TABLE OF CONTENTS

TESTING OF TOAST

Rather than using a simple timer circuit to control the toasting, the Smart-Toaster views the bread being toasted to gauge how done it is. The Smart-Toaster waits until the bread looks burnt/cooked/toasted enough to match which level the user selected. To measure how toasted the bread is, Smart-Toaster samples with a colour sensor. The colour sensor used is the Parallax ColorPAL 24bit RGB colour sensor (Figure xx, below).

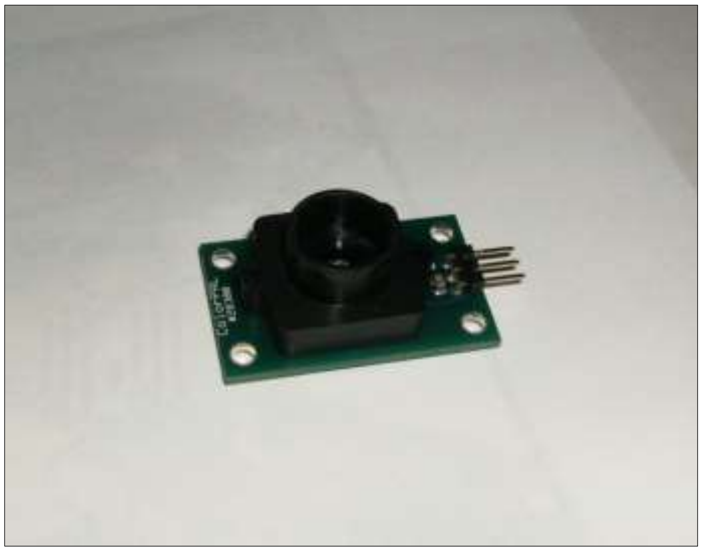


Figure 1

The ColorPAL has a built in microcontroller which communicates over a single line serial connection. Commands are sent to the ColorPAL over this serial line to tell the built in micro to sample black/white, sample full colour, output a colour, and et cetra. For the function of the getting the toast's darkness, the command code "m" is sent. As seen in the ColorPAL's datasheet (PARALLAX, 2009), the 'm' command tells the colorPAL to measure ambient light, do individual measurements for each of RGB (red, green, blue) and then output a string of three 3-digit hex numbers to represent each of the three RGB values.

The communication with the ColorPAL is done with Arduino's Software-Serial I/O at 4800 baud using an open-drain line. The exact operation of the serial communication to the ColorPAL required some modifications to the line to work with the Arduino's software serial. As see in Figure xxx below, the 2 pins (Tx, Rx) for the Arduino's serial are connected to the ColorPAL's single line by means of a diode to the Tx pin.

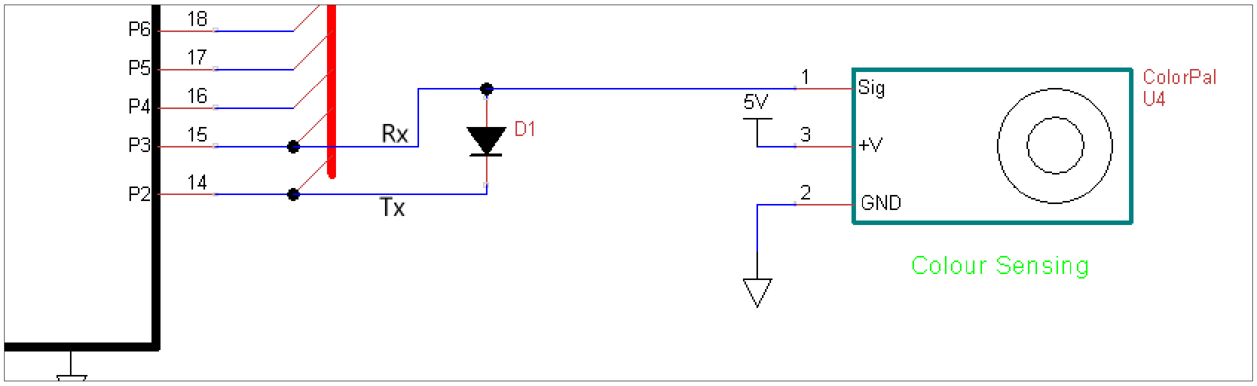


Figure 2

The reading of the colour data is done with a getColour() function within the Smart-Toaster's code.
The function sends the required command to the ColorPAL to read RGB colour data and then reads the incoming data, filling an array for RGB values.

In order to test if the toast is sufficiently cooked, the Smart-Toaster takes the colour values obtained with a getColour() sample and compares them to a predetermined set of ranges. Each of the three selectable toasting ranges has its own set of pre-determined colour ranges for the RGB values. The function testToast takes the RGB array values and compares them to whichever set of ranges was selected, if they are within range then it is treated as the toast being ready.

The values for the ranges of toast, (Light, Medium, & Dark) were measured by placing pieces of bread already toasted to the degree that each level represents into the toaster and taking a plethora of measurements. The observed ranges for each level and colour were recorded and hard-coded into the Smart-Toaster. Table xxx shows the ranges used for comparison. Examples of the degree of toasting for each level are shown below in Table 1.

Table 1

	Red Low	Red High	Green Low	Green High	Blue Low	Blue High
Level						
Light	0x40	0x44	0x30	0x36	0x48	0x51
Medium	0x42	0x47	0x30	0x36	0x46	0x54
Dark	0x34	0x39	0x30	0x37	0x46	0x50

The values in the above table are in unsigned hexadecimal. Testing was done with the values and tweaked until operation was optimal. The Smart-Toaster would reliably detect toast that matched the description of each level. Examples of the degree of toasting for each level are shown below in Figure xxx.

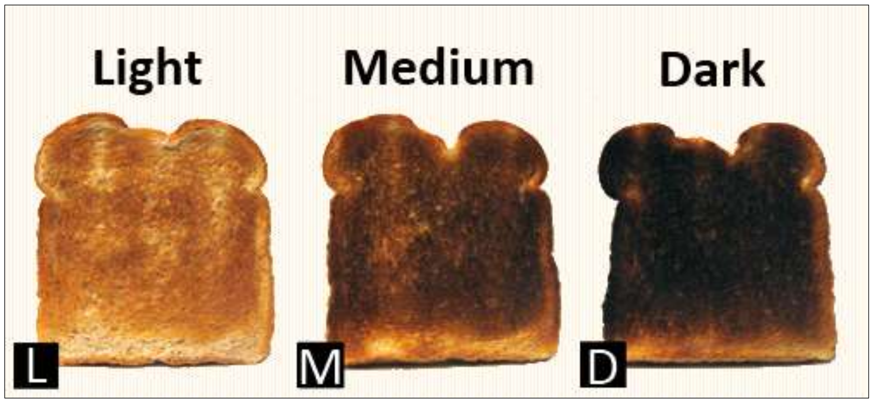


Figure 3

As one may observe, the colouration of the toast is not smooth and consistent for the whole piece. Variations in darkness are normal, so using a very small area to test for level of toasting would be less accurate than averaging a larger area. The colour sensor in the Smart-Toaster is situated in a way that it observes an estimated 4 cm^2. Adjusting positioning and optics could result in greater observed area and results but little time was spent to maximize along those lines.

TOASTER CONTROL

The Smart-Toaster's design demands requires a remote method of pushing down and pulling up the plunger that controls it. Implementation of an electro-mechanical system for actuating the toaster's plunger was needed. After a failed attempt to use a pull-type solenoid, a design revolving around a low voltage, low current stepper motor was developed.

The Smart-Toaster's microcontroller interfaces to the toaster element via a stepper motor and a chain. The stepper motor is driven with an L298N based driver board and controlled by the Arduinio microcontroller. The motor is wired up according to the schematic Figure xxx.

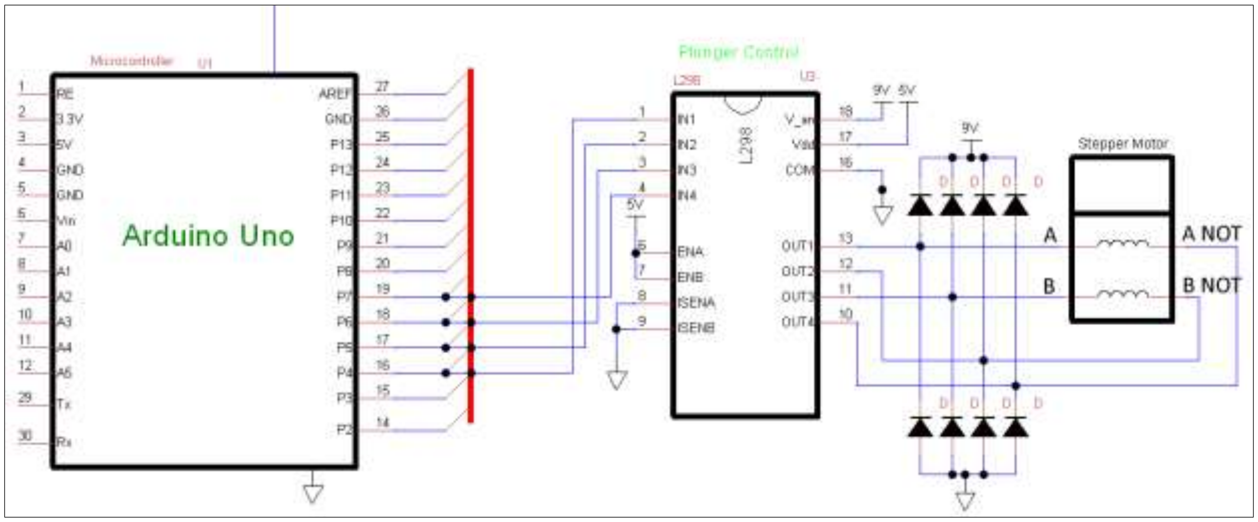


Figure 4

Pulsing the Arduino's output pins 4 → 7, which are connected to the L298N motor driver's INx inputs will step the motor forward a predictable amount each pulse. Each one of these pulses to the L298 (and therefor the stepper motor) is referred to as a *step*. By moving a set number of steps forward or back, the chain attached to the stepper motor will move a set distance (see Figure xxx).

