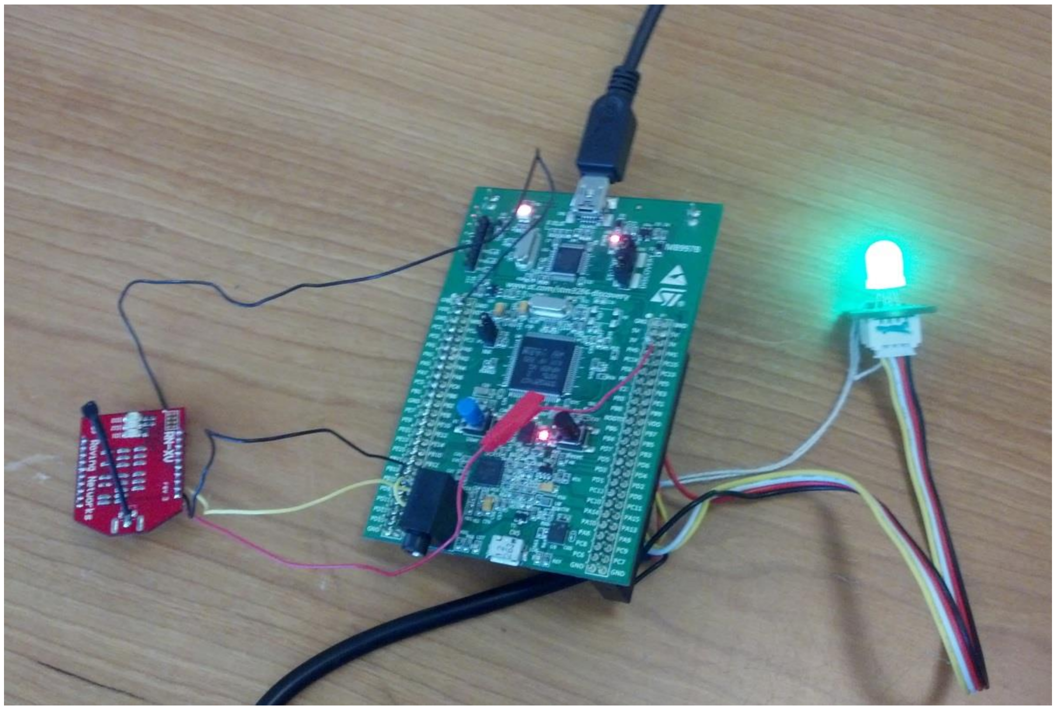
ENGS 62 Final Project:

HTTP Client for Monitoring Financial Data



Alex Murphy

March 12, 2013

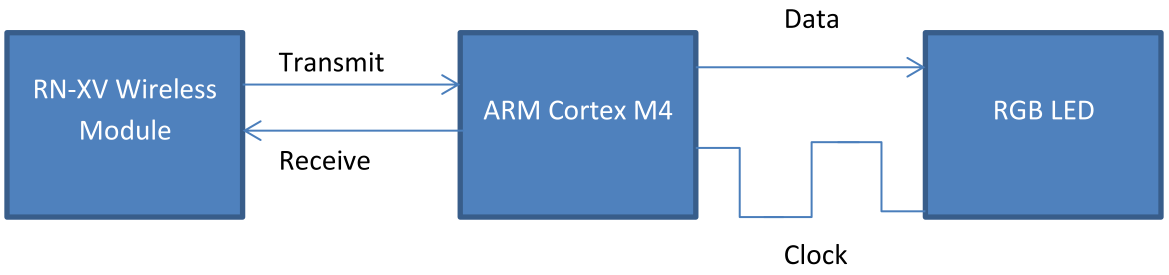
Project Description

The goal of this project is to utilize the STM32F4 Discovery board coupled with peripheral devices to create a visual representation of the performance of the stock market throughout the day. The main reasons for pursuing this project are to better understand wireless networking, data transfer, and serial communication. There are three main components that comprise this system: the wireless module, the microprocessor, and the tricolor LED. These three pieces communicate through serial data transfer using USART, both synchronously and asynchronously.

