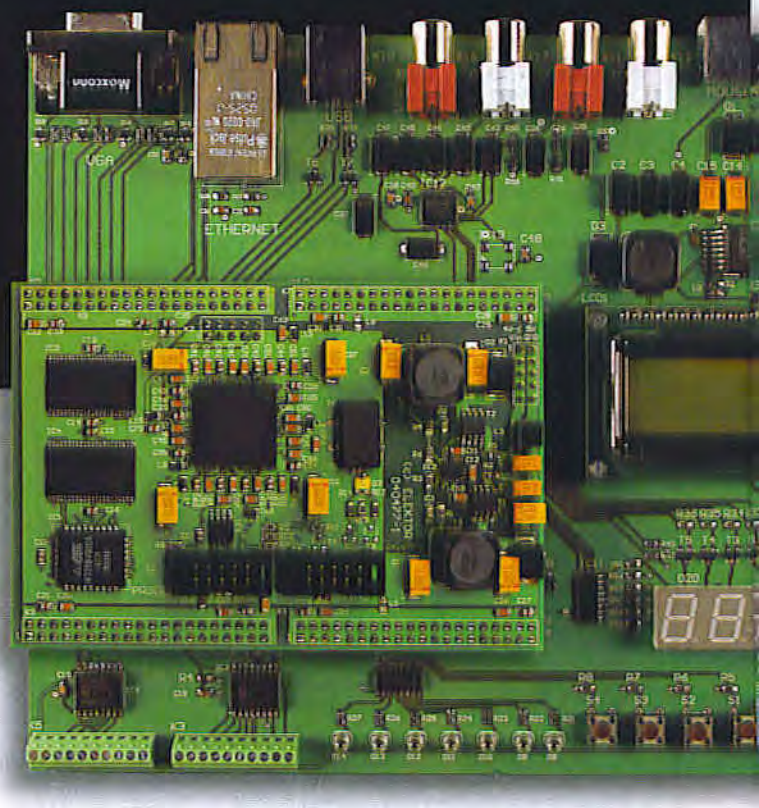
FPGA Course (9)

Part 9 : The final act



Paul Goossens &
Andreas Voggeneder

We've used nearly every I/O port of the prototyping board at least once in this FPGA course. We say 'nearly' because we haven't used the Ethernet port up to now. The final instalment of the course sets this right with a full-fledged webserver application.

The prototyping board has an Ethernet connector and associated Ethernet PHY. This makes it possible to connect the FPGA to an Ethernet network, and ultimately to the Internet.

Ethernet

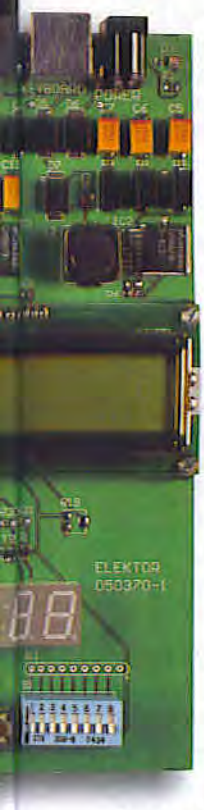
An Ethernet network can be used to interconnect several computers and other devices. The Internet is the largest and best-known example of an Ethernet network. This article shows you how to connect the FPGA module to an Ethernet network. This can be a small local area network in you home, but it can also be the Internet.

Various sorts of logic in the FPGA module are necessary for this purpose. Although the FPGA prototyping board has an Ethernet connector and Ethernet PHY, it takes quite a bit more than this to make an Ethernet connection. The task of the Ethernet PHY is to transmit and receive data in serial format according to the Ethernet standard. Besides this PHY, you also need a media access controller (MAC). The functions of the MAC component are:

- automatic data flow control
- detecting data collisions and retransmitting data if something goes wrong (CSMA/CD)
- adding a CRC checksum
- adding a preamble on transmission and deleting the preamble on reception

For the experts among our readers, this is Layer 2 of the OSI model. Layer 1 consists of the combination of the PHY, the transformer and the connector.
You also need a microcontroller with the right software to add the TCP/IP protocol and the application to the system. Just as in the previous instalments, we use the T51 core for this purpose.
The software is based on the software for our MSC1210 webserver. We used the uC51 compiler and associated TCP/IP stack to compile the code. This compiler is available from Wickenhäuser (see inset).

MAC



The media access controller (MAC) comes from www.opencores.com, and it is written in Verilog. The MAC uses a 32-bit data bus, while the T51 works with an 8-bit data bus. The *ethernet_verilog.vhd* file was modified so these two busses can be connected together.
The details of how this MAC works fall outside the scope of this article. In any case, you don't need to understand the details to use the MAC properly. With the help of the software provided with the course, you can quickly write your own applications for the Ethernet port.

