WIZnet iMCU Design Contest 2010 - Entry # 003179

Net Butler

The Net Butler is a multifunction network appliance designed around the WIZnet W7100 internet MCU that takes care of all the housekeeping tasks that help make maintaining a home network more convenient. It also adds several features that would normally require additional network hardware. Like a real butler, it is adept at performing the same tasks over and over again without needing to be reminded, and is easily upgradeable and adaptable to handle new tasks as they come along.

It's currently set up to do the following:

- DNS proxy with domain name block list and activity log display
- Download and display daily and weekly weather summary
- Track and report currently and previously connected network devices
- Web server for viewing system activity and configuration settings
- Enable and disable wifi network by local pushbutton, web page, or timer
- Control power for two printers by local pushbutton, web page, or timer
- Print server for a USB printer

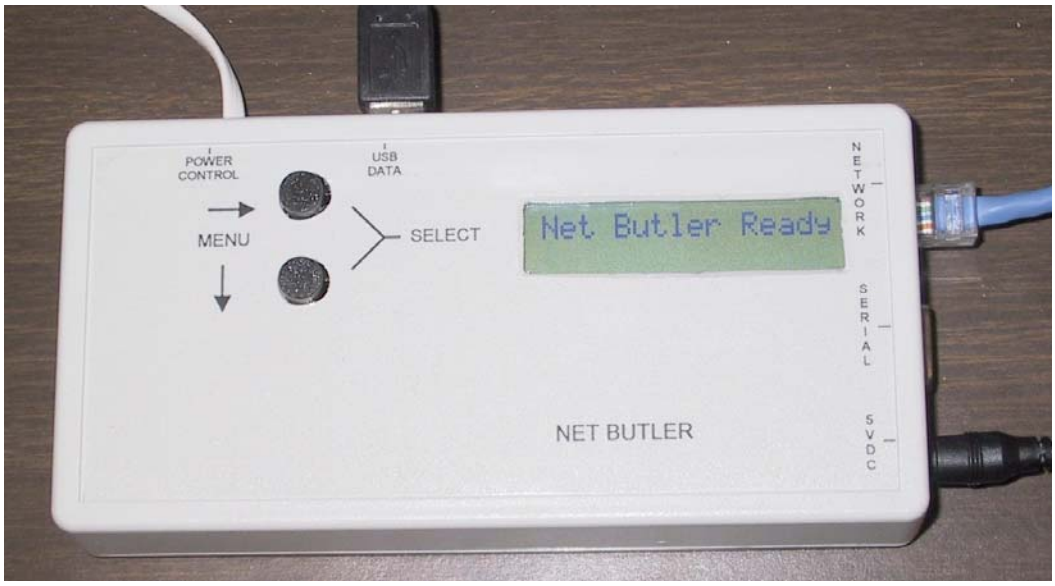
The System

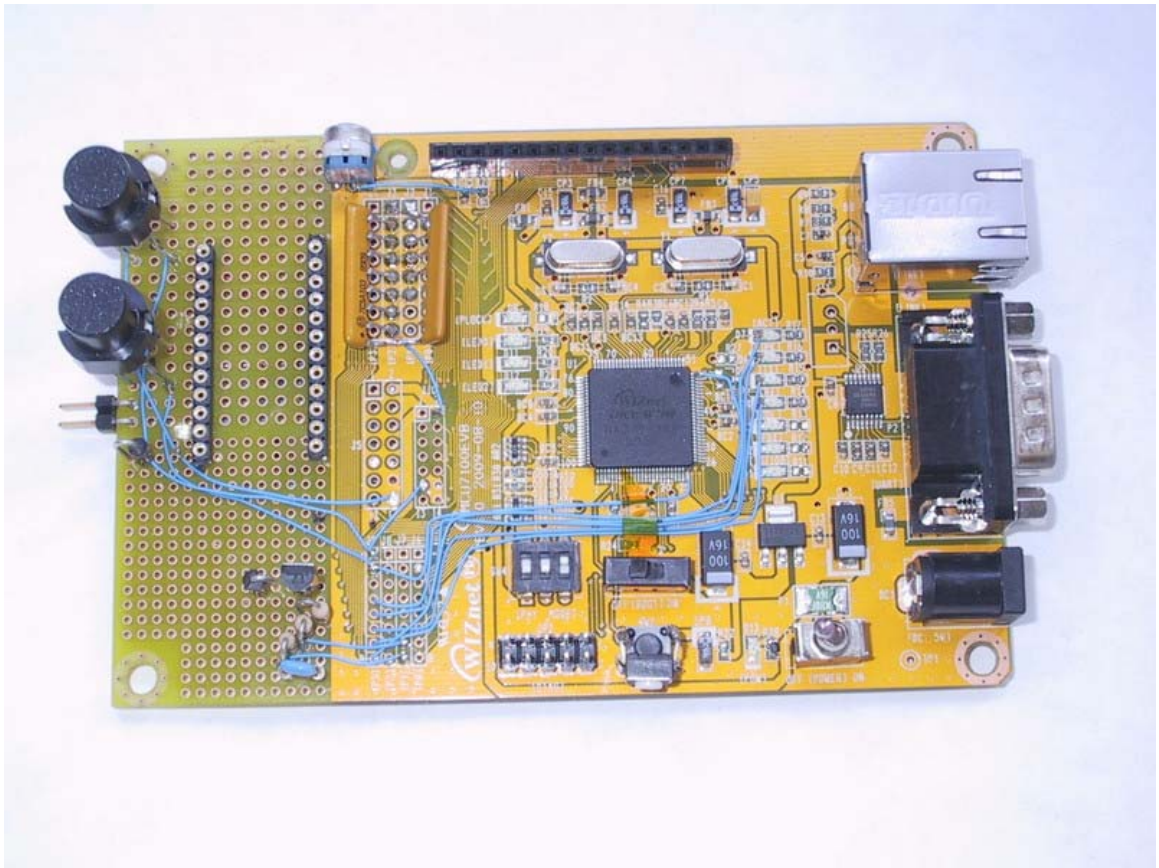
The system is built around an iMCU7100EVB evaluation board, which contains a W7100 iMCU and support circuitry. It has ethernet and serial ports, and an LCD display. I added a USB interface module, a pair of pushbuttons, and an output driver to control a relay. Power comes from a 5V regulated wall adapter.

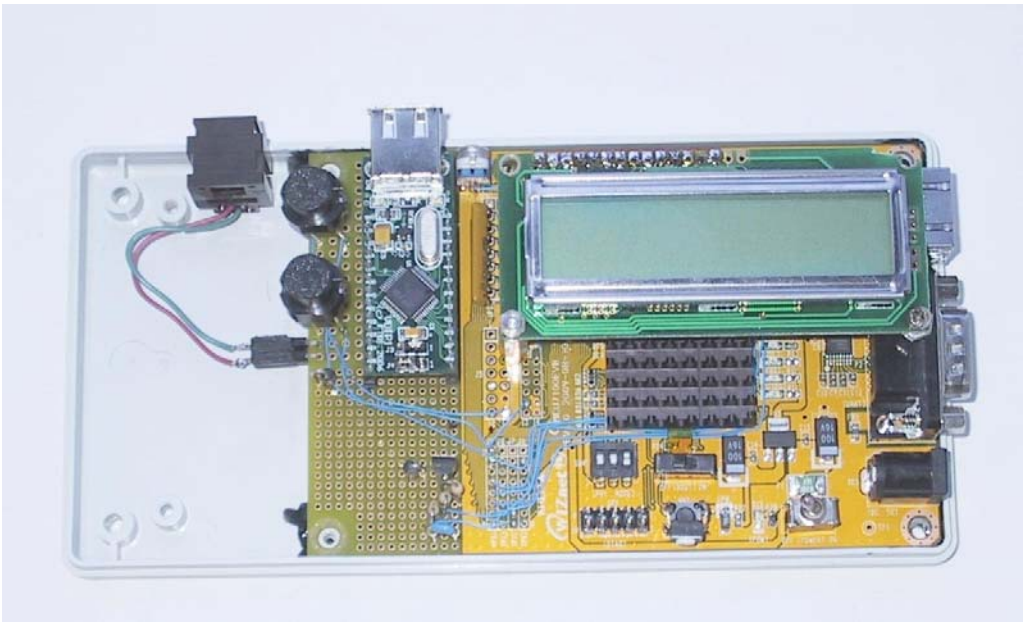
The code was written in C, using IAR Embedded Workbench for 8051. The code uses almost all of the 64K bytes of Flash, all of the 64K bytes of RAM, and all eight of the network sockets along with all of the socket memory. There is a separate bootloader in Flash memory which can download new code over the network using TFTP and restart the system in only a few seconds.

Each major function consists of an application routine and a separate protocol engine, making it easy to add new applications as new functions are required. The protocols it currently supports are ARP, DHCP client, DNS server & client, HTTP server & client, ICMP, and TELNET. The code is written so all protocols and functions can be active simultaneously without interfering with each other.

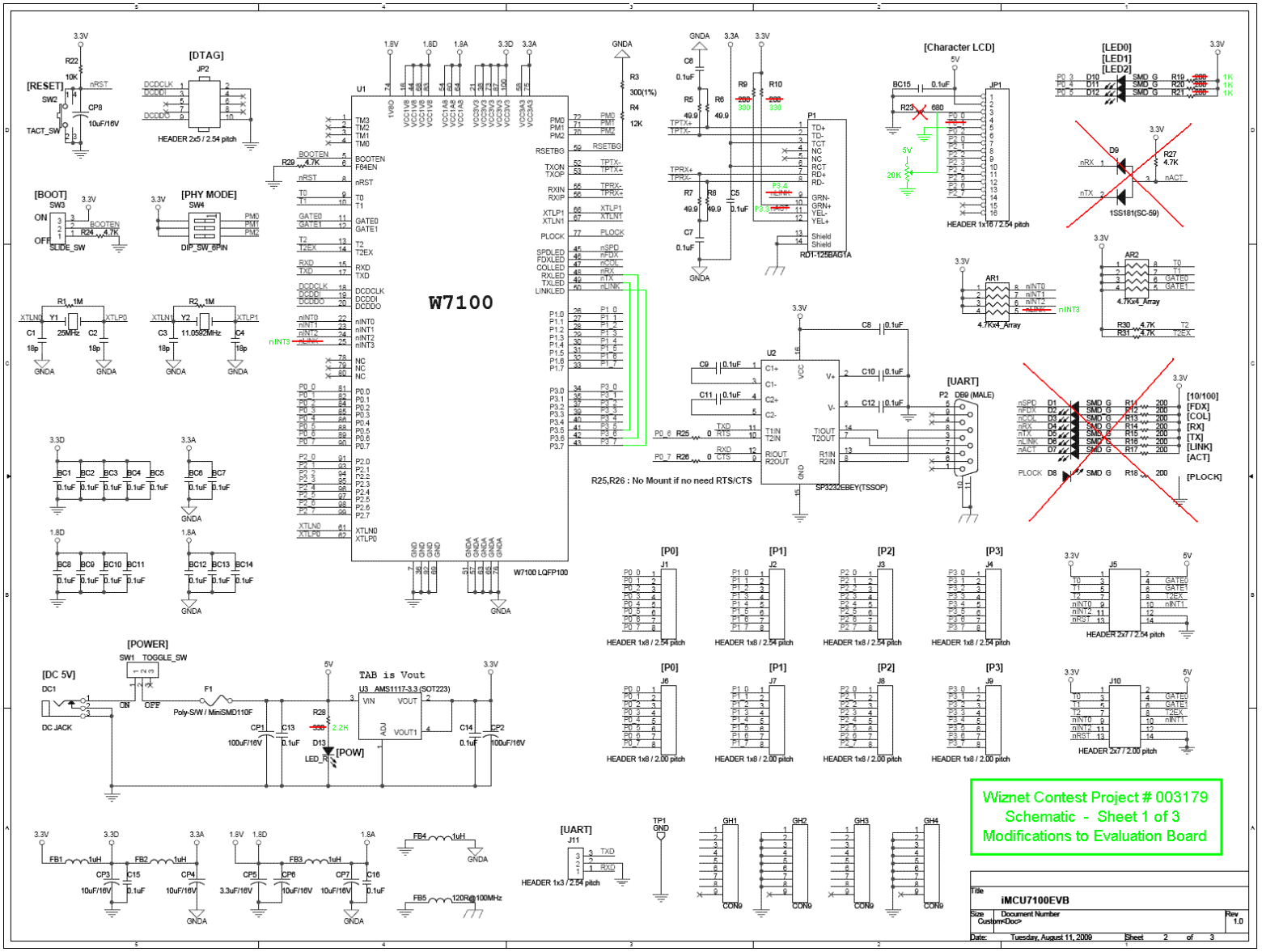
The Net Butler is controlled either from its LCD display menu system using two buttons for navigation, which allows control of all of its major functions, or from a set of web pages which provide access to the entire system.

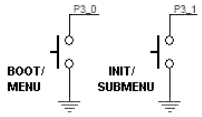

POWER
CONTROL
USB
DATA
MENU
SELECT
Net Butler Ready
NETWORK
SERIAL
5VDC
NET BUTLER

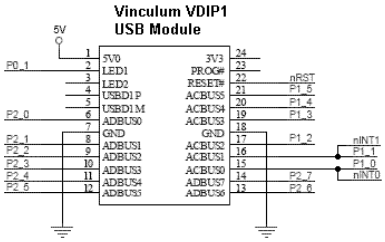


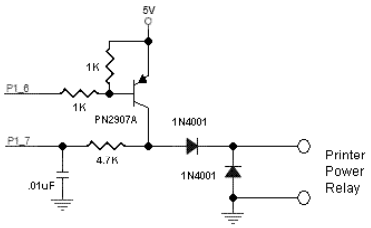


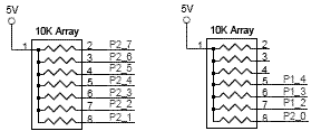

USB
DATA
SELECT
NETWORK
SERIAL
5VDC
NET BUTLER


W7100
W7100 LQFP100
[RESET]
[DTAG]
[BOOT]
[PHY MODE]
[Character LCD]
[LED0]
[LED1]
[LED2]
[UART]
[POWER]
[DC 5V]
[POW]
TAB is Vout
R25,R26 : No Mount if no need RTS/CTS
SP3232EBEY(TSSOP)
RDT-125BAGTA
1SS181(SC-59)
4.7Kx4_Array
HEADER 2x5 / 2.54 pitch
HEADER 1x16 / 2.54 pitch
HEADER 1x8 / 2.54 pitch
HEADER 2x7 / 2.54 pitch
HEADER 1x3 / 2.54 pitch
Wiznet Contest Project # 003179
Schematic - Sheet 1 of 3
Modifications to Evaluation Board
iMCU7100EVB
Tuesday, August 11, 2009

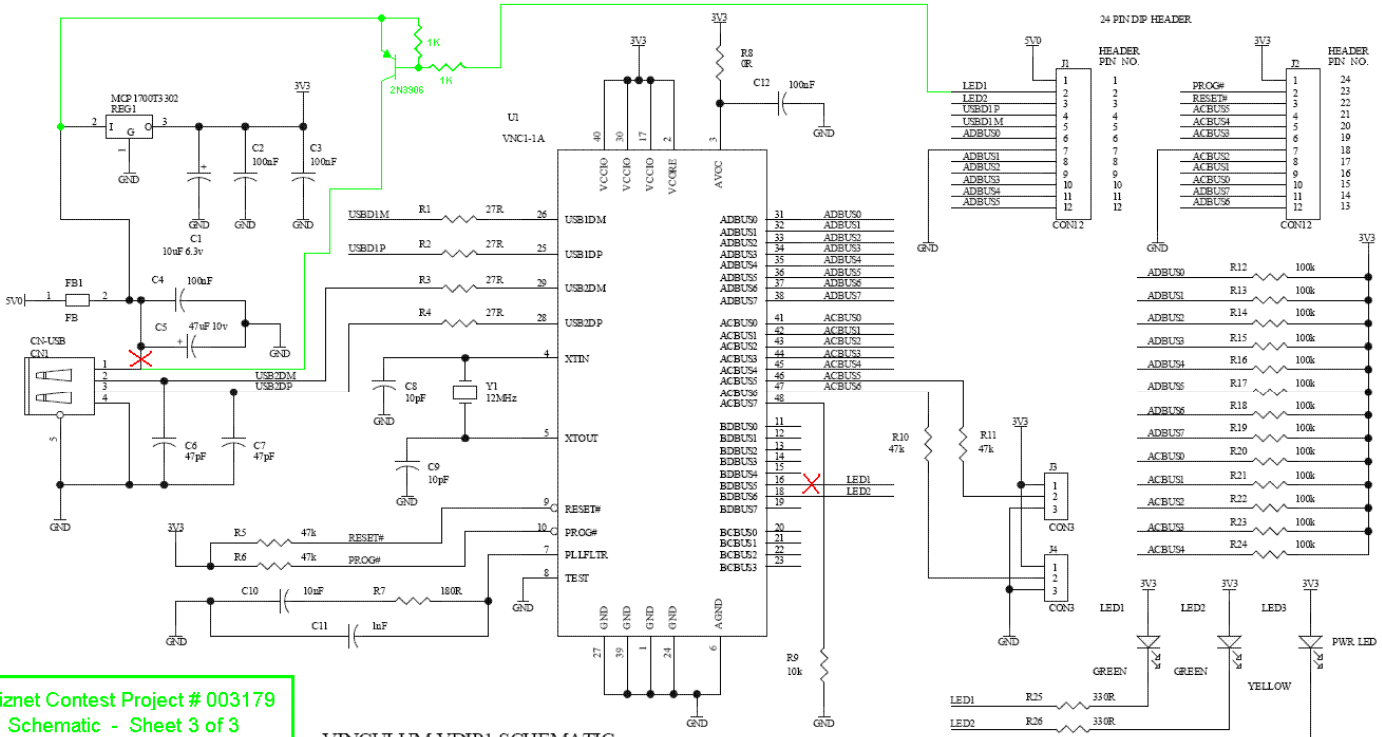






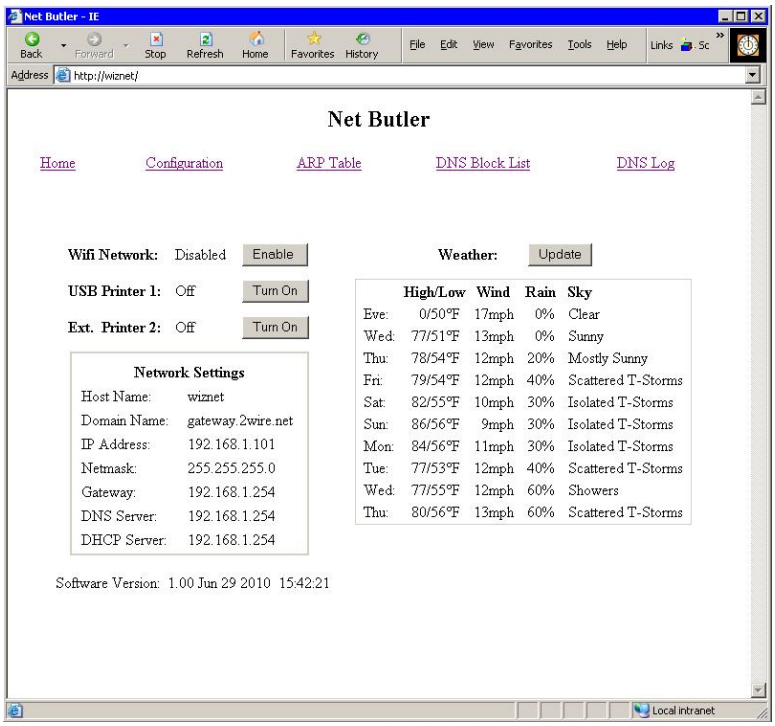


Wiznet Contest Project # 003179
Schematic - Sheet 2 of 3
Additions to Evaluation Board



Wiznet Contest Project # 003179
Schematic - Sheet 3 of 3
Modifications to VDIP1 Board

VINCULUM VDIP1 SCHEMATIC


Net Butler - IE
Back
Forward
Stop
Refresh
Home
Favorites
History
File
Edit
View
Favorites
Tools
Help
Links
Address
http://wiznet/
Net Butler
Home
Configuration
ARP Table
DNS Block List
DNS Log
Wifi Network: Disabled Enable
USB Printer 1: Off Turn On
Ext. Printer 2: Off Turn On
Network Settings
Host Name: wiznet
Domain Name: gateway.2wire.net
IP Address: 192.168.1.101
Netmask: 255.255.255.0
Gateway: 192.168.1.254
DNS Server: 192.168.1.254
DHCP Server: 192.168.1.254
Software Version: 1.00 Jun 29 2010 15:42:21
Weather: Update
High/Low Wind Rain Sky
Eve: 0/50°F 17mph 0% Clear
Wed: 77/51°F 13mph 0% Sunny
Thu: 78/54°F 12mph 20% Mostly Sunny
Fri: 79/54°F 12mph 40% Scattered T-Storms
Sat: 82/55°F 10mph 30% Isolated T-Storms
Sun: 86/56°F 9mph 30% Isolated T-Storms
Mon: 84/56°F 11mph 30% Isolated T-Storms
Tue: 77/53°F 12mph 40% Scattered T-Storms
Wed: 77/55°F 12mph 60% Showers
Thu: 80/56°F 13mph 60% Scattered T-Storms
Local intranet

```
      switch (httpc_state) {

            case HTTPC_STATE_IDLE:
                  break;

//          Parse the URL and send out a DNS lookup request

            case HTTPC_STATE_START:
                  if (httpc_parse_url ()) {
      // Parse URL in save_buf
                        if (dns_request (SOCKET_HTTP_C, url_host))
      // Start DNS lookup

                              set_state (DNS);
                  }
                  else {
                        set_state (IDLE);
                  }
                  break;

//          Wait for the DNS lookup result.

            case HTTPC_STATE_DNS:
                  dns_status = dns_result (host_ip);

                  if (dns_status == DNS_STATUS_DONE) {            //
DNS done
                        retry_count = 0;

                        set_state (REQUEST);
                  }
                  else if (dns_status != DNS_STATUS_BUSY) { // DNS
error
                        set_state (IDLE);
                  }
                  break;

//          Initialize the request variables.

            case HTTPC_STATE_REQUEST:
                  httpc_init_req ();

                  set_state (SOCKET);
                  break;

//          Open the socket in TCP mode.

            case HTTPC_STATE_SOCKET:
                  if (socket (SOCKET_HTTP_C, Sn_MR_TCP, 0, 0) == true)
{
```

```
                        d2_printf ("< Open >\n");

                        set_state (OPEN);
                  }
                  break;

//          Wait until the socket is open, then connect to the host.

            case HTTPC_STATE_OPEN:
                  if (sr == SOCK_INIT) {
                        d2_printf ("Opened\n");

                        connect (SOCKET_HTTP_C, host_ip, url_port);
                        d2_printf ("< Connect >\n");

                        set_state (CONNECT);
                  }
                  break;

//          Wait for the connection to be established, then send the
HTTP request.
//          If the socket is closed, go to Retry state.

            case HTTPC_STATE_CONNECT:
                  if (sr == SOCK_ESTABLISHED) {
                        d2_printf ("Connected\n");

                        httpc_send_req ();

                        set_state (RECV_HEADER);
                  }
                  else if (sr == SOCK_CLOSED) {
                        d1_printf ("Closed!!\n");

                        hdr_status = HDR_STATUS_ABORTED + httpc_state;
                        httpc_status = HTTPC_STATUS_RETRY;

                        set_state (CLOSE);
                  }
                  break;

//          Process the incoming data.

            case HTTPC_STATE_RECV_HEADER:
            case HTTPC_STATE_RECV_DATA:

                  httpc_receive_data ();

                  break;

//          Disconnect from the host.

            case HTTPC_STATE_DISCONNECT:
                  disconnect (SOCKET_HTTP_C);
                  d2_printf ("< Disconnect >\n");

                  set_state (CLOSE);
```

```
                  break;

//          Wait for the socket to close, discarding any additional
data that comes in.

            case HTTPC_STATE_CLOSE:
                  if (sr == SOCK_CLOSE_WAIT || sr == SOCK_FIN_WAIT) {

                        len = getSn_RX_RSR (SOCKET_HTTP_C);

                        if (len != 0) {
                              len = recv (SOCKET_HTTP_C, NULL, len);
                              d2_printf ("< Discard %u bytes >\n",
len);
                        }
                  }
                  else if (sr == SOCK_CLOSED) {

                        close (SOCKET_HTTP_C);
                        d2_printf ("< Close >\n");

                        set_state (DONE);
                  }
                  break;
```