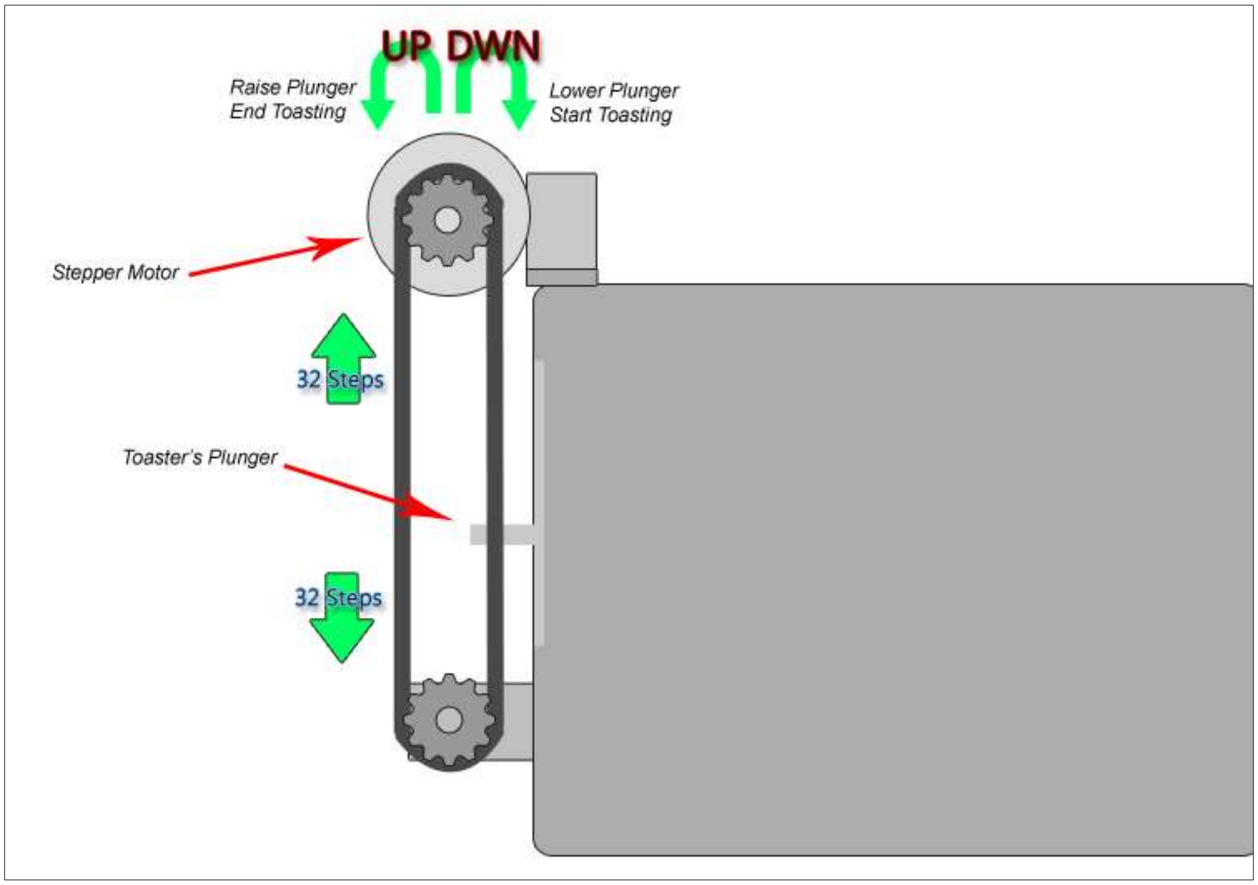


Figure 5

As shown, the Smart-Toaster is programmed to move the chain up/down 32 steps to raise and lower the toaster's plunger (which controls its heater) enough to turn on/off. Software wise, the stepper motor is controlled with a STEPMOT() function. A simple input of direction when called will make the motor+chain+plunger go up/down the specified amount. The function is ran to activate the toaster when starting the toasting process, and is ran again when the toast is detected to be in range of the level specified.

ARDUINO UNO

The Arduino Uno works as the center processing of the unit. It receives the alarm time data from the Ethernet Shield and color sensor data. The actions whether waiting, toasting and lift up the slot are dependent on the program flowchart and the data it receives.

HARDWARE CONFIGURATIONS

The Arduino uses pin2 and pin3 for the receiving color sensor data. In addition, the pin 4- 7 generates pulse to control the motor. The Analog pin5 reads the thermistor voltage to detecting temperature for safety reason. The port can be set by code " **pinMode (pin#, INPUT/OUTPUT)** " statement.

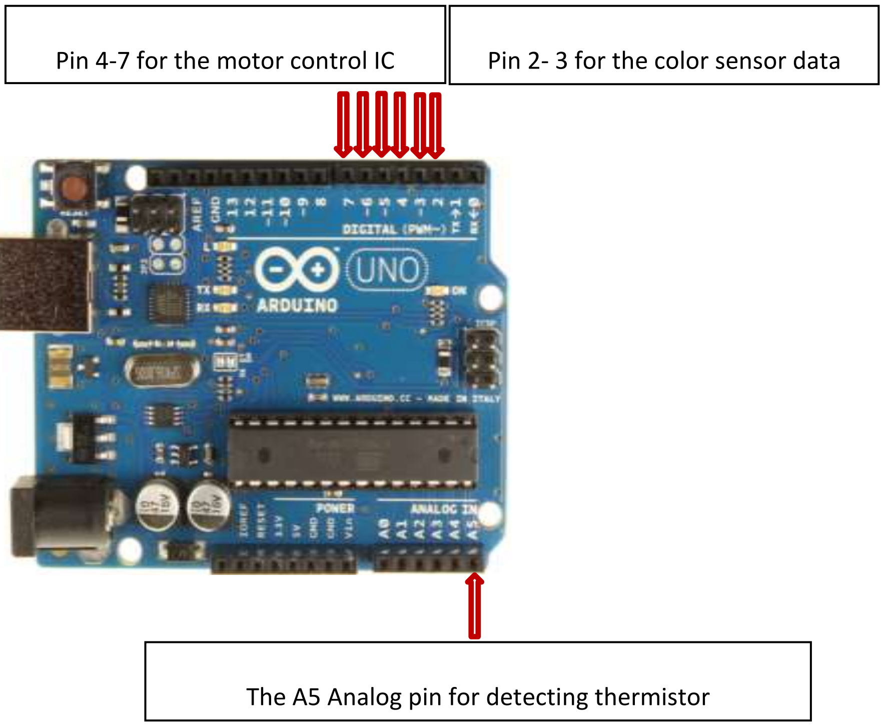

Pin 4-7 for the motor control IC
Pin 2- 3 for the color sensor data
The A5 Analog pin for detecting thermistor

Figure 6

CONTROL PROCESS

Basically, it gets the "waiting time" and the "doneness" data from the web-page. It passes the "waiting time "signals for waiting. And "doneness" signal to color sensor portion. Afterward, it waits until the time set. Then it calls the STEPMOT to push down the slot and start toasting. When it gets the "done signal" from color sensor The STEPMOT will be called to pop up the bread.

INTERRUPT

MsTimer2::set (period , INTupdate);
First parameter is the period (how often the interrupt)in millisecond unit, and the second parameter is the address of the interrupt routine.

Therefore the interrupt routine **(INTupdate)** can do the jobs: running the clock and detecting the temperature.

STRINGSETUPALARM

StringSetupAlarm will get the string of time value from WEBPAGE. Then the alarm time data in the SMART module will be determined by the string it passed. For example, if String HOUR = “01”, String MIN = “02”, String SEC = “03” is passed to the function, then the toasting will start after 1 hour, 2 minutes, and 3 seconds.

CHECKDONE

After the alarm time is set, the CHECKDONE function will be the “bell” of the alarm. The function checks the running clock time data on the smart module are equal to the alarm time or not. It returns one during waiting. It return zero for escaping the while loop when the time equals to the alarm time. On the other words, it determines if the smart toaster to wait or to push down the slot and start toasting.

STEPMOT

In the STEPMOT function, the Arduino control the step motor so that the motor pushes down the slot and turn on the toaster or vice versa. Physically, Pins P4 –P7 generate pulses to the motor control IC. In addition, the Boolean parameter determines the direction of the stepper motor:
1 = push down the slot and 0 = lift up the slot.

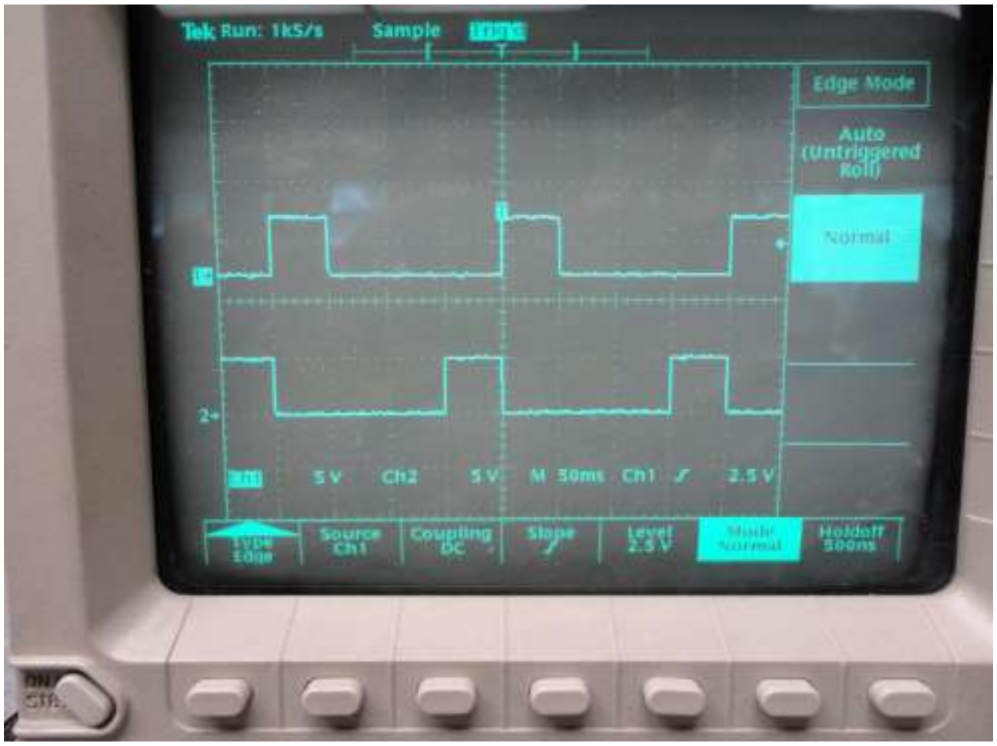


Figure 7 the pulses come from pin 3 and 4 of Arduino

SMART TOASTER SERVER OVERVIEW

The smart toaster server acts as a web server receiving data from clients and operating Arduino Uno with given data as well as it transmits data to the client as they want to see a status of toasting.

The smart toaster server is a tiny web server designed and implemented only for this project. Thus, this server only can handle certain data and requests. The smart toaster server uses Arduino Ethernet shield as HTTP adapter in which Arduino Uno (main micro controller) and client communicate with each other.

HARDWARE CONFIGURATIONS

The Smart Toaster consists of several components such as Arduino Ethernet Shield, router, RJ 45, and Arduino Uno.

Arduino Ethernet Shield

The Arduino Ethernet Shield allows Arduino Uno board to the internet. Inside of the Ethernet Shield, the Ethernet chip, W5100, provides a network (IP) stack capable of both TCP and UDP. In order to connect to the internet, RJ45 receiver is attached to the Ethernet Shield so that it can be adaptable of any type of routers.

The Arduino Ethernet Shield is simply attached onto the Arduino Uno board and powered over the main board through USB attached on the main board. Between the main board and the Ethernet Shield, SPI (Serial Peripheral Interface Bus) enables them to communicate. Once the HTTP packets reach to the Ethernet Shield, it attempts to interpret the packet following OSI 7 Layer. If the packets are valid, it transfers to the Arduino Uno via SPI bus.

Arduino Uno

The Arduino Uno is a micro-controller board based on ATmega328 [see Appendix D, ***ATMEGA328***]. Once the HTTP packets are transferred from the Ethernet Shield, the program in the Arduino Uno attempts to control other processes such as timer, operating motor and sensor.

Router

In this project, D-Link router is used to set up a private network since the network in BCIT doesn't allow internal access from one computer to another. To simulate the internet environment, the private network is required for this project.

LAN cables

LAN cables are connected between the Arduino Ethernet and the router. In addition, to simulate client-server environment, a laptop is connected together via RJ45 connector [Appendix D, ***RJ45***].

Installation

This section shows how to set up the environment for the Smart toaster server.