First example

Our first example (**ex24**) uses the T51 core, the MAC core, and some peripheral logic. The Ethernet core eats up a fair amount of the scarce memory in the FPGA, so the maximum amount of ROM available to the T51 is 14 kB. This is a rather small amount of memory for a webserver. Among other things, the entire TCP/IP stack, the control logic for the MAC and the PHY chip, and the webserver application must all be fitted into this 14 kB.
The RAM consists of an external SRAM IC that is already present in the FPGA module. This IC is connected to the microcontroller via the wishbone bus. The memory layout is shown in **Figure 1**. The first 52 kB of RAM are provided by the SRAM IC. Addresses 0xD000 through 0xDFFF are reserved for the Ethernet MAC. Finally, the remaining 8 kB is used as buffer memory for transmitted and received data packets.
Now the FPGA prototyping board has to be connected to a network. You can use a router or a switch for this. The TCP/IP address of this application is set to a fixed value of 192.168.0.1. Ensure that your network is configured such that address 192.168.0.1 is valid on the network. If you use a router, the TCP/IP address of the router must be configured such that the first three numbers are 192.168.0, and the last number must **not** be 1 since this number is already used by the FPGA module. If the router has a DHCP server, you will have to adapt it to the new TCP/IP addresses.
If everything goes the way it should, after you program the FPGA you can use the web browser on your PC to view the web pages in the FPGA by entering the following address (URL):

http://192.168.0.1

If this doesn't work, check the network settings of your PC and router.

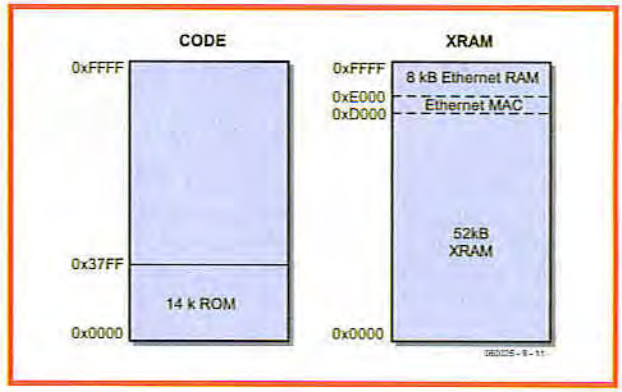


Figure 1. Memory layout for example ex24.

Even more

The webserver of example 24 has just enough oomph to publish a couple of HTML pages using the FPGA module. This is rather on the meagre side, and it certainly isn't what we really want. The limiting factor here is program memory.

If you can expand this memory, you can then write more 'meaningful' applications for the webserver. Fortunately, the FPGA module has some extra RAM and flash memory. If you use the flash memory to store the program code, in theory you can use programs up to 128 kB in size. This brings us up against a problem: you can program the memory in the FPGA using Quartus, but you can't program external memory.

Bank registers

Bank_En (0xF8)

Bit 4: 1 = VGA enabled; 0 = VGA disabled
(0x8000–0xBFFF)
Bit 5: 1 = SRAM always bank 0;
0 = SRAM depends on Bank_Sel
Bit 6: 1 = SRAM enabled; 0 = flash enabled
Bit 7: 1 = Ethernet disabled; 0 = Ethernet enabled
(0xD000–0xFFFF)

Bank_Sel(0xF9)

This register selects the current bank.

Features of example 25

- 8052 controller running at 25 MIPS
- 10/100 Mb/s Ethernet interface
- 256 kB RAM
- 128 kB flash
- VGA interface
- PS/2 interface
- Firmware updating via BOOTP

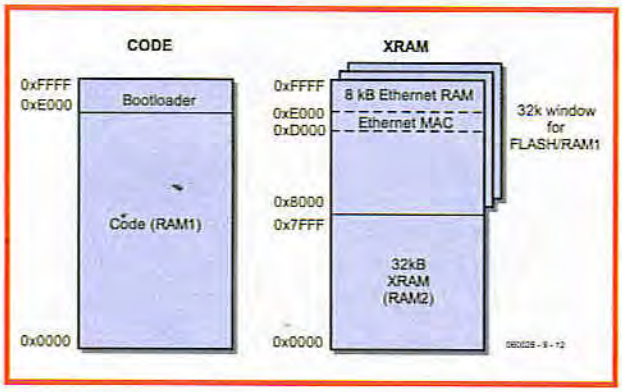


Figure 2. Memory layout for example ex25 when the bootloader is NOT executed.

