USB TO 100BASE-FX OPTICAL FIBER NETWORK CARD

Optical fiber is an intriguing technology, deployed all over the world connecting computer networks with the speed of light (well… almost). Unfortunately, due to the inherent complexity of fiber network installation and management, optical fiber devices have never found their way in the hands of the user, and are usually deployed only by professionals for things such as backbones, long hauls or really fast interconnections.

Luckily enough, older optical fiber Ethernet components, especially 100MBit ones, are now available as a reasonably low price, so I decided to design a couple of USB to 100BASE-FX network cards just for fun and to learn more about working with optical fibers.

This project contains two complete hardware designs for USB to 100BASE-FX network cards, one with a 1×9 transceiver and one with an SFP slot. Both designs are based on the ASIX AX88772B chip, and fit in a compact Hammond 1551 series box.
Also, there are some useful links and information about designing with OF transceivers and SFP modules.

Background
The first thing you may ask is: why would you want to design a silly 100Mbps optical fiber network card when 100BASE-FX is being basically phased-out? (most Fast Ethernet PHYs do not support FX mode anymore) The answer is easy: that's the best you can design with in-house hobbyist grade tools (read: for fun).
For example, just stepping up to Gigabit Ethernet would require a much more complex design, involving a multi layer PCB with a tuned stackup for controlled impedance balanced lines (LVDS and LVPECL all over the place), and a couple of expensive BGA ICs with all the related availability issues… Not really something you want to use for a DIY project.

Optical Fiber Transceiver

The key component and most expensive part of the circuit is the optical fiber transceiver which, as you’ll see, is a self-contained device on its own, including the necessary circuits to drive the transmission LED or laser, the receiving logic, the actual fiber connector and, if hot-pluggable, an identification EEPROM.

A big issue with optical fiber is that there is no auto-negotiation at all, and there is a relatively high number of standards and proprietary extensions, covering any possible application with many combinations of bandwidth, range, connectors, wavelength, single-mode or multi-mode support, single-fiber options and WDM in general. In short, this means that the two transceivers at the ends of the fiber have to be checked for compatibility: just look at the number of transceivers available from DigiKey to understand the size of the problem.
Luckily enough, at least from a electrical design point of view, things are a bit easier as you mostly need to worry about two things: footprint and electrical signaling.

For the footprint, there are essentially three standards currently in use:
- 1×9 (PTH): that's the footprint of the oldest (and cheaper) transceivers, it's relatively bulky, available with both single-mode and multi-mode drivers, and usually comes with an SC-Duplex or SC (for single mode option) connector. That's what I used in my first design.
2×5 (PTH): that’s a modern footprint for fixed transceivers up to multi-gigabit speed. It’s low profile and usually comes with LC-Duplex or MTRJ connectors.

SFP: this is the current standard for hot pluggable transceivers, with a high speed variant (SFP+) supporting up to 10Gbps. SFP modules always include an identification I2C EEPROM.
All three footprints provide separate power paths for transmitter and receivers, two differential lines and a signal detect line for carrier detection.

As for the electrical levels, every transceiver datasheet specifies one or two possible matching networks, so you should study carefully the datasheets of all the possible transceivers you want to deploy, and implement a compatible version in your schematic.
Luckily enough, the SFP standard covers at least some essential electrical design aspects, such as AC coupling and terminations, but beware of the differences between the “Signal Detect” and “Loss of Signal” definitions: same thing, different polarity!

The differential lines usually carry two PECL or LVPECL pairs, but while the gigabit (and faster) transceivers require a dedicated driver on the PHY, 100Mbps ones are usually connected to the same pins as the standard Fast Ethernet pairs (the one you normally wire to the 1:1 transformer).

Fast Ethernet transceivers supporting FX operations just need to provide the additional “Signal Detect” pin and a configuration bit to disable the scrambler and MLT-3 encoder (only used in wired mode operation).

Sounds complicated? It is, but at least Vitesse was kind enough to design an ultra versatile PHY, the VSC8211, that supports (almost) all of the standards. They also published all of the documentation without NDA, including a great application note that covers many different configurations: SimpliPHY Dual Media Copper/Fiber/SFP.