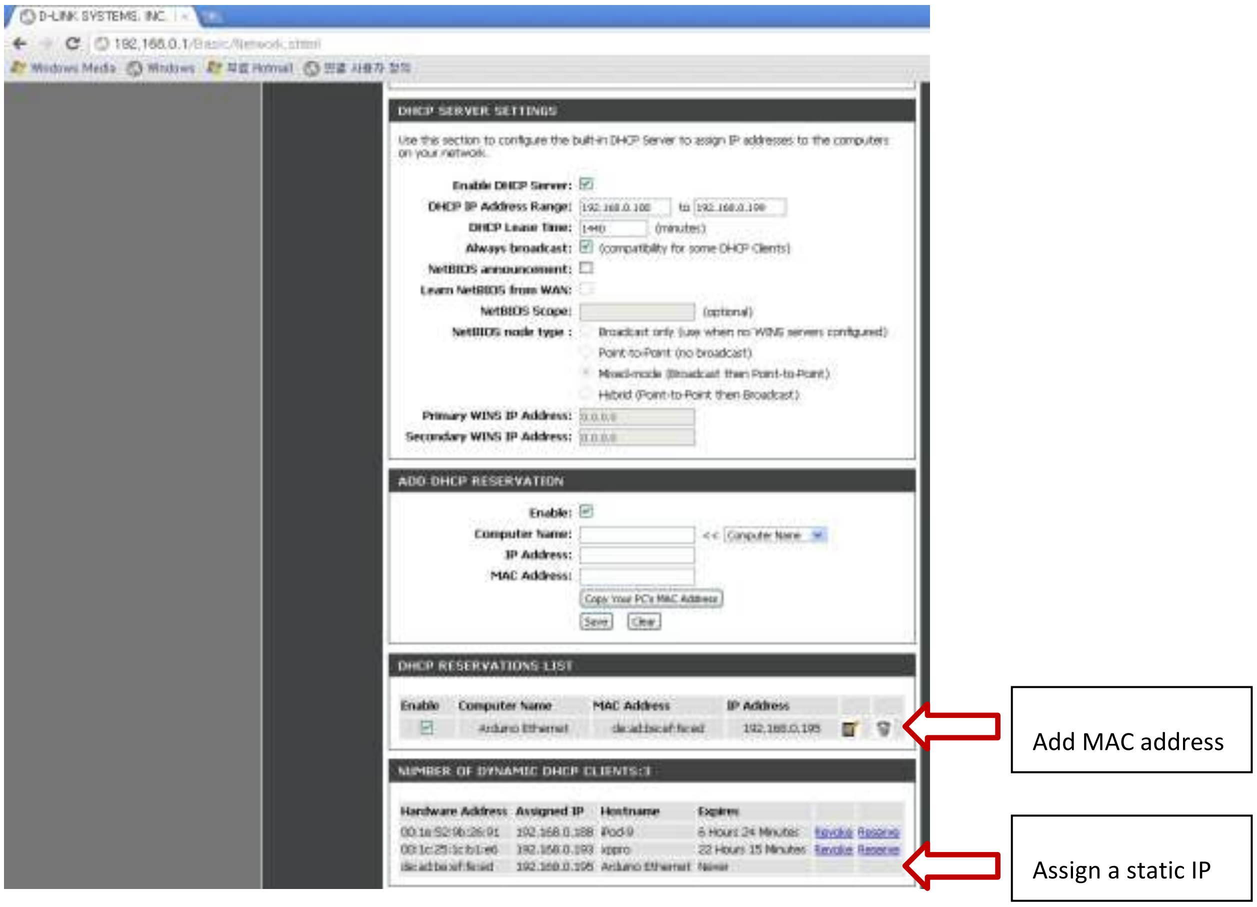
1. Arduino Ethernet Shield

   The Arduino Ethernet Shield simply mounts onto the Arduino Uno. The SPI bus connects together so that the Ethernet Shield and main board are able to communicate with each other.
   The Ethernet Shield is powered from the main board and the main board is connected via a standard USB cable from a computer.

2. Router

   For this project, D-Link wireless router is used to set up a local network. In general, routers recognize computers and devices based on their MAC addresses. However, unlike the instruction from the Arduino web site, the router didn't recognize the MAC address printed on top of the Arduino Ethernet.
   Instead of using the MAC address provided on the top of the Arduino Ethernet, use "0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED".

   In addition, the Arduino Ethernet needs a static IP address, so type the MAC address manually and assign a static IP among the available IPs [Figure 8 router setting].



D-LINK SYSTEMS, INC.

192.168.0.1/Basic/Network.shtml

DHCP SERVER SETTINGS

Use this section to configure the built-in DHCP Server to assign IP addresses to the computers on your network.

Enable DHCP Server:	✓
DHCP IP Address Range:	192.168.0.100 to 192.168.0.199
DHCP Lease Time:	1440 (minutes)
Always broadcast:	✓ (compatibility for some DHCP Clients)
NetBIOS announcement:	☐
Learn NetBIOS from WAN:	☐
NetBIOS Scope:	(optional)
NetBIOS node type :	○ Broadcast only (use when no WINS servers configured)
	○ Point-to-Point (no broadcast)
	● Mixed-mode (Broadcast then Point-to-Point)
	○ Hybrid (Point-to-Point then Broadcast)
Primary WINS IP Address:	0.0.0.0
Secondary WINS IP Address:	0.0.0.0

ADD DHCP RESERVATION

Enable:	✓
Computer Name:	<< Computer Name
IP Address:	
MAC Address:	

Copy Your PC's MAC Address

Save Clear

DHCP RESERVATIONS LIST

Enable	Computer Name	MAC Address	IP Address		
✓	Arduino Ethernet	de:ad:be:ef:fe:ed	192.168.0.195		

NUMBER OF DYNAMIC DHCP CLIENTS:3

Hardware Address	Assigned IP	Hostname	Expires		
00:1e:52:9b:26:91	192.168.0.188	iPod-9	6 Hours 24 Minutes	Revoke	Reserve
00:1c:25:1c:b1:e6	192.168.0.193	xppro	22 Hours 15 Minutes	Revoke	Reserve
de:ad:be:ef:fe:ed	192.168.0.195	Arduino Ethernet	Never		

Figure 8 router setting

PROTOCOL

In order to communicate with a serve and clients, the protocol is required. HTTP protocol is used for this project so that the clients can request and get a response from the Smart toaster server.

HTTP

HTTP protocol, which is based one TCP/IP [Appendix D, ***TCP/IP***] communication protocol is commonly used in web based client-server computing system. Clients using a web browser open a connection and send a request message to an HTTP server, which is the Smart toaster server; the server then returns a response message, usually containing the resource that was requested. After delivering the response, the server closes the connection.

HTTP Structures

HTTP message consists of 3 parts in general such as initial line, header, and body. The structure of the message may be differed from HTTP GET method [Appendix D, ***HTTP GET METHOD***] and HTTP POST method [Appendix D, ***HTTP POST METHOD***]. In addition, the contents of the message will be differed from the request and response.

The Smart toaster server only handles HTTP GET method, which means as the clients request to the Smart toaster server after choosing the start time and doneness, the start time for toasting and doneness data are attached to the request URL (E.g. http://192.168.0.195/setting?hr=01&m=03&sc=20&d=M). As seen in the example, “192.168.0.195” is the Smart toaster server IP, and the parameters are followed by the name of the page, setting.

Initial Line

The initial line is placed ahead of the Header and body in HTTP protocol. It will be changed based on the Request and Response message. [Figure 2] shows the sequence of the request and response between a client and Smart toaster server.

1. Request
   - As the users type “the server IP/setting”, the web browser places the URL into the initial line and adds HTTP/1.1 at the end of the initial line.
2. Response
   - As the Smart toaster server accepts the request from the client successfully, then the server puts “HTTP/1.1 200 OK” into the initial line and sends it with HTML.

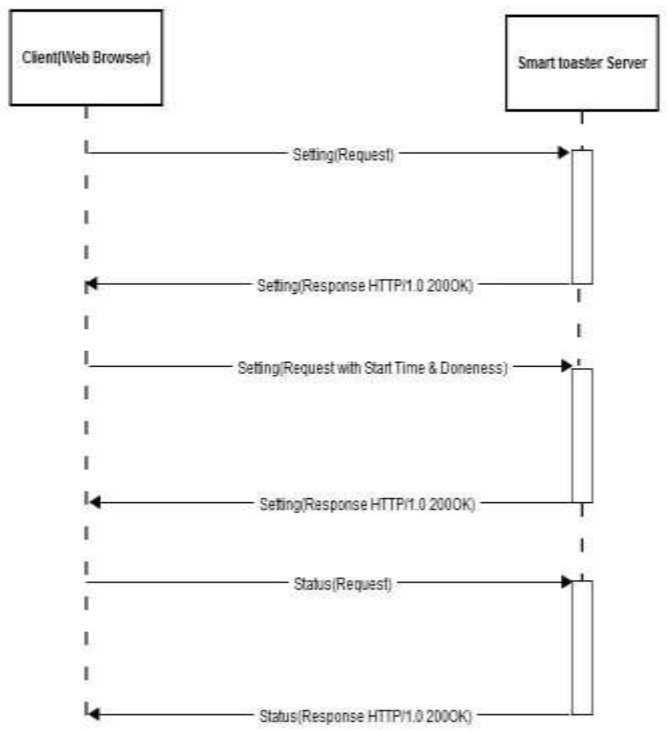

Client(Web Browser)
Smart toaster Server
Setting(Request)
Setting(Response HTTP/1.0 200OK)
Setting(Request with Start Time & Doneness)
Setting(Response HTTP/1.0 200OK)
Status(Request)
Status(Response HTTP/1.0 200OK)

Figure 9 HTTP request/response

Header

Header line contains the information about the request and response or about the object sent in the message body. In this project, the Smart toaster server only uses HTTP GET method, therefore the information about the object is neglected in the server.

If the server needs to response different web browsers, then use the “user-agent” parameter, which tells what type of browser accesses to the server.

Body

The body in HTTP message is followed by the header. If the requested resources are needed to return to the client, the body contains the information of the contents such as text/html or image/gif.

However, the Smart toaster server only allows text/html contents, so a parameter, Content-type: text/html is attached in body part as the server responses to the client. The other content type is ignored in this project.

WEB

The flow of communication between the Smart toaster server and clients consist of two parts. First, clients request actions to the server. Second, the server responses to the client based on the request. When a client attempts to the server using a designated URL e.g. http://192.168.0.195/setting, the server tries to interpret the request. If the request is valid, the server transmits a setting HTML page to the client.

As same as the setting response, a status HTML page is transmitted as a client requests the status web page from the server e.g. http://192.168.0.195/setting.