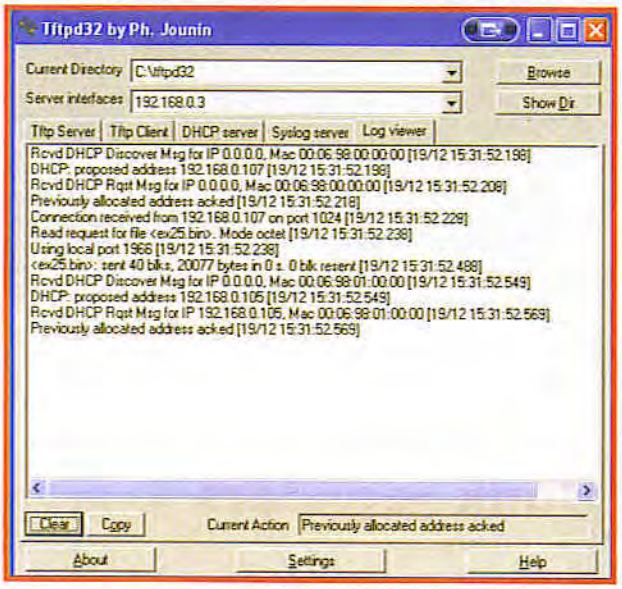


Figure 3. DHCP server settings in the program tftpd32.

µC-51

The software used in examples 24 and 25 was generated using the Wickenhäuser µC51 C compiler (www.wickenhaeuser.de).

We used this compiler on account of the TCP/IP stack that comes with the compiler. The free version of the compiler can generate programs with up to 8 kB of code, which is not enough for our software. If you wish to modify and recompile the software, you will need the full version of the compiler. The price of the compiler was € 84.50 (approx.£ 59) at the time of writing.

Up to now we have not been able to find a suitable TCP/IP stack for SDCC. If at some point we manage to find an open-source TCP/IP stack that is suitable for SDCC, we will make this known via our FPGA forum.

Naturally, we would also be grateful to hear about any suitable TCP/IP stack written by one of our readers or encountered by our readers somewhere on the Internet.

Boot loader

In example 25 (**ex25**), we connect the microcontroller to the flash memory and both SRAM chips. We also place a boot loader program in the internal ROM. If you set dipswitch 6 to the ON position and then reset the circuit by pressing button 1, this program will run.
The boot loader uses the DHCP protocol to try to obtain a valid TCP/IP address, such as the address of the router or PC. If this succeeds, it then uses the BOOTP protocol to request data. The data that the microcontroller receives in response to this request is stored in the external flash memory.

If the system is started up with dipswitch 6 in the OFF position, the content of the flash memory is copied to the SDRAM (IC4 on the prototyping board). The program code in the SDRAM is then executed. The reason for copying the program code is that the flash memory is relatively slow (access time 150 ns). The SDRAM has an access time of 10 ns and is thus much faster. The long access time of the flash memory makes it necessary to insert three wait states for each read instruction. As a result, the microcontroller effectively runs at 6.25 MHz instead of 25 MHz. This restriction is not necessary if the program code is executed from the SDRAM, so the microcontroller is free to operate at full speed.
It is not essential to store the program code directly in the flash memory when you are developing new software. For this reason, we added the option of storing program code in external SRAM (IC4) instead of flash memory. To do so, set dipswitch 7 to the ON position. The program will then be executed from SDRAM after the firmware has been downloaded.

RAM

The RAM memory is also expanded in example 25. To help clarify the following explanation, the memory layout of example 25 is shown in **Figure 2**. This layout is only valid if the boot loader is ***not*** running. A different memory layout is used when the boot loader is running, so that data can be written to the SRAM chip (IC4).
The memory layout can be configured using two SFR registers: Bank_En (0xF8) and Bank_Sel (0xF9).
Bit 4 of Bank_En determines whether VGA memory is addressable. If it is set to '1', VGA memory can be accessed in the address range 0x8000–0xBFFF.
Bit 7 of the same register determines whether the memory of the Ethernet interface is accessible as XRAM in the address range 0xD000–0xFFFF.
Bit 6 selects either flash or RAM for the upper 32 kB. If bit 6 is set to '0', flash memory can be accessed in the memory range 0x8000–0xFFFF. If this bit is set to '1', the content of the SRAM (IC3) is accessible in this address range.

Note that the VGA interface and the Ethernet interface have the highest priority. If either of these interfaces is enabled, the associated addresses cannot be used for the SDRAM or flash memory.
Bank-switching capability
is also provided to enable using the entire SDRAM and flash memory. You can use the Bank_Sel register to choose which part of the flash memory or SDRAM is selected. In case of the flash memory, this register determines which of the 32-kB blocks is currently in use. With the SDRAM, this register determines which 64-kB block is currently in use.

