHYs and SFPs in high end interfaces. But as yesterdays high end becomes tomorrows middle ground, multi-gigabit becomes normal for all systems and we can expect Linux to take back control of the PHYs and SFPs.

Although it is possible to write a MAC driver for a multi-gigabit interface using phylib, phylink provides a much more natural interface for the dynamic MAC-PHY link. All future MAC drivers, which don't make use of firmware to control the PHY, should strongly consider using phylink. As of today, the number of multi-gigabit PHY drivers is very limited, but more are in the works, and there quality and features will in time exceed closed source firmware.

The same is true for any interface with an SFP, even if it is just a 1G SFP. Making use of the core SFP infrastructure means less code, less bugs, and more features.