JEENODE, NETWORK

EtherCard library API

In Software on Jun 19, 2011 at 00:01

As you may have noticed in the last few weblog posts, the API of the EtherCard library has changed quite a bit lately. I'm not doing this to be different, but as part of my never-ending quest to try and simplify the calling interface and to reduce the code size of the library (these changes shaved several Kb off the compiled code).

The main change was to switch to a single global buffer for storing an outgoing Ethernet packet and for receiving the next packet from the controller. This removes the need to pass a buffer pointer to almost each of the many functions in the library.

Buffer space is scarce on an ATmega, so you have to be careful not to run out of memory, while still having a sufficiently large buffer to do meaningful things. The way it works now is that you have to allocate the global buffer in your main sketch:

```
byte Ethernet::buffer[700];
```

This particular style was chosen because it allows the library to access the buffer easily, and more importantly: without requiring an intermediate pointer.

To make this work, you have to initialize the EtherCard library in the proper way. This is now done by calling the *begin()* function as part of your *setup()* code:

```
if (ether.begin(sizeof Ethernet::buffer, mymac) == 0)
  Serial.println( "Failed to access Ethernet controller");
```

The *ether* variable is defined globally in the EtherCard.h header file. The *begin()* call also needs the MAC address to use for this unit. The simplest way to provide that is to define a static array at the top of the sketch with a suitable value (it has to be unique on your LAN):

```
static byte mymac[] = { 0x74,0x69,0x69,0x2D,0x30,0x31 };
```

Next, you can use DHCP to obtain an IP address and locate the gateway and DNS server:

```
if (!ether.dhcpSetup())
  Serial.println( "DHCP failed");

ether.printIp("My IP: ", ether.myip);
ether.printIp("Netmask: ", ether.mymask);
ether.printIp("GW IP: ", ether.gwip);
ether.printIp("DNS IP: ", ether.dnsip);
```

The *printIp()* utility function can optionally be used to print some info on the Serial port.

If you are going to set up a server, then a fixed IP address might be preferable. There's a new *staticSetup()* function you can use when not doing DHCP:

```
ether.staticSetup(myip, gwip, dnsip);
```

The gateway IP address is only needed if you're going to access an IP address outside of your LAN, and the DNS IP addres is also optional (it'll default to Google's "8.8.8.8" DNS server if you do a DNS lookup). To omit values, pass a null pointer or leave the arguments off altogether:

```
ether.staticSetup(myip);
```

Just remember to call either *dhcpSetup()* or *staticSetup()* after the *begin()* call.

DNS lookups are also very simple:

```
if (!ether.dnsLookup(website))
  Serial.println("DNS failed");

ether.printIp("Server: ", ether.hisip);
```

The one thing to keep in mind here, is that the *website* argument needs to be a *flash-based* string, which must be defined as follows:

```
char website[] PROGMEM = "www.google.com";
```

Note the "PROGMEM" modifier. See the Saving RAM space weblog post for more info about this technique.

This concludes the intialization part of the EtherCard library. Next, we need to keep things going by frequently polling for new incoming packets and responding to low-level ARP and ICMP requests. The easiest way to do so is to use the following template for *loop()*:

```
void loop () {
  word len = ether.packetReceive();
  word pos = ether.packetLoop(len);
  ...
}
```

The *packetReceive()* function polls for new incoming data and copies it into the global buffer. The return value is the size of this packet (or zero if there is none).

The *packetLoop()* function looks at the incoming data and takes care of low-level responses. The return value is the offset in the global packet buffer where incoming TCP data can be found (or zero if there is none).

As to what to do next: it really all depends on what you're after. Check out the examples in the Ethercard library for how to build web servers and web clients on top of this functionality.

To get an idea of the code overhead of the EtherCard library: a simple web client using DHCP and DNS is around 10 Kb, while an even simpler one using static IP addresses (no DHCP and no DNS) is under 7 Kb. The fairly elaborate EtherNode sample sketch, which includes DHCP and the RF12 library is now ≈ 13 Kb.

IOW, lots of room for adding your own app logic!