Down to work

That's enough explanation – now let's try example 25 in practice! Besides the usual software, you need one more program file for this, which is named *tftpd32*. It can be downloaded from the Internet free of charge (http://tftpd32.jounin.net). This program file contains a DHCP server and a BOOTP server. You need both of them in order to load software into the flash memory.
To start off, you have to ensure that your PC uses a fixed IP address. As already mentioned, we selected address 192.168.0.1 for this.

If you router has a DHCP server, you must disable it, since only one DHCP server is allowed in a network. If you use a router, it must also be configured with a fixed IP address (in our case, we used address 192.168.0.2 for the router).
After starting up *tftp32*, you have to configure the right DHCP server settings. In our example (**Figure 3**), we reserved addresses 192.168.0.105 through 192.168.0.115 for dynamic addresses. We also selected file *ex25.bin* as the BOOTP file.
The next step is to use Quartus to configure the FPGA. Make sure that dipswitch 6 is ON and dipswitch 7 is OFF. After the configuration is completed, the boot loader will start running and request an IP address. It will then use BOOTP to ask for a program. You can see this in the Log Viewer of the *tftp32* window.
If everything goes the way it should, the IP address of the FPGA will appear on the LCD screen. In our case, the FPGA was assigned address 192.168.0.107. Now you can locate the FPGA module in your web browser at address http://192.168.0.107. If your FPGA module is assigned a different address, simply enter the corresponding address in your web browser.

Pitfalls

The file that is sent vial BOOTP (the firmware for the microcontroller) must be a binary file. The µC51 compiler will generate a binary file automatically if you use the 'make' files provided with the course.
If you want to use software generated by a different compiler, make sure that it generates a binary output file. Most compilers can generate binary files. If you compiler can't do this, there are several freeware programs available that can convert hex files into binary files.

Conclusion

You can refine and extend the final embedded system of example 25 to generate your own applications. The features of this system are quite impressive, especially considering that the complete source code is available in VHDL. You can use this source code to enlarge your knowledge of VHDL.
We hope this course has helped you carry out your first practical experiments with programmable logic and VHDL.
The course is naturally far from complete – the subject of FPGAs and VHDL is so broad that you could easily write a whole set of books about it. Nevertheless, we think the course provides enough material – in part because of the numerous examples – to enable you to start developing your own designs in VHDL.

We have set up a separate topic in the forums section of our website so users of the FPGA module can share their experience. There's a good chance that you can pick up new ideas there, and you can also ask other readers for help with any problems you may encounter.
In any case, we hope you have a lot of fun (and learn something) with the FPGA module!

(060025-9)

DHCP and BOOTP

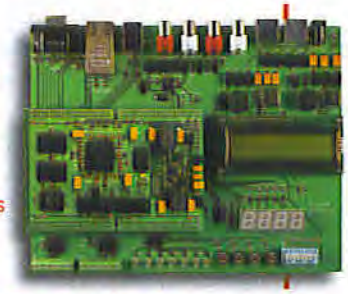
The DHCP protocol can be used to assign dynamic addresses to devices with Ethernet ports. Each device in a network requires a unique IP address so it can communicate with other devices. This can be achieved by manually assigning each PC or device a unique address. However, people quickly realised that it would be convenient to have IP address configuration be handled automatically by a server.

The protocol for this is called the DHCP protocol. It requires a DHCP server. Most routers for home use have built-in DHCP servers.

Another protocol that has been developed is the BOOTP protocol. This protocol makes it possible to boot PCs from a network if they do not have internal hard disks. If a BOOTP request is sent to a server, it uses the same protocol to send a small program back to the PC. The PC runs this program after receiving it. This program usually enables the PC to continue starting up from the server. This makes it a lot easier for system managers to supply new software to all PCs on the network in a single operation or install updates.

In our example, we use BOOTP to send new firmware to the FPGA module. This firmware is loaded into flash memory or SDRAM (depending on the position of dipswitch 7) and then executed.

Join the FPGA Course with the Elektor FPGA Package!



The basis of this course is an FPGA Module powered by an Altera Cyclone FPGA chip, installed on an FPGA Prototyping Board equipped with a wealth of I/O and two displays (see the March 2006 issue).

Both boards are available ready-populated and tested. Together they form a solid basis for you to try out the examples presented as part of the course and so build personal expertise and know-how in the field of FPGAs.

Further information may be found on the shop/kits & modules pages at www.elektor-electronics.co.uk