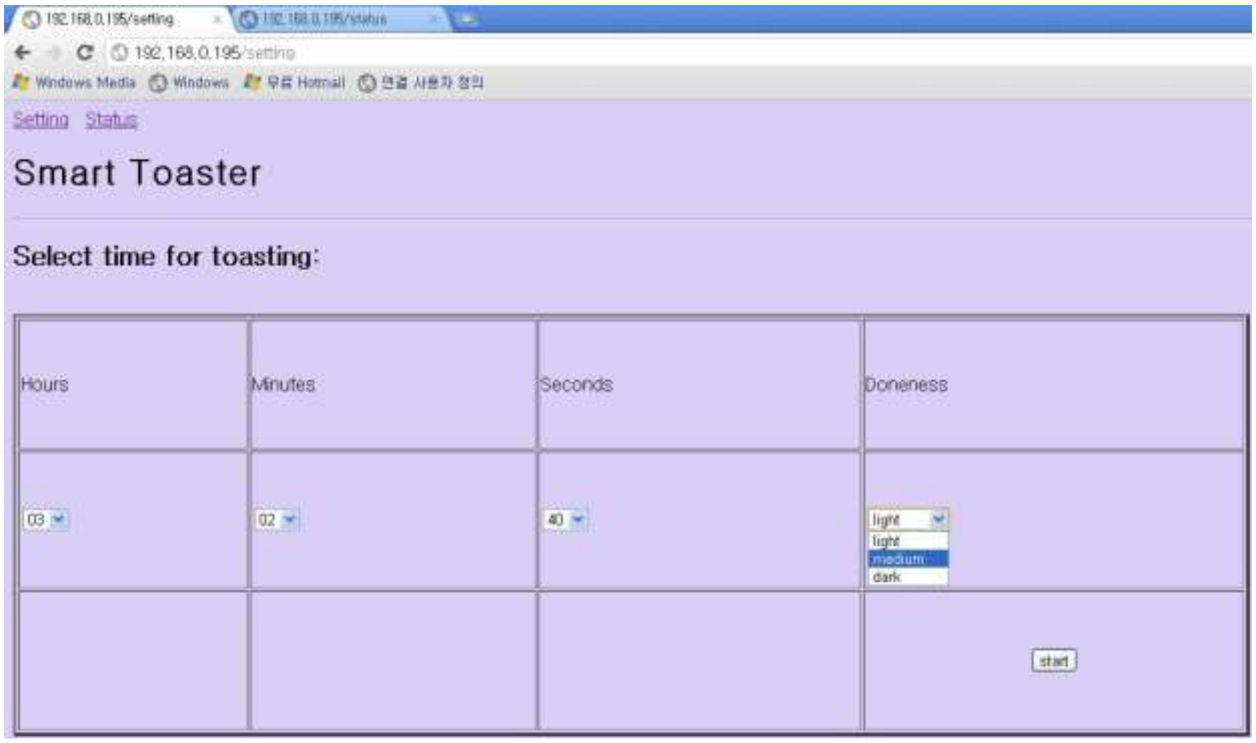
Setting

The setting web page is main page in the Smart toaster Server. The URL of this page is "http://server ip/setting". In this project, the server IP is 192.168.0.195. The setting page provides the users to choose a start time for toasting. The select component in HTML is used to select hours, minutes, seconds, and doneness. Users are able to choose a time whenever they want to toast out of their home through the internet. After selecting a start time and doneness, the toaster will be started by clicking the start button on the web page.

Once the start button is pressed, the web browser send data to the Smart toaster server and the server operates the toaster according to the user data.

- *FUNCTIONS PROVIDED BY SETTING PAGE [*Figure 10 setting*]*

  - This menu provides the users to choose start time.
  - The users are able to set up the start time (year, month, days, hours, minutes, and seconds) using select buttons provided by GUI**.**
  - In addition, this menu allows the users to choose the degree of toasting or desired doneness using select button (e.g. light, medium, dark)**.**



Setting Status

Smart Toaster

Select time for toasting:

Hours	Minutes	Seconds	Doneness
03	02	40	light (light / medium / dark)
			start

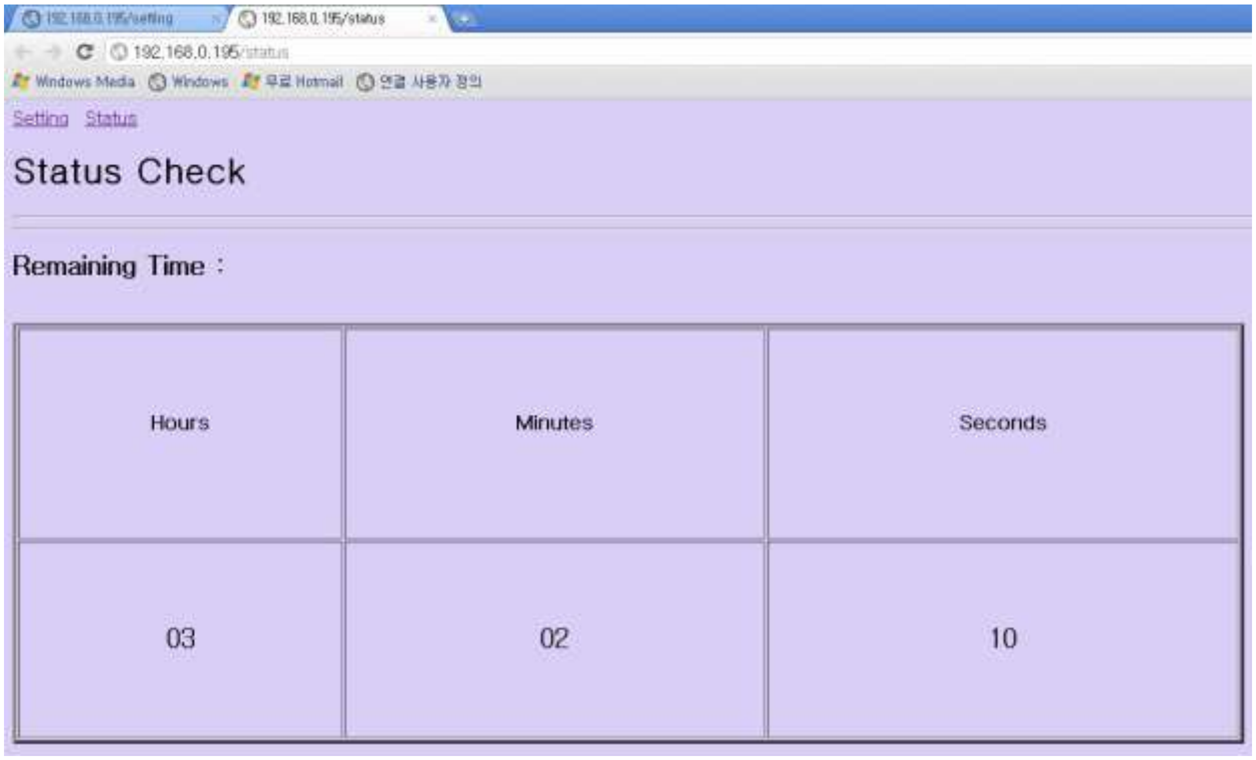
Figure 10 setting

Status

The status web page is linked on the setting web page. The URL of this page is http://server ip/status; however, users don't need to type the URL as the setting page does. Simply clicking the status' link menu leads users to status page. Before the check the status page, users are required to start toast first, otherwise, the Smart toaster server will not response the request properly.

- *FUNCTIONS PROVIDED BY STATUS PAGE [*Figure 11 status*]*

  - When the users choose the status menu, the users can easily notice the remained time to be done toast**.**
  - In the initial design, the status bar was provided by GUI telling the users how much toasting is done graphically. However, this graphical effect on this page was discarded because of the lack of the memory size of Arduino Uno and due to the server pending. Unlike other web servers operated in the PC or enterprise server, Arduino Uno has a limited memory size e.g. SDRAM 2KB, therefore, it can't handle the large amount of HTMLs, which consist of string.



Setting Status

Status Check

Remaining Time :

Hours	Minutes	Seconds
03	02	10

Figure 11 status

OUTCOMES (SMART TOASTER SERVER)

This section tells what the initial expectations are. Based on these expectations, what results are achieved and not achieved. In addition, this section involves what the test results are against expectations.

Initial Design

The Smart toaster server was designed for the first time, the users are able to select the start time for toast in the setting web page. As the users want to check the status regarding how long the toast will take to be done, displaying a remaining time as well as a graphical status bar provided on the status web page. Consequently, the Smart toaster server fully provides a remote controlled operation for the users.

Initial GUI design (setting)

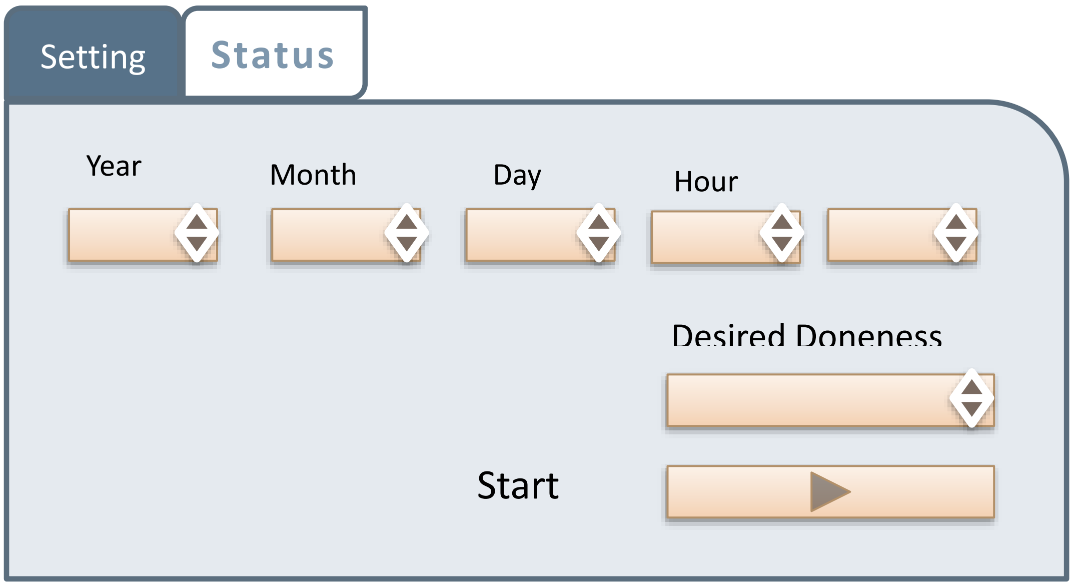


Figure 12 Initial setting GUI

Initial GUI design (status)

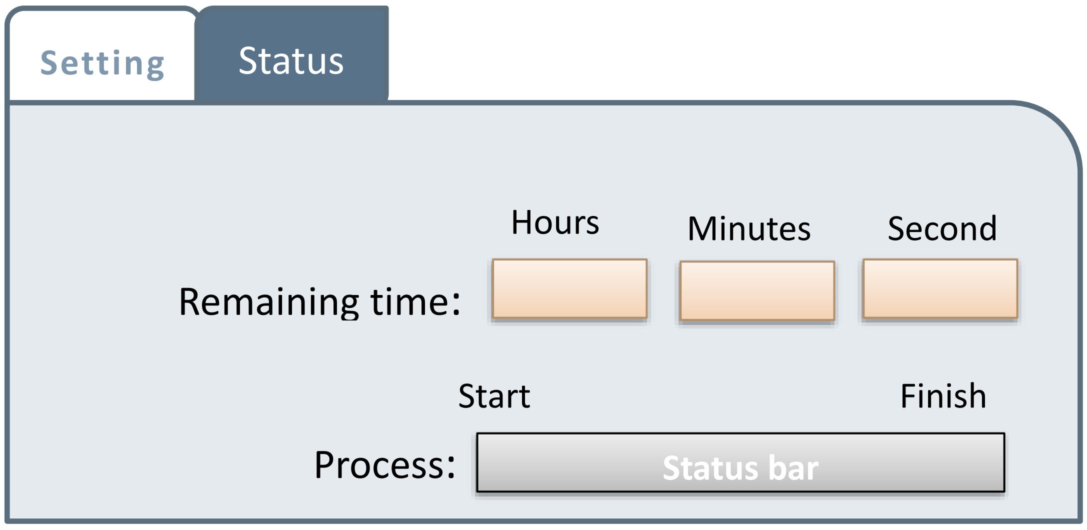


Figure 13 Initial status GUI

Current Status

As seen in Figure 10 setting and Figure 11 status, the setting web page is implemented almost as same as the initial design[Figure 12 Initial setting GUI]. The difference between the initial design and current design is that in the initial design, the users can select the setting and status web pages using the selection tap provided on the setting web page. However, the current setting web page provides a link menu instead of selection tap.

Initially, the status web page planned to provide not only the remaining time, but also a graphical status bar [Figure 11 status GUI]. However, this function couldn't be implemented in current status page since in order to do that, java scripts are needed to add up the status web page, which brings about overusing the memory in Arduino Uno. Due to the lack of the memory, this function was ruled out in the current version.