The only drawback: the VSC8211 only comes in a funky rectangular 10x14mm BGA package… Not exactly well suited to a cheap 2 layer PCB, so let’s just stick to 100BASE-FX for the moment.

**USB to Fast Ethernet Controller**

Most USB to Ethernet controllers are used for cheap laptop adapters or embedded applications (SMSC ones are quite common on evaluation boards), and do not support FX operation.

The exception here is the **AXIS AX88772B**, that includes an embedded PHY with FX support enabled by a bit in the configuration EEPROM.

AXIS controllers are notoriously very reliable, and have some properly designed software drivers for most operating systems, especially for the Linux driver that includes EEPROM modification support through standard `ethtool` commands – and we need that to enable FX mode.

Finally, on the AXIS website you’ll find all the documentation, including the EEPROM layout, register description, and an example schematic and layout with a 1×9 100BASE-FX transceiver.

So far so good, where do we buy the IC from? DigiKey does not list it… Well, the best solution I have come up with, was to buy a couple of USB to Ethernet keys from **Olimex** and remove the controller and EEPROM using a hot air rework station. Can you do better?

**Enabling FX Operation**

The AX88772B will run in copper mode by default and to put it into fiber mode you need a configuration EEPROM to modify.
I already had a programmed EEPROM as I desoldered it from a working USB Ethernet key, so the first thing to do is a complete dump to be saved in a safe place, just in case. That’s how you do it (you need to be root, note the interface name is `eth1` for me):

```
# ethtool -e eth1

     Offset          Values
     ------          -------
           0x0000          15 5a ec 75 20 12 29 27 00 21 f3 88 72 f0 09 04
           0x0010          60 22 71 12 19 0e 3d 04 3d 04 3d 04 3d 04 80 05
    ...
```

Here we are interested in the low byte of the “Flag” register, and that’s at offset 0x02. The bit five \((1 << 5)\) is defined in the datasheet as:

<table>
<thead>
<tr>
<th>EPOM: Embedded PHY copper/fiber Operation Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Sets embedded PHY in copper mode (default)</td>
</tr>
<tr>
<td>0: Sets embedded PHY in fiber mode</td>
</tr>
</tbody>
</table>

So we check that we are pointing at the right byte with:

```
# ethtool -e eth1 offset 2 length 1

     Offset          Values
     ------          -------
           0x0002          ec
```

And finally we change that register to mask out our bit with:

```
# ethtool -E eth1 magic 0xdeadbeef offset 2 length 1 value 0xcc
```

If you get no error messages, congratulations! Your AX88772B is now running in FX mode!

The magic number is defined in the kernel driver, and it’s used to ensure that you are pointing to the right device.

**Design Issues**

The actual schematics for the converter are quite simple and mostly based on the reference design. You may want to keep into account that the optical transceiver uses a significant amount of power to operate (I measured up to ~1.8W from USB in my tests), and a minimum of dissipation area for the regulator is necessary for continuous operation. An even better approach would be to include a proper switching regulator.
The only issue I have found is figuring out if the polarization network was needed or not for the SFP version. I did not have the datasheet for the transceiver I used, and the one I found on the net had the polarization network inside, so I just left it out of the design.

Turns out I was wrong, but nothing that can't be fixed with a bit of wire and some resistor.

1×9 Version

The first version of this project uses an Avago AFBR-5803Z, and was tested using an old PCnet32+ based network card bought on eBay (which mounted the Agilent component: the HFBR5803).

Here are some pictures of the board, mounted inside of a Hammond 1551R box.
The second version is more interesting, as it’s using an SFP transceiver from River Stone network that I got from eBay for just USD $5!

The SFP cage uses press-fit tabs to lock on the PCB, and that’s great, but I had to modify it a bit to fit in the Hammond 1551H box... the end result is a really compact design.
Curious of what's inside the transceivers? Me too!
This is a picture of the Avago module:
And this is under the cage of the River Stone SFP module: