Arduino Ethernet Shield and MEGA 2560 Web Server Controlling 24 LED Outputs with Checkboxes

22-28 minutes

Created on: 31 March 2015

This tutorial shows how to use the Arduino MEGA 2560 and Arduino Ethernet shield to make a web server that hosts a web page allowing 24 outputs to be controlled by clicking checkboxes on the web page.

Ajax is used on the web page and Arduino to control the LEDs and communicate between the web page and the Arduino via the Ethernet connection.

To demonstrate the outputs working, 24 LEDs with series resistors are connected to 24 pins of the Arduino MEGA. Pins 26 to 49 of the MEGA are used as outputs and connected to the LEDs.

The video below shows the web page and circuit working.

Can't see the video? View on YouTube →

Understanding the Code

Basic Code Structure

The code consists of an Arduino sketch that is loaded to the Arduino MEGA and an index.htm file that is copied to the micro SD card. The index.htm page contains HTML, CSS and JavaScript. The micro SD card is inserted into the micro SD card socket on the Ethernet shield.

The Arduino MEGA and Ethernet shield are programmed to behave as a web server that serves up the web page from the micro SD card when a web browser connects to the server and requests the page.

The JavaScript that is embedded in the page communicates with the Arduino web server using Ajax after the web page has been loaded to the web browser.

Prerequisites for Understanding the Code

The code for this tutorial is listed and explained below, but a basic understanding of the technologies used in this tutorial will be needed. All the basics of how an Arduino web server, web pages and Ajax work are explained in the Arduino Ethernet web server tutorial on this website.

How the Code Design was Started

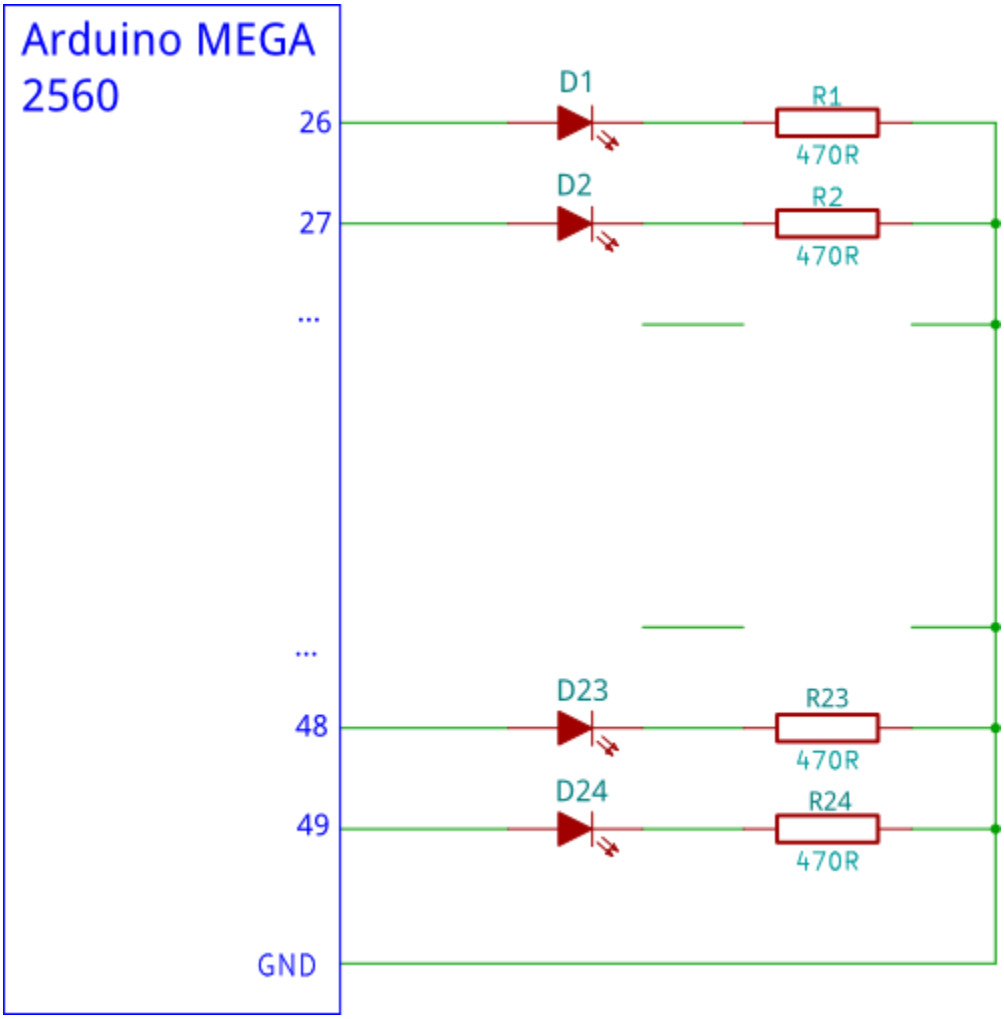
The code in this tutorial is based on part 16 of the web server tutorial – SD card web server I/O but is modified to use only outputs for the 24 outputs.

It would be a lot of work and also the HTML file would be bigger if this project accessed the outputs directly using unique names like the project that it is based on does. Because of this, the other major change to the base code was to make use of loops to access the 24 outputs sequentially in the Arduino code and JavaScript.

Hardware and Circuit Diagram

An Arduino MEGA 2560 and Arduino Ethernet shield are used in this project. A 2GB micro SD card is used to store the web page that is hosted by the Arduino.

The circuit diagram shows how the LEDs are connected to the Arduino. Only the first two and last two LEDs with series resistors are shown in the circuit as the rest of the LEDs are connected in the same way sequentially from pin 26 to pin 49 of the Arduino MEGA.



Arduino MEGA LED Connection Circuit Diagram

NOTE: Pins 50 to 53 (four pins) can not be used as outputs when using the Ethernet shield. This is because these are SPI port pins that are used to control the Ethernet chip and SD card.

Update and Correction 3 June 2015: In the original article, pin 53 was said to be the SS (Slave Select – of the SPI port) that controls the Ethernet chip (W5100). Although pin 53 on the MEGA is a SS pin from the SPI port, it is not the same SS used by the Ethernet shield.

To summarise: only pins 50 to 52 connect to the Ethernet shield through the ICSP header for SPI data and clock purposes – pin 53 is actually available. Pin 53 on the MEGA is called SS and pin 10 on the Ethernet shield is called SS but they are not connected.

Further to the above, the Arduino Ethernet shield web page has the following to say:

Arduino communicates with both the W5100 and SD card using the SPI bus (through the ICSP header). This is on digital pins 10, 11, 12, and 13 on the Uno and pins 50, 51, and 52 on the Mega. On both boards, pin 10 is used to select the W5100 and pin 4 for the SD card. These pins cannot be used for general I/O. On the Mega, the hardware SS pin, 53, is not used to select either the W5100 or the SD card, but it must be kept as an output or the SPI interface won't work.

Pin-Mega	Pin-Ethernet shield
4	4
10	10
11	50
12	51
13	52
Reset	Reset
IOREF	IOREF
AREF	AREF

Download the Arduino Code and HTML Page

Although the Arduino sketch and HTML page can be copied and pasted from the source listings below, it is recommended to download the files here because the downloaded HTML page will be slightly smaller in size as it uses tab characters instead of spaces for indenting the code.

Arduino sketch and index.htm page in one zipped file: eth_websrv_SD_Ajax_24_out.zip 4.4kB

Arduino Sketch Code

In the Arduino code shown below, change the MAC address in the source code to use the MAC address printed on the sticker on the bottom of your Ethernet shield in the following line of code:

```
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
```

Change the IP address in the code to suit your own network in the line:

```
IPAddress ip(192, 168, 0, 20);
/*-----------------------------------------------------------
  Program:      eth_websrv_SD_Ajax_24_out
```

```
  Description:  Arduino web server allows 24 outputs to be
                switched on and off using checkboxes.
                The web page is stored on the micro SD card.

  Hardware:     Arduino MEGA 2560 and official Arduino Ethernet
                shield.
                Tested with 24 LEDs connected from pin 26 to 49
                Can't use pins 50 to 53 when Ethernet shield
                plugged in, these pins are for SPI.
                2Gb micro SD card formatted FAT16.

  Software:     Developed using Arduino 1.0.6 software
                Should be compatible with Arduino 1.0 +
                SD card contains web page called index.htm

  References:   - WebServer example by David A. Mellis and
                  modified by Tom Igoe
                - SD card examples by David A. Mellis and
                  Tom Igoe
                - Ethernet library documentation:
                  http://arduino.cc/en/Reference/Ethernet
                - SD Card library documentation:
                  http://arduino.cc/en/Reference/SD
                - Based on code from the Ethernet shield
                  tutorial:
                  http://startingelectronics.org/tutorials/arduino/ethernet-shield-web-
server-tutorial/SD-card-IO/

  Date:         30 March 2015

  Author:       W.A. Smith, http://startingelectronics.org
------------------------------------------------------------*/

#include <SPI.h>
#include <Ethernet.h>
#include <SD.h>
// size of buffer used to capture HTTP requests
#define REQ_BUF_SZ   60

// MAC address from Ethernet shield sticker under board
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192, 168, 0, 20);    // IP address, may need to change depending on network
EthernetServer server(80);        // create a server at port 80
File webFile;                     // the web page file on the SD card
char HTTP_req[REQ_BUF_SZ] = {0};  // buffered HTTP request stored as null terminated
string
char req_index = 0;               // index into HTTP_req buffer
unsigned char LED_state[3] = {0}; // stores the states of the LEDs, 1 bit per LED

void setup()
{
    int i;

    // disable Ethernet chip
    pinMode(10, OUTPUT);
    digitalWrite(10, HIGH);

    Serial.begin(9600);       // for debugging

    // initialize SD card
    Serial.println("Initializing SD card...");
    if (!SD.begin(4)) {
        Serial.println("ERROR - SD card initialization failed!");
```

```
        return;    // init failed
    }
    Serial.println("SUCCESS - SD card initialized.");
    // check for index.htm file
    if (!SD.exists("index.htm")) {
        Serial.println("ERROR - Can't find index.htm file!");
        return;  // can't find index file
    }
    Serial.println("SUCCESS - Found index.htm file.");

    // pins 26 to 49 are outputs
    for (i = 26; i <= 49; i++) {
        pinMode(i, OUTPUT);    // set pins as outputs
        digitalWrite(i, LOW);  // switch the output pins off
    }

    Ethernet.begin(mac, ip);  // initialize Ethernet device
    server.begin();           // start to listen for clients
}

void loop()
{
    EthernetClient client = server.available();  // try to get client

    if (client) {  // got client?
        boolean currentLineIsBlank = true;
        while (client.connected()) {
            if (client.available()) {   // client data available to read
                char c = client.read(); // read 1 byte (character) from client
                // limit the size of the stored received HTTP request
                // buffer first part of HTTP request in HTTP_req array (string)
                // leave last element in array as 0 to null terminate string (REQ_BUF_SZ
- 1)
                if (req_index < (REQ_BUF_SZ - 1)) {
                    HTTP_req[req_index] = c;          // save HTTP request character
                    req_index++;
                }
                // last line of client request is blank and ends with \n
                // respond to client only after last line received
                if (c == '\n' && currentLineIsBlank) {
                    // send a standard http response header
                    client.println("HTTP/1.1 200 OK");
                    // remainder of header follows below, depending on if
                    // web page or XML page is requested
                    // Ajax request - send XML file
                    if (StrContains(HTTP_req, "ajax_inputs")) {
                        // send rest of HTTP header
                        client.println("Content-Type: text/xml");
                        client.println("Connection: keep-alive");
                        client.println();
                        SetLEDs();
                        // send XML file containing input states
                        XML_response(client);
                    }
                    else {  // web page request
                        // send rest of HTTP header
                        client.println("Content-Type: text/html");
                        client.println("Connection: keep-alive");
                        client.println();
                        // send web page
                        webFile = SD.open("index.htm");        // open web page file
                        if (webFile) {
                            while(webFile.available()) {
```

```
                                client.write(webFile.read()); // send web page to client
                            }
                            webFile.close();
                        }
                    }
                    // display received HTTP request on serial port
                    //Serial.print(HTTP_req);
                    // reset buffer index and all buffer elements to 0
                    req_index = 0;
                    StrClear(HTTP_req, REQ_BUF_SZ);
                    break;
                }
                // every line of text received from the client ends with \r\n
                if (c == '\n') {
                    // last character on line of received text
                    // starting new line with next character read
                    currentLineIsBlank = true;
                }
                else if (c != '\r') {
                    // a text character was received from client
                    currentLineIsBlank = false;
                }
            } // end if (client.available())
        } // end while (client.connected())
        delay(1);      // give the web browser time to receive the data
        client.stop(); // close the connection
    } // end if (client)
}

// checks if received HTTP request is switching on/off LEDs
// also saves the state of the LEDs
void SetLEDs(void)
{
    char str_on[12] = {0};
    char str_off[12] = {0};
    unsigned char i;
    unsigned int  j;
    int LED_num = 1;

    for (i = 0; i < 3; i++) {
        for (j = 1; j <= 0x80; j <<= 1) {
            sprintf(str_on,  "LED%d=%d", LED_num, 1);
            sprintf(str_off, "LED%d=%d", LED_num, 0);
            if (StrContains(HTTP_req, str_on)) {
                LED_state[i] |= (unsigned char)j;  // save LED state
                digitalWrite(LED_num + 25, HIGH);
                Serial.println("ON");
            }
            else if (StrContains(HTTP_req, str_off)) {
                LED_state[i] &= (unsigned char)(~j);  // save LED state
                digitalWrite(LED_num + 25, LOW);
            }
            LED_num++;
        }
    }
}

// send the XML file with analog values, switch status
//  and LED status
void XML_response(EthernetClient cl)
{
    unsigned char i;
    unsigned int  j;
```

```
    cl.print("<?xml version = \"1.0\" ?>");
    cl.print("<inputs>");
    for (i = 0; i < 3; i++) {
        for (j = 1; j <= 0x80; j <<= 1) {
            cl.print("<LED>");
            if ((unsigned char)LED_state[i] & j) {
                cl.print("checked");
                //Serial.println("ON");
            }
            else {
                cl.print("unchecked");
            }
            cl.println("</LED>");
        }
    }
    cl.print("</inputs>");
}

// sets every element of str to 0 (clears array)
void StrClear(char *str, char length)
{
    for (int i = 0; i < length; i++) {
        str[i] = 0;
    }
}

// searches for the string sfind in the string str
// returns 1 if string found
// returns 0 if string not found
char StrContains(char *str, char *sfind)
{
    char found = 0;
    char index = 0;
    char len;

    len = strlen(str);

    if (strlen(sfind) > len) {
        return 0;
    }
    while (index < len) {
        if (str[index] == sfind[found]) {
            found++;
            if (strlen(sfind) == found) {
                return 1;
            }
        }
        else {
            found = 0;
        }
        index++;
    }

    return 0;
}
```

Setup

The **setup()** function initializes the SD card, then sets the Arduino pins 26 to 49 as outputs and switches them low (or off).

The Ethernet is then initialized before the main loop is run.

The Main Loop

The main loop in the sketch handles the browser requests. It responds to two types of requests – a request for the web page and an Ajax request.

Page Request

When a web browser is used to connect to the web server at it's IP address, the Arduino web server responds with the **index.htm** page which then gets loaded to the browser.

Once the web page is loaded to the browser, the JavaScript in the web page will start to run and will do an Ajax request every second.

Ajax Request

The JavaScript on the web page will send the state of any of the checkboxes to the Arduino with each Ajax request.

The Arduino responds to the Ajax request with an XML file that contains all the states of the LEDs / checkboxes. This is so that if a second browser connects to the web server, the correct states of the checkboxes (checked or unchecked) will be set on the page.

Each time the an Ajax request is received by the Arduino, the **SetLEDs()** function is run which sees which checkboxes have been checked or unchecked and then sets or clears the corresponding LED.

The state of each LED is stored as a bit in the **LED_state** array which consists of 3 bytes of 8 bits each = 24 bits corresponding to the 24 outputs. Whenever the state of an LED is set or cleared, the corresponding bit is set in the array to keep track of the LED states.

Because of this array, each time that the array is accessed, it is accessed in a **for** loop as 3 bytes and a second **for** loop accesses each bit in the current byte using bitwise operators.

After **SetLEDs()** is called in response to the received Ajax request, **XML_response()** is called to send back the XML file containing the LED states.

HTML, CSS and JavaScript

The HTML, CSS and JavaScript for the project are all embedded in a single file called **index.htm** and no external files are used.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Arduino MEGA 24 Output</title>
        <script>
        strLED = "";

        function GetArduinoIO()
        {
            nocache = "&nocache=" + Math.random() * 1000000;
            var request = new XMLHttpRequest();
```

```
            request.onreadystatechange = function()
            {
                if (this.readyState == 4) {
                    if (this.status == 200) {
                        if (this.responseXML != null) {
                            // XML file received - contains LED states
                            var re;
                            var num_LEDs;
                            var i;
                            var ledstr = "";

                            re = this.responseXML.getElementsByTagName('LED');
                            num_LEDs = re.length;

                            for (i = 0; i < num_LEDs; i++) {
                                ledstr = "LED" + (i + 1);
                                if (re[i].childNodes[0].nodeValue === "checked") {
                                    document.getElementsByName(ledstr)[0].checked = true;
                                }
                                else {
                                    document.getElementsByName(ledstr)[0].checked =
false;
                                }
                            }
                        }
                    }
                }
                // send HTTP GET request with LEDs to switch on/off if any
                request.open("GET", "ajax_inputs" + strLED + nocache, true);
                request.send(null);
                setTimeout('GetArduinoIO()', 1000);
                strLED = "";
            }
            // service LEDs when checkbox checked/unchecked
            function GetCheck(led_num_str, cb)
            {
                if (cb.checked) {
                    strLED += ("&" + led_num_str + "=1");
                }
                else {
                    strLED += ("&" + led_num_str + "=0");
                }
            }
        </script>
        <style>
            .out_box {
                float: left;
                margin: 0 20px 20px 0;
                border: 1px solid blue;
                padding: 0 5px 0 5px;
                min-width: 280px;
            }
            input {
                margin: 10px;
            }
            input {
                vertical-align: -3px;
            }
            h1 {
                font-size: 120%;
                color: blue;
                margin: 0 0 10px 0;
```

```
        }
        h2 {
            font-size: 85%;
            color: #5734E6;
            margin: 5px 0 5px 0;
        }
        p, form, button {
            font-size: 80%;
            color: #252525;
        }
        .small_text {
            font-size: 70%;
            color: #737373;
        }
    </style>
    </head>
    <body onload="GetArduinoIO()">
        <h1>Arduino MEGA 24 Output</h1>
        <div class="out_box">
            <form class="check_LEDs" name="LED_form1">
                <input type="checkbox" name="LED1" value="0" onclick="GetCheck('LED1',
this)" />LED 1 (D26)
                <input type="checkbox" name="LED2" value="0" onclick="GetCheck('LED2',
this)" />LED 2 (D27)<br />
                <input type="checkbox" name="LED3" value="0" onclick="GetCheck('LED3',
this)" />LED 3 (D28)
                <input type="checkbox" name="LED4" value="0" onclick="GetCheck('LED4',
this)" />LED 4 (D29)
            </form>
        </div>
        <div class="out_box">
            <form class="check_LEDs" name="LED_form2">
                <input type="checkbox" name="LED5" value="0" onclick="GetCheck('LED5',
this)" />LED 5 (D30)
                <input type="checkbox" name="LED6" value="0" onclick="GetCheck('LED6',
this)" />LED 6 (D31)<br />
                <input type="checkbox" name="LED7" value="0" onclick="GetCheck('LED7',
this)" />LED 7 (D32)
                <input type="checkbox" name="LED8" value="0" onclick="GetCheck('LED8',
this)" />LED 8 (D33)
            </form>
        </div>
        <div class="out_box">
            <form class="check_LEDs" name="LED_form3">
                <input type="checkbox" name="LED9" value="0" onclick="GetCheck('LED9',
this)" />LED 9 (D34)
                <input type="checkbox" name="LED10" value="0" onclick="GetCheck('LED10',
this)" />LED 10 (D35)<br />
                <input type="checkbox" name="LED11" value="0" onclick="GetCheck('LED11',
this)" />LED 11 (D36)
                <input type="checkbox" name="LED12" value="0" onclick="GetCheck('LED12',
this)" />LED 12 (D37)
            </form>
        </div>
        <div class="out_box">
            <form class="check_LEDs" name="LED_form4">
                <input type="checkbox" name="LED13" value="0" onclick="GetCheck('LED13',
this)" />LED 13 (D38)
                <input type="checkbox" name="LED14" value="0" onclick="GetCheck('LED14',
this)" />LED 14 (D39)<br />
                <input type="checkbox" name="LED15" value="0" onclick="GetCheck('LED15',
this)" />LED 15 (D40)
```

```
                <input type="checkbox" name="LED16" value="0" onclick="GetCheck('LED16',
this)" />LED 16 (D41)
            </form>
        </div>
        <div class="out_box">
            <form class="check_LEDs" name="LED_form5">
                <input type="checkbox" name="LED17" value="0" onclick="GetCheck('LED17',
this)" />LED 17 (D42)
                <input type="checkbox" name="LED18" value="0" onclick="GetCheck('LED18',
this)" />LED 18 (D43)<br />
                <input type="checkbox" name="LED19" value="0" onclick="GetCheck('LED19',
this)" />LED 19 (D44)
                <input type="checkbox" name="LED20" value="0" onclick="GetCheck('LED20',
this)" />LED 20 (D45)
            </form>
        </div>
        <div class="out_box">
            <form class="check_LEDs" name="LED_form6">
                <input type="checkbox" name="LED21" value="0" onclick="GetCheck('LED21',
this)" />LED 21 (D46)
                <input type="checkbox" name="LED22" value="0" onclick="GetCheck('LED22',
this)" />LED 22 (D47)<br />
                <input type="checkbox" name="LED23" value="0" onclick="GetCheck('LED23',
this)" />LED 23 (D48)
                <input type="checkbox" name="LED24" value="0" onclick="GetCheck('LED24',
this)" />LED 24 (D49)
            </form>
        </div>
    </body>
</html>
```

Setting and Clearing an LED

Whenever a checkbox is clicked, the **GetCheck()** function is called which is passed the LED name and object reference so that the state of the checkbox can be determined in the function.

The **GetCheck()** function adds the LED name and state to a string which will be sent to the Arduino with the Ajax request. When the Arduino receives the Ajax request, it will set or clear the corresponding LED and keep a record of the state of all the LEDs.

Ajax Request and Response

The **GetArduinoIO()** function is called every second to check for an Ajax response and send the next Ajax request.

If an Ajax response is received, the code between the braces of the following **if** statements is run.

```
if (this.readyState == 4) {
        if (this.status == 200) {
                if (this.responseXML != null) {
```

If all the **if** statements evaluate to true, then the code in the body of the last **if** statement goes through each state of each LED that is sent back in the XML file and then checks or unchecks the corresponding checkbox. This may seem unnecessary because when a checkbox is clicked in a web page, it is automatically marked with a tick.

The reason for setting the states of the checkboxes / LEDs sent back from the Arduino is so that the states of the checkboxes / LEDs is stored in one central place – the web server. This has two advantages, firstly when additional web browsers connect to the web server, they will start to do Ajax requests after getting the initial web page. The states of all of the checkboxes / LEDs will then be updated in the web page after the first Ajax request / response. Secondly if the browser is closed and then opened again and connected to the Arduino web server, the Arduino remembers the states of all the LEDs and corresponding checkboxes. The checkboxes are then set to the correct states after the first Ajax request / response. The same applies if the web page is refreshed.

In the JavaScript code, a **for** loop is used to go through each of the LED state fields and set the checkboxes to either checked or unchecked.