Moreover, before combining the other members' modules, the Smart toaster server serves both setting and status web pages properly. However, the server was pending due to the lack of memory after combining the other members' modules. For this reason, the status web page was not part of the final version of the server.

Test results

1. Test setting web page
    - A user accesses to the Smart toaster server with a server IP
    - Type 192.168.0.195/setting on the web browser
    - As seen in figure xxx, choose the start time for toast and doneness, then press the start button.
    - As soon as the start button pressed, the web browser transmits data to the Smart toaster server.
    - The server calls timer function, and if the server time is same as the start toasting time, the toaster will start.
2. Test status web page.
    - This page hasn't tested with a real data from the sensor's modules.
    - The test was done using the arbitrary time only for testing the status page.
    - A user accesses to the status web page clicking the status' link on the setting web page.
    - Assume the remaining time is sent and display the remaining time on the status web page.

Problems

While the implementation and test are proceeding, three problems have taken place. First of all, the Smart toaster server fully provides both setting and status web page to the users due to the lack of the resources. String object and library corresponding with the object provide convenient functions to the developer so that the developer can save a time for development. However, using the String object and the library of this object consume a lot of memory rather than using the primitive type and library corresponding with this type.

Since HTMLs consist of sequence of the string, in order to server multi web pages, they require more than 2KB provided by Arduino Uno. Also, if considering enough space for other members' modules, the Smart toaster server need more memory.

Second, the development tool provided by Arduino is made of JAVA, and as for the compiler the avr-gcc is used for controlling the Arduino Uno. JAVA doesn't fully support interfacing with C++; therefore, the developers have difficulties to debug codes.

Lastly, The Arduino Ethernet Shield only supports four TCP connections. Unlike other commonly used web server such as Apache and WebLogic fully providing connection management and session management, the Ethernet Shield is quite weak at managing those, so if a connection has problem, the server always is pending.

CONCLUSION (SMART TOASTER SERVER)

The above section, Problems, describes what problems are as the project has proceeded. This section addresses what can be the better solutions.

Recommendations

1. Apply MVC (Model, View, and Controller) model.
   - The Smart toaster server serves GUIs to the clients. Instead the server separates GUIs, business model, and the actions resulted from the request and response, those are implemented same logical and physical files. This brought about several side effects, for example, if one of those has a problem, then the others get influenced. As a result, the server easily turns to be pending.
   - Design view (HTML), Model (in this case, it could be any behaviors accessing core data such as ADC, Schedules), and Controller (request and response).

2. Apply distribution system.
   - In order to apply MVC model, the distribution system is necessary for the Smart toaster server. Placing HTMLs and controller to the web server (i.e. Apache) so that it not only lessens the loads that the Arduino Uno and Ethernet Shield have, but also serves more powerful resources and libraries provided by the web server.
3. Use flash memory or SD card.
   - The flash memory takes 32KB in Arduino Uno, so compared to SDRAM (2KB) the flash memory is more spacious than SDRAM.
   - HTML consists of sequence of string, so it takes a lot of memory. Instead of using SDRAM, using the sizeable SD card.
   - If the server uses the SD card, HTML tags are stored as contents. Whenever the request takes place, the server reads tag by tag and combines tags with data from the micro controller.

APPENDIX A (ARDUINO)

MAIN

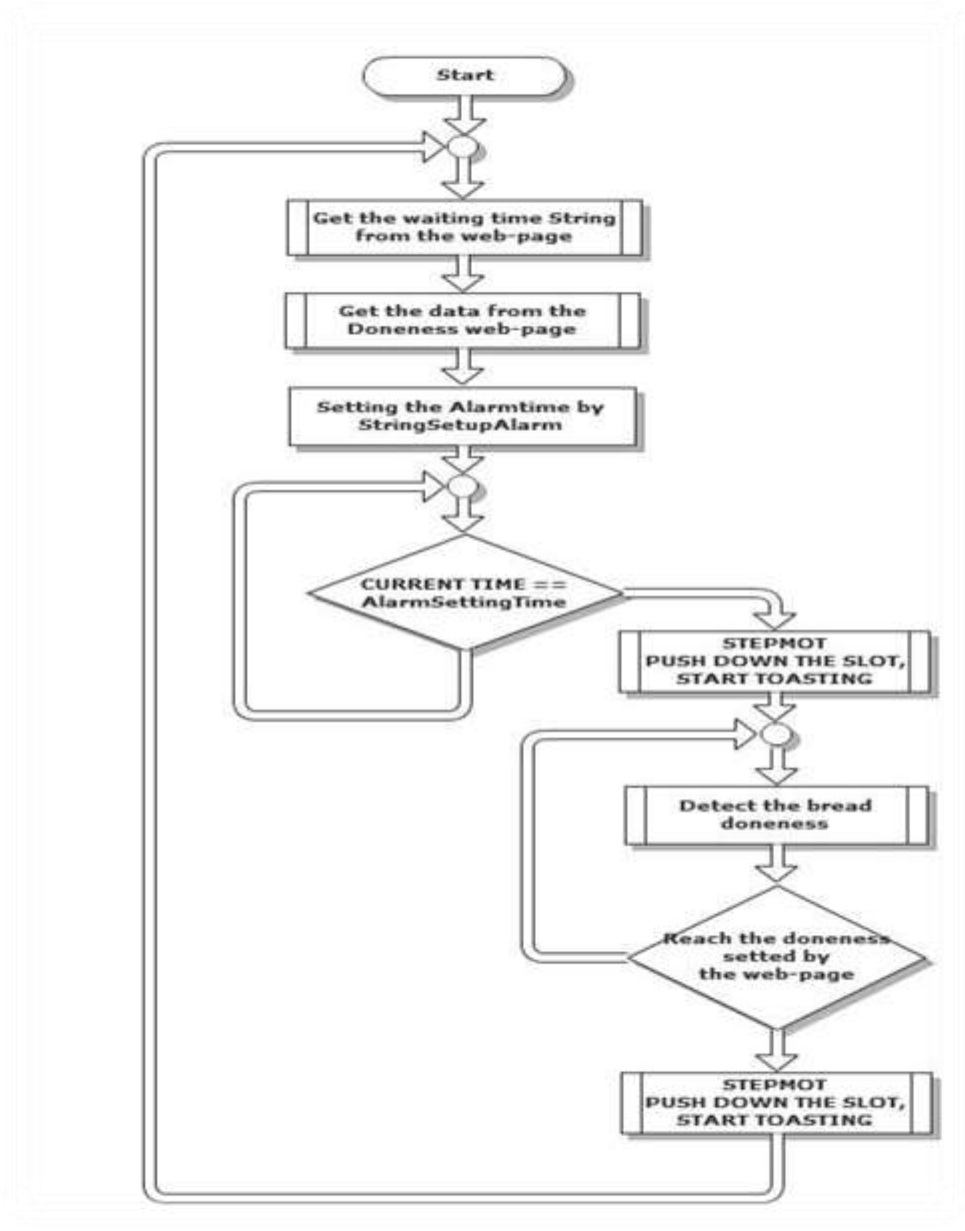

Start
Get the waiting time String from the web-page
Get the data from the Doneness web-page
Setting the Alarmtime by StringSetupAlarm
CURRENT TIME == AlarmSettingTime
STEPMOT PUSH DOWN THE SLOT, START TOASTING
Detect the bread doneness
Reach the doneness setted by the web-page
STEPMOT PUSH DOWN THE SLOT, START TOASTING

Figure 14 the main program flowchart

1. The waiting time and doneness are passed from HTML PAGE.
2. The Arduino waits until the set time.
3. Stepper motor pushes down the slot and starts toasting.

4. Arduino gets the color sensor data to see if the beard is done.
5. Pop up the beard when the toasting is finished.

StringSetupAlarm

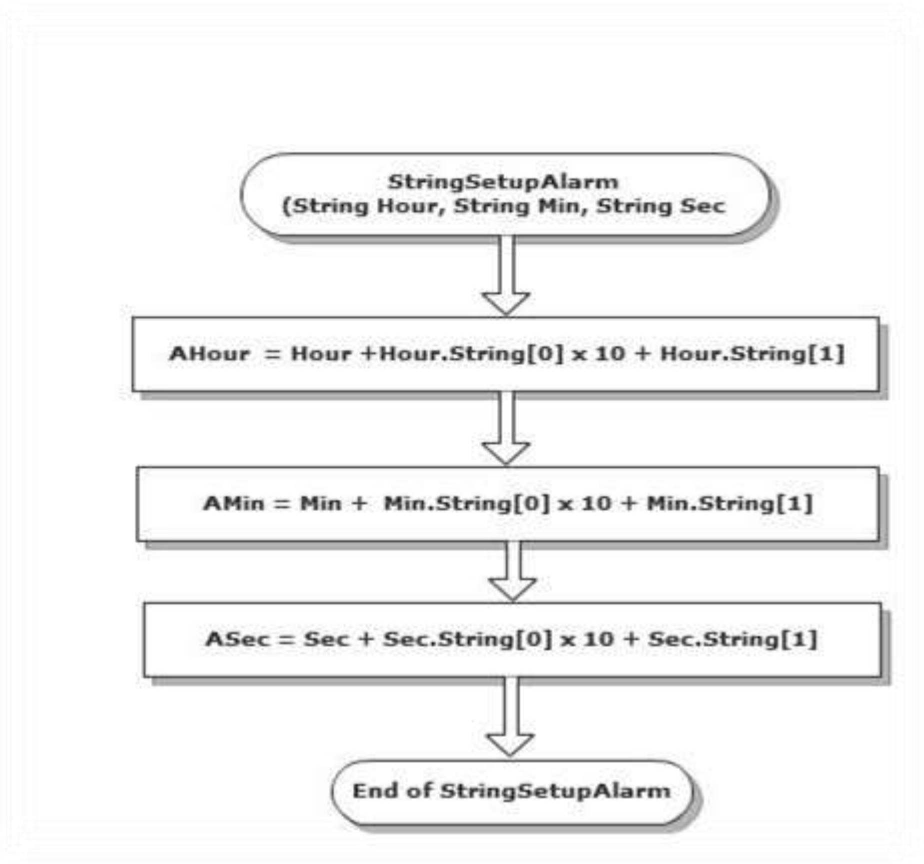

StringSetupAlarm
(String Hour, String Min, String Sec
AHour = Hour +Hour.String[0] x 10 + Hour.String[1]
AMin = Min + Min.String[0] x 10 + Min.String[1]
ASec = Sec + Sec.String[0] x 10 + Sec.String[1]
End of StringSetupAlarm

Figure 15 StringSetupAlarm flowchart

1. Get the string Hour, Min, and Sec from the parameters.
2. Set the waiting time value as adding the current time data by the value that represented by the string.

Timer Interrupt

The Timer Interrupt routine increments the time data.
In addition, it check the thermistor voltage in ADC in order to see exceed temperature.

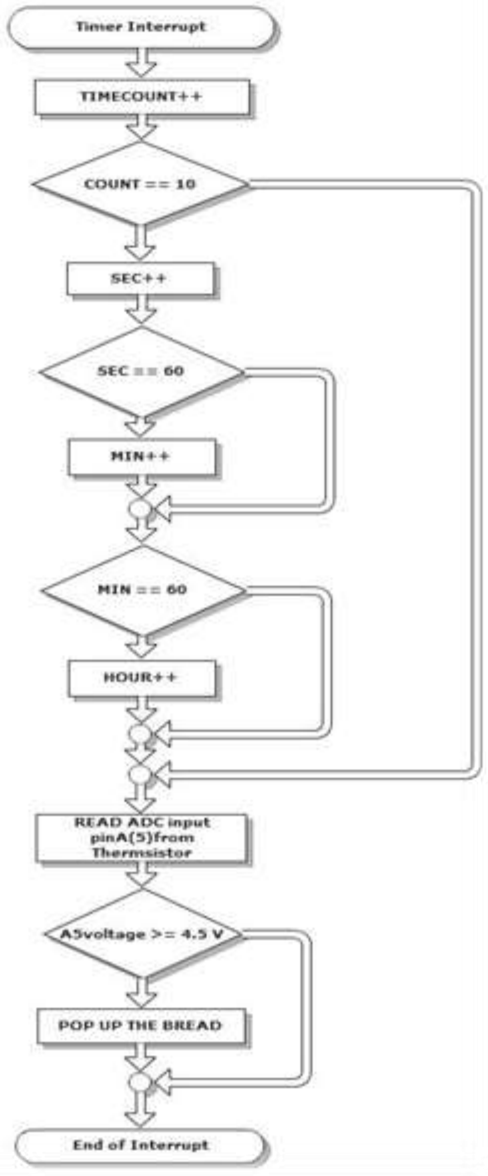


Figure 16 the timer interrupt routine

CHECKDONE

The CHECKDONE program sees if the time reaches the point StringSetupAlarm sets. If so, it returns zero to escape a while loop. Otherwise, it returns one.

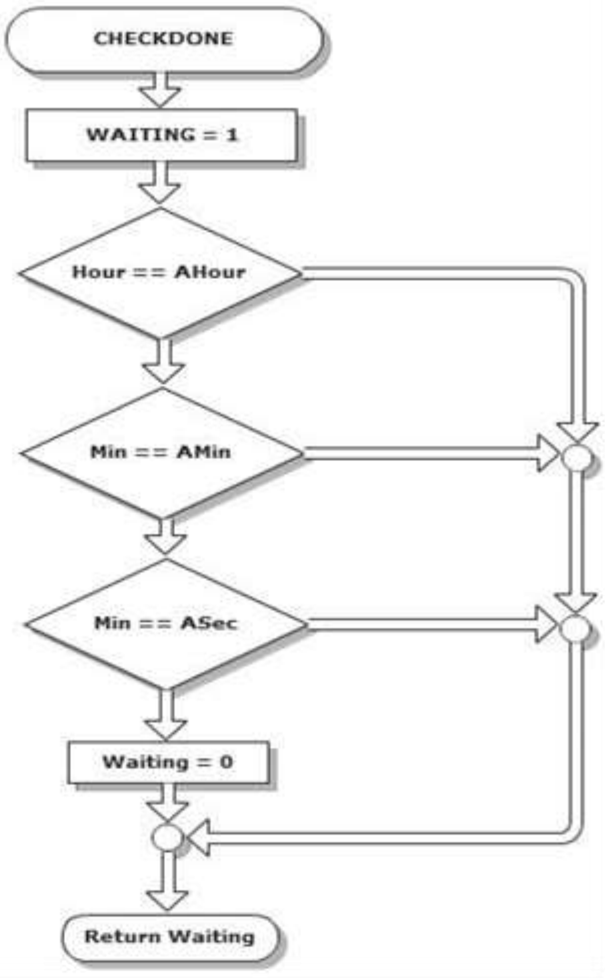


Figure 17 the CHECKDONE flowchart

STEPMOT

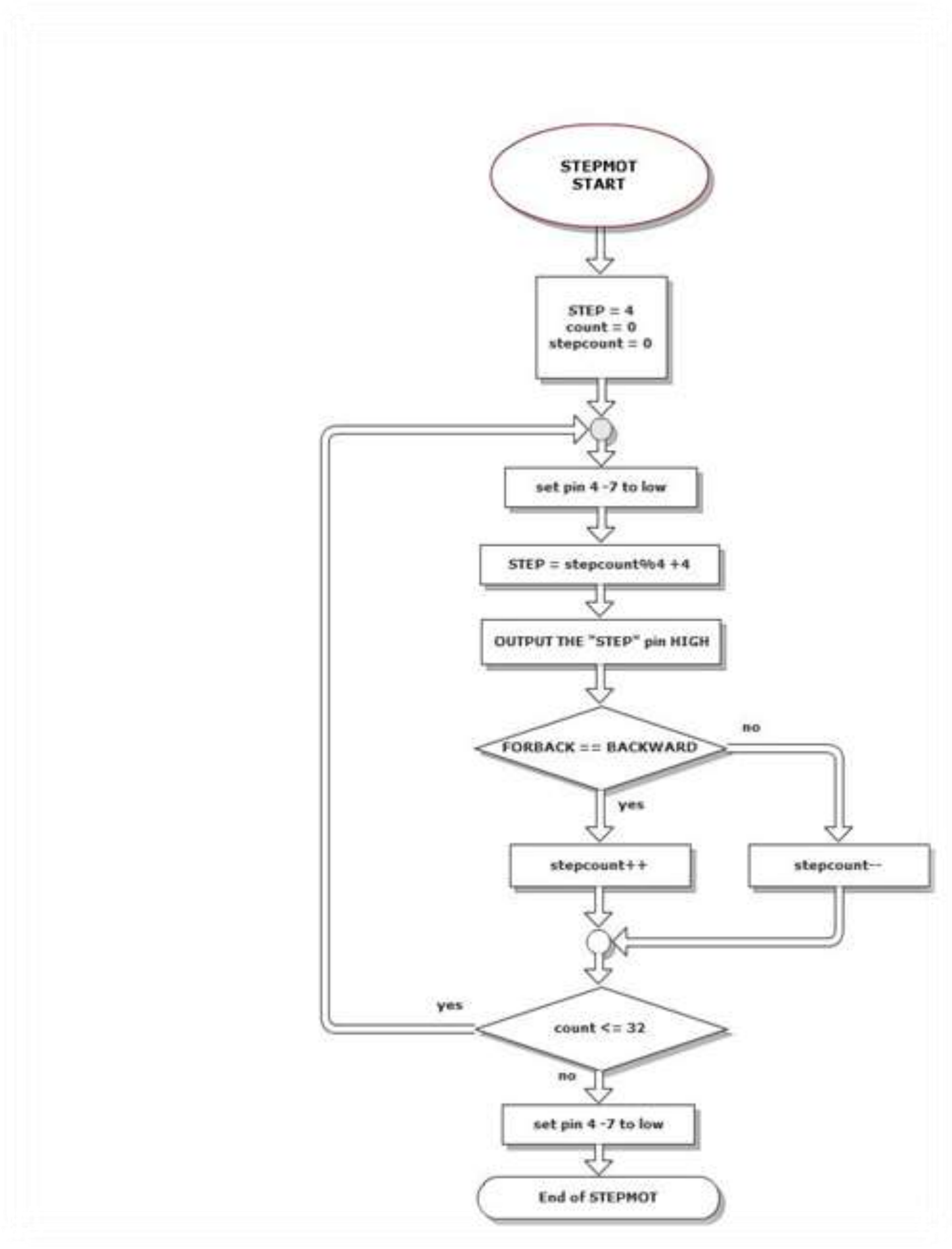


Figure 18

The function generates pulses on PIN4 to PIN 7 to the stepper motor control IC in order to turn the step motor. The direction whether pushing down or pulling up is determined by the parameter (FORBACK).

APPENDIX B (WEB)

This appendix contains terminologies that have not been fully explained in the Smart toaster server's documentation.

Flow chart (Set up)

As the Smart toaster server starts, the server executes the set up function. Using the Ethernet Shield library provided by Arduino mounts the server given MAC address, IP, Gateway address, and Subnet Mask. The clients only need to know the server IP to access the Smart toaster server.

- *ADDRESS INFORMATION*
  - MAC ADDRESS : 0XDE, 0XAD, 0XBE, 0XEF, 0XFE, 0XED
  - IP : 192,168,0, 195
  - GATEWAY : 192, 168, 0, 1
  - SUBNET MASK : 255, 255, 255, 0

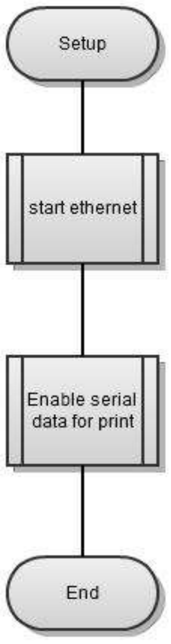


Figure 19 set up

Flow chart (Server main)

Once the set up function is called, the Smart toaster server is ready to response from the request of the client. HTTP messages are separated by CRLF, so the server tries to read request until meet the CRLF. The server uses a character buffer to store the request from the client. As the client's request is ended, the server disconnects the connection of the clients.

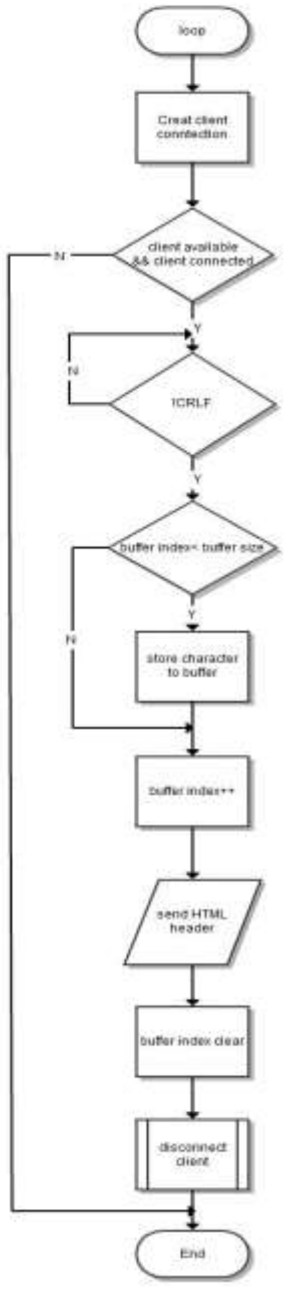


Figure 20 main loop

Flow chart (HTTP GET Parser)

This procedure is not made of function. This code is involved in the main server module; however, to void the main server's flowchart getting large and complicated, this module is apart from the main server module. The Smart toaster server only handles the HTTP GET method, so if a client request to start toast with parameters of the start time and doneness, the Smart toaster server attempts to parse the data such as hours, minutes, seconds

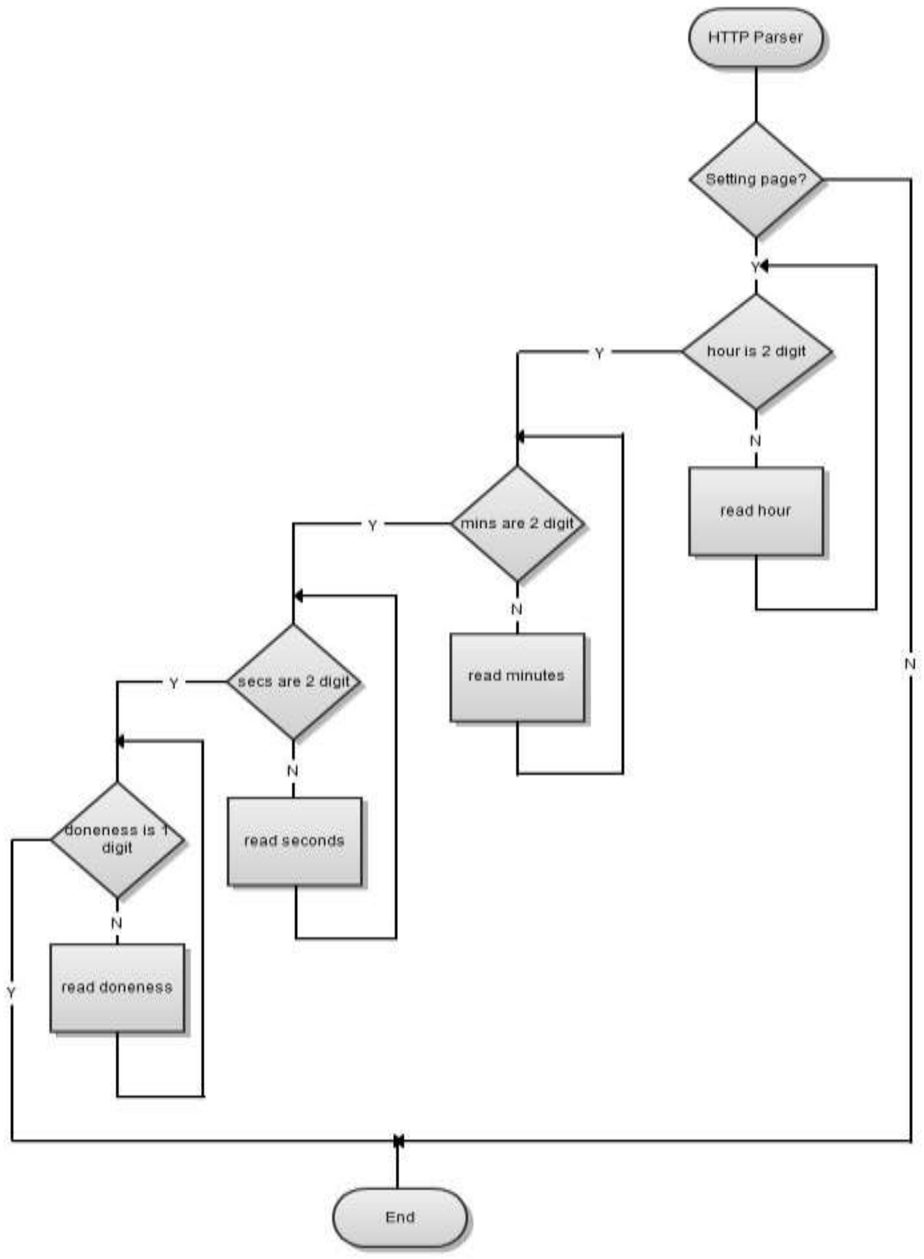


Figure 21 HTTP parser

Codes (Web Server and Sensor modules)

```
#include <SPI.h>
#include <Ethernet.h>
#include"Smart.h"

#include <SoftwareSerial.h>

#define FALSE 0
#define TRUE  1

//DEFINES FOR TOAST VALUES

#define lRl 0x40
#define lRh 0x42
#define mRl 0x36
#define mRh 0x47
#define dRl 0x32
#define dRh 0x39
#define lGl 0x30
#define lGh 0x37
#define mGl 0x30
#define mGh 0x37
#define dGl 0x30
#define dGh 0x36
#define lBl 0x48
#define lBh 0x51
#define mBl 0x46
#define mBh 0x54
#define dBl 0x46
#define dBh 0x53

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192,168,0, 195 };
byte gateway[] = { 192, 168, 0, 1 }; //ip address of the gatewa or router
byte subnet[] = { 255, 255, 255, 0 }; //ip address of the gatewa or router

Server server(80);                                      //server port

//String readString = String(50); //string for fetching data from address
#define STRING_BUFFER_SIZE 40
char buffer[STRING_BUFFER_SIZE];
char flag = false;

Smart toaster;

unsigned char carr[3];

boolean getColour(char *rgb);
void colour_init(void);
boolean testToast(char level, char r, char g, char b);

SoftwareSerial Color90(2, 3);

void setup(){
//start Ethernet
  Ethernet.begin(mac, ip, gateway, subnet);

//enable serial datada print
  Serial.begin(4800);
  Serial.println("Server Start!");
  toaster =  Smart::Smart();//call Timer


}
void loop(){
```

```
// Create a client connection
Client client = server.available();
  if (client) {
    int bufindex = 0; // reset buffer
    char c;
    if (client.connected() && client.available()) {

        buffer[0] = client.read();
        buffer[1] = client.read();
        //Serial.print(buffer[0]);
        //Serial.print(buffer[1]);
        bufindex = 2;
        // read the first line to determinate the request page
        while (buffer[bufindex-2] != '\r' && buffer[bufindex-1] != '\n') { // read full row and
save it in buffer
          c = client.read();
          //Serial.print(c);
          if (bufindex<STRING_BUFFER_SIZE) buffer[bufindex] = c;
          bufindex++;
        }
        //Serial.println(buffer);

         // client.println("<table width=\"300\"></table>");
        client.println("HTTP/1.1 200 OK");
        client.println("Content-Type: text/html");
        client.println();
        client.println("<html><body bgcolor=\"#D8CEF6\">");
        client.println("<a href=\"/setting\">Setting</a>&nbsp &nbsp<a href=\"/status\"
target=\"_blank\">Status</a>");
        client.println();
        char * startSetting =  strstr(buffer, "/setting");
        char * startStatus =  strstr(buffer, "/status");
        unsigned char index,i = 0;

          //char[] hours,mins,secs,doneness=;//add space for null terminator
           if(startSetting)
            {

               String hours ="";
               String mins ="";
               String secs ="";
               String doneness ="";//add space for null terminator

              //searching hours
              startSetting = strstr(buffer, "hr");
              index =0;//rese
              if(startSetting)
              {
                index = index+3;//reset initial point for searching hour

                for(i=0;i< 2; i++)
                {
                  hours += startSetting[index++];
                }
                //Serial.print("hours : "); Serial.println(hours);
              }
              //searching minutes
              //GET /setting?hr=00&m=01&sc=10&d=L HTTP/1P

              startSetting = strstr(buffer, "m");
              index = 0;//reset index
              if(startSetting)
              {
                index = index +2;//reset initial point for searching min
                for(i=0; i<2; i++)
                {
                  mins += startSetting[index++];

                }
                //Serial.print("mins : "); Serial.println(mins);
              }

              //searching seconds
              //GET /setting?h=01&m=01&sc=10&d=L HTTP/1.1
              startSetting = strstr(buffer, "sc");
              index = 0;//reset index
              if(startSetting)
              {
                index = index +3;//reset initial point for searching sec
                for(i=0; i<2; i++)
                {
                  secs += startSetting[index++];
                }
                Serial.print("secs : "); Serial.println(secs);

              }

               //searching doneness
              //GET /setting?hour=01&min=01&sec=10&d=L HTTP/1.1
              startSetting = strstr(buffer, "d");
              index = 0;//reset index
              if(startSetting)
              {
                index = index +2;//reset initial point for searching doneness
                 for(i=0; i<1; i++)
                {
                  doneness +=startSetting[index++];
                   Serial.print("doneness : "); Serial.println(doneness.charAt(0));
                }
                //Serial.print("doneness : "); Serial.println(doneness);
              }
              //call timer to start toasting

               if(hours != "")
               {


                    toaster.StringSetupAlarm( hours, mins, secs );
                    while(toaster.CHECKDONE())
                    {
                      if(toaster.OLD_NEW_SEC())
                      {
                       //oaster.PRINTTIME(); //
                      }

                    } //end while
                    Serial.println("Start Toaster!!!!!!!!!!");
                    STEPMOT(1);//init motor
                    //Serial.println("Start Motor!!!!!!!!!!");
                    colour_init(); //start colour sensor
                    while(flag == 0){
                      do{
                            flag = getColour(carr);
                        }while(flag == 0);
                       flag = testToast(doneness.charAt(0),carr[0], carr[1], carr[2]);
                       Serial.print("R :");Serial.print(carr[0], HEX); Serial.print(" G
:");Serial.print(carr[1], HEX); Serial.print(" B :");Serial.println(carr[2], HEX);
                    }//end while
                    //toast is ready
                    flag = 0;
                    STEPMOT(0);

             }//end while
             hours,mins,secs,doneness= "";
             memset(buffer, 0, 40);//clear buffer

             //clearing string for next read
            //send first heading
            client.println("<h1>Smart Toaster</h1>");
            client.println("<hr />");
            //output some sample data to browser

            //drawing simple table
            client.println("<h2>Select time for toasting:</h2>");
            client.println("<br />");
            client.println("<table border=3 width=\"80%\" height=\"40%\">");
            client.println("<form name='time' method=get>");

client.println("<tr><td>Hours</td><td>Minutes</td><td>Seconds</td><td>Doneness</td></tr>");
            client.println("<tr><td><select name=hr ><option value=\"00\">00</option><option
value=\"02\">02</option><option value=\"03\">03</option></select></td>");
            client.println("<td><select name=m><option value=\"00\">00</option><option
value=\"01\">01</option><option value=\"02\">02</option><option value=\"03\">03</option>");
            client.println("<option value=\"05\">05</option></select></td>");
            client.println("<td><select name=sc ><option value=\"10\">10</option><option
value=\"20\">20</option>");
            client.println("<option value=\"30\">30</option><option
value=\"40\">40</option><option value=\"50\">50</option></select></td>");
            client.println("<td><select name=d ><option value=\"L\">light</option><option
value=\"M\">medium</option><option value=\"D\">dark </option></select></td>");
            client.println("</tr><tr><td/><td/><td/><td align='center'><input type=submit
value=start></td></tr>");
            client.println("</form></table>");
            client.println("<br />");
            client.println("<hr />");
            client.println("</body></html>");
            //clearing string for next read

          }


           bufindex = 0;//for next

          }

        //delay for client to receive
        delay(1);
        client.stop();
    }


}//end loop

boolean getColour(unsigned char *rgb)
{

  unsigned char rByte[9];   //data recieved
  boolean rr;               //flag for correct data

  Color90.begin(4800);      //start software serial connection
  Color90.print("= (00 $ m) !"); //read color data (send m command)
  pinMode(3,INPUT);    //recieve color data from colorpal

  //read color data from colorPAL

  rByte[0] = Color90.read();   //check if data coming in is color info or not
  if( rByte[0] == '$' ) {

    //get 9 bytes, 3 for each colour (RGB)
    for(int i=0; i<9; i++) {
      rByte[i] = Color90.read();
      //store 9 bytes in temp array
    }

        rr = TRUE;  //signal that color data was obtained with call to getColour


        //transfer to new rgb array
       // DISPLAY RGB VALUES OBTAINED
       rgb[0] = (rByte[1] * 16)+ (rByte[2] - 0x30); //RED IN RED BYTE
       rgb[1] = (rByte[4] * 16)+ (rByte[5] - 0x30); //GRN IN GRN BYTE
```

```
        rgb[2] = (rByte[7] * 16)+ (rByte[8] - 0x30); //BLU IN BLU BYTE

  }
  else
    {
      rr = FALSE;  //getColour did not receive any color data
    }


  return rr;  //return flag for if colour data was collected

}//getcolor



///////////////////////////////////////
//    colour_init()
//
//    initiation sequence to enable
//    the colorpal color sensor to
//    read and output colour
//////////////////////////////////////
void colour_init(void)
{
  Color90.begin(4800);

  //follow sequence for startup as shown in datasheet for colorPAL
  pinMode(2,INPUT);
  pinMode(3,INPUT);
  digitalWrite(2,HIGH); // Enable the pull-up resistor
  digitalWrite(3,HIGH); // Enable the pull-up resistor

  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
  digitalWrite(2,LOW);
  digitalWrite(3,LOW);

  pinMode(2,INPUT);
  pinMode(3,INPUT);



  while( digitalRead(2) != HIGH || digitalRead(3) != HIGH ) {

    delay(50);
  }

  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
  digitalWrite(2,LOW);
  digitalWrite(3,LOW);
  delay(80);

  pinMode(2,INPUT);
  pinMode(3,OUTPUT);
  delay(100);



}//c_init


/////////////////////////////////////////////////////
//      testToast()
//
//      FUNCTION FOR CHECKING IF TOAST IS READY
//
//      function takes rgb values from color sensor
//      and compares the values to defined ranges
//      for three different toasting levels
//      (l, m, d) the toasting level to check for
//      is also input as a char. if the values are
//      in range it indicated the toasting is done
//      and will return TRUE. will return FALSE
//      if color data is not in range.
//
//      USE:
//   |----------------------------------|
//   |   do{                            |
//   |      flag = getColour(carr);     |
//   |     }while(flag == 0);           |
//   |----------------------------------|
//
//      to wait until colour data is collected (1 sample)
//
//
//
/////////////////////////////////////////////////////
boolean testToast(char level, char r, char g, char b){

//have defines for lRl, lRh, mRl, mRh, dRl, dRh
//                lGl, lGh, mGl, mGh, dGl, dGh
//                lBl, lBh, mBl, mBh, dBl, dBh

//declare variables
        //ranges
        char rRl;
        char rRh;
        char rGl;
        char rGh;
        char rBl;
        char rBh;
        //flags
        char rgbflag = 0x00;
        boolean tdone = FALSE;


//check level input for toasted level to check for Light, Medium, Dark
switch(level){

                case 'l':
                case 'L':
                        //set range to l
                        rRl = lRl;
                        rRh = lRh;
                        rGl = lGl;
                        rGh = lGh;
                        rBl = lBl;
                        rBh = lBh;
                        break;

                case 'm':
                case 'M':
                        //set range to m
                        rRl = mRl;
                        rRh = mRh;
                        rGl = mGl;
                        rGh = mGh;
                        rBl = mBl;
                        rBh = mBh;

                        break;

                case 'd':
                case 'D':
                        //set range to dark
                        rRl = dRl;
                        rRh = dRh;
                        rGl = dGl;
                        rGh = dGh;
                        rBl = dBl;
                        rBh = dBh;
```

```
                      break;

               default:
                      break;
               }//end switch

//check each colour (rgb) if in range
//check REDs

if(r >= rRl && r <= rRh){
       rgbflag += 0x01;   //red in range
Serial.println("r+ ");
}


//check GRNs

if(g >= rGl && r <= rGh){
  Serial.println("g+ ");
       rgbflag +=  0x02; //grn in range
}


//check BLUs

if(b >= rBl && r <= rBh){
  Serial.println("b+ ");
       rgbflag += 0x04;   //blu in range
}


//if all three colours (or just two?) are in range than output TRUE

if(rgbflag == 0x07)
{
//RETURN TRUE

tdone = TRUE;
}


return tdone;

}//tt
```

APPENDIX C: TOAST TESTING

COLOUR SENSING

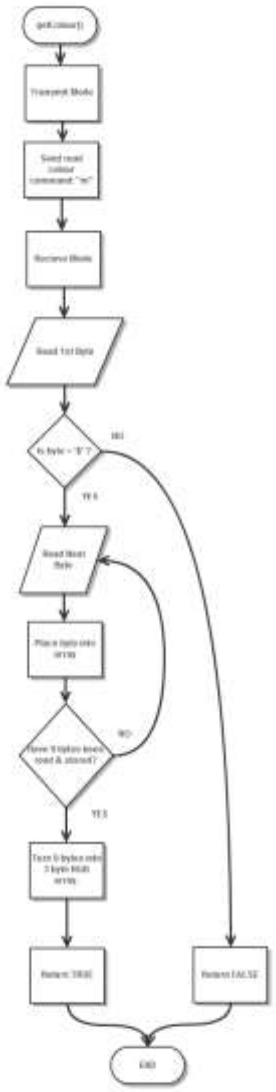


Figure 22

1. Micro sends “m” command to ColorPAL over serial line to initiate colour sampling.
2. Set Rx pin to INPUT to receive back data.
3. Read first byte to determine if proper colour data is being sent back from ColourPAL.
4. Fill array with received colour data.
5. Convert received values to R, G, & B bytes.
6. Return TRUE to signal successful colour data collection.

COLOUR TESTING

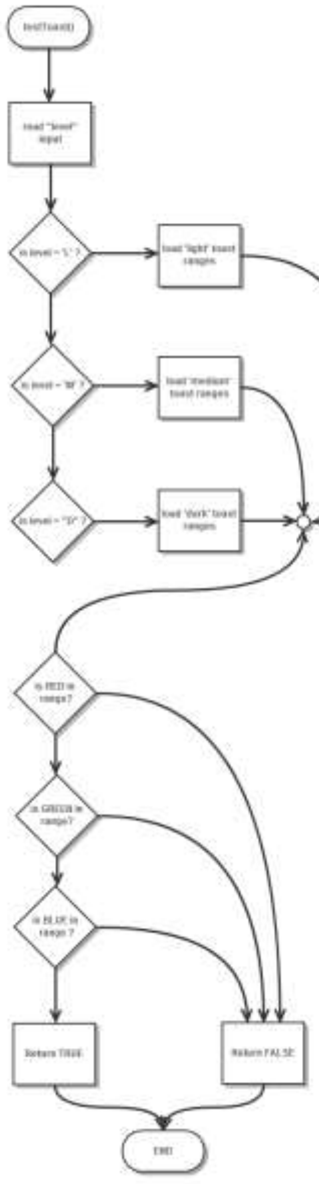


Figure 23

1. The specified level of toasting to test for is input and used to select the colour ranges to test for.
2. Each colour, R, G, &B is tested against the High and Low of the specified range.
3. If all colours are in range, TRUE is returned to signal the toast is toasted to level specified.

APPENDIX D: GLOSSERY

Arduino Uno

The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started.

ATmega328

An ATmega328 in DIP package, pre-loaded with the Arduino Duemilanove (16MHz) Bootloader. This will allow you to use Arduino code in your custom embedded project without having to use an actual Arduino board.

To get this chip working with Arduino IDE, you will need an external 16MHz crystal or resonator, a 5V supply, and a serial connection. If you are not comfortable doing this, we recommend purchasing the Arduino Uno board that has all of these built into the board.

HTTP GET Method

In HTML, one can specify two different submission methods for a form. The method is specified inside a FORM element, using the METHOD attribute. The difference between METHOD="GET" (the default) and METHOD="POST" is primarily defined in terms of form data encoding. The official recommendations say that "GET" should be used if and only if the form processing is idempotent, which typically means a pure query form. Generally it is advisable to do so. There are, however, problems related to long URLs and non-ASCII character repertoires which can make it necessary to use "POST" even for idempotent processing.

HTTP POST Method

A POST request is used to send data to the server to be processed in some way, like by a CGI script. A POST request is different from a GET request in the following ways:

- There's a block of data sent with the request, in the message body. There are usually extra headers to describe this message body, like Content-Type: and Content-Length:

- The request URI is not a resource to retrieve; it's usually a program to handle the data you're sending.
- The HTTP response is normally program output, not a static file.

OSI 7 Layer

The Open Systems Interconnection model (OSI model) is a product of the Open Systems Interconnection effort at the International Organization for Standardization. It is a prescription of characterizing and standardizing the functions of a communications system in terms of abstraction layers. Similar communication functions are grouped into logical layers. An instance of a layer provides services to its upper layer instances while receiving services from the layer below.

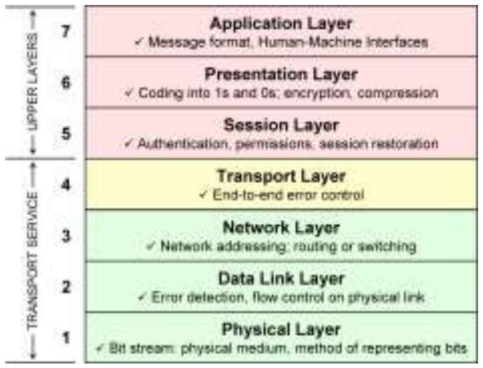

UPPER LAYERS
TRANSPORT SERVICE
7
Application Layer
✓ Message format, Human-Machine Interfaces
6
Presentation Layer
✓ Coding into 1s and 0s; encryption, compression
5
Session Layer
✓ Authentication, permissions, session restoration
4
Transport Layer
✓ End-to-end error control
3
Network Layer
✓ Network addressing; routing or switching
2
Data Link Layer
✓ Error detection, flow control on physical link
1
Physical Layer
✓ Bit stream: physical medium, method of representing bits

Figure 24 OSI 7 Layer

RJ45

RJ45 is a standard type of connector for network cables. RJ45 connectors are most commonly seen with Ethernet cables and networks. RJ45 connectors feature eight pins to which the wire strands of a cable interface electrically. Standard RJ-45 pin outs define the arrangement of the individual wires needed when attaching connectors to a cable.

Figure 25 RJ45

SPI

The Serial Peripheral Interface Bus or SPI (pronounced like "S.P.I." or "spy") bus is a synchronous serial data link standard named by Motorola that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) lines. Sometimes SPI is called a "four-wire" serial bus, contrasting with three-, two-, and one-wire serial buses.

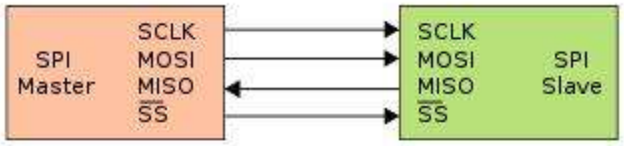


Figure 26 SPI

TCP and UDP

TCP (Transmission Control Protocol) is the most commonly used protocol on the Internet. The reason for this is because TCP offers error correction. When the TCP protocol is used there is a "guaranteed delivery." This is due largely in part to a method called "flow control." Flow control determines when data needs to be re-sent, and stops the flow of data until previous packets are successfully transferred. This works because if a packet of data is sent, a collision may occur. When this happens, the client re-requests the packet from the server until the whole packet is complete and is identical to its original.

UDP (User Datagram Protocol) is another commonly used protocol on the Internet. However, UDP is never used to send important data such as webpages, database information, etc. UDP is commonly used for streaming audio and video. Streaming media such as Windows Media audio files (.WMA), Real Player (.RM), and others use UDP because it offers speed! The reason UDP is faster than TCP is because there is no form of flow control or error correction. The data sent over the Internet is affected by collisions, and errors will be present. Remember that UDP is **only** concerned with speed. This is the main reason why streaming media is not high quality.

TCP/IP

TCP/IP (Transmission Control Protocol/Internet Protocol) is used to connect hosts on the internet. The TCP/IP model describes a set of general design guidelines and implementations of specific networking protocols to enable computers to communicate over a network. TCP/IP provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed and received at the destination. Protocols exist for a variety of different types of communication services between computers.

BILL OF MATERIAL

The budget list is just the approximate amount we expected to spend. It may vary due to the project design improvement.

Table 2 BOM

BUDGET ITEM	AMOUNT
Microcontroller (Arduino)	$ 35
Light Sensor	$ 25
Thermostat Sensor	$ 20
Toaster (Second Hand)	$ 20
Solenoid	$ 35
DB-9 connector	$ 2
Other Small Components (which are included switch, LED light, solder…etc.)	$ 30

REFERENCES

Arduino. (n.d.). *Ethernet*. Retrieved from Schematic: http://www.arduino.cc/en/Reference/Ethernet

Arduino. (n.d.). *Learning*. Retrieved from Examples: http://arduino.cc/en/Tutorial/HomePage

Arduino. (n.d.). *String*. Retrieved from http://arduino.cc/en/Reference/StringObject

PARALLAX. (2009). *COLORPAL DATASHEET.*

Wikipedia. (n.d.). *OSI Model*. Retrieved from OSI Model: http://en.wikipedia.org/wiki/OSI_model

Wikipedia. (n.d.). *Serial Peripheral Interface Bus*. Retrieved from SPI: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

Wikipedia. (n.d.). *TCP/IP Model*. Retrieved from TCP/IP Model: http://en.wikipedia.org/wiki/TCP/IP_model