The process begins by pre-configuring the wireless module to connect to the proper network and manage the correct settings for use as an HTTP client. The first step was to connect to the network "Dartmouth Registered." This network is hidden, but is accessible anywhere on campus provided the MAC address of the device is registered with the network. Connecting to this network requires entering the WLAN settings for SSID name and authentication mode. The next step is to configure the device for use as a client. This configuration is then saved on the wireless module so on reboot the device will restore these settings. Now the wireless module has been configured to receive HTML data following a valid HTTP request once a connection has been opened. The wireless device communicates back and forth with the microprocessor using UART: the microprocessor sends settings and commands and the wireless module sends back HTML data.

The LED also communicates serially, but it instead requires synchronous communication which the USART ports on the microprocessor are capable of handling. The LED does not exactly accept USART data, so the way values are sent to the LED required modification. For one, the LED takes a data in and a clock in, which can be configured on the microprocessor. Furthermore, USART data is sent with the least significant bit first, but the LED expects the most significant bit to arrive first. Once these differences are accounted for and the clock is configured, sending RGB values to the LED will result in the correct display of color.

All that remains is to receive the data from the HTTP request, process it, and then send it to the LED for output.



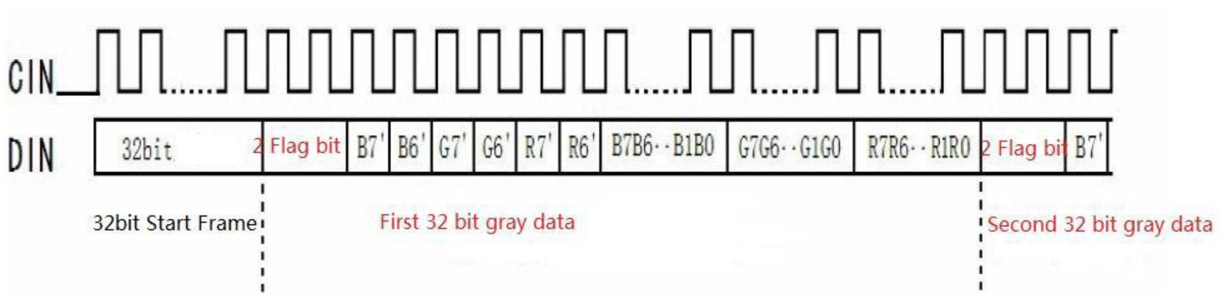
Peripheral Devices

The two peripheral devices for this project as mentioned are the Roving Networks XV WiFly Module and the Grove Chainable RGB LED. The RN XV wireless module is a simple way to communicate over WiFi and requires only four connections: power (3.3V), ground, TX, and RX. The module communicates using UART and has two modes of operation: command mode and data mode. Data mode is the active mode upon powering on, and command mode is entered by sending "$$$" from the microprocessor to the wireless module. In command mode, the programmer configures WLAN, DNS, and IP settings in addition to actually opening the connection to the webserver. Once connected to the "Dartmouth Registered" network, configuration is done by typing the following commands into the serial terminal. The microprocessor receives characters from USART1 (serial terminal) and sends them over USART3 (wireless module).

```
set ip proto 18  /* this enables html client */
set dns name download.finance.yahoo.com /* set name of the webserver */
set ip address 0 /* Have IP address assigned*/
set ip host 0 /* Use DNS */
set ip remote 80 /* use port 80 */
set comm remote 0 /* Turn off the REMOTE string */
```

The first command sets the IP protocol to enable TCP server and client and to enter HTTP client mode. The second line sets the name of the host for the TCP/IP connection. For this project, the webserver download.finance.yahoo.com is used to retrieve stock market data. The third and fourth commands tell the module to use DNS to automatically assign the device an IP address. The fifth line tells the module to connect to the specified webserver on port 80, the default port used by HTTP. Finally, the last line disables sending a string to the TCP client when the TCP port is opened. These commands do not have to be input every time the device runs; rather they are saved in a configuration file on the wireless module and are loaded when the device is powered on.

The LED also relies on serial communication, but expects a slightly different format. The timing diagram, shown below, from the datasheet explains how the light expects to receive data. Colors are represented in the standard way, one 8-bit number each for red, green and blue. To initiate setting the color, the microprocessor must send a 32-bit zero, or in this case 4 8-bit zeros. Then, a flag bit is sent followed by the "inverse codes" and then the 8-bit numbers for blue, green and red in that order. Once the RGB values have been sent, the microprocessor must send another 32-bit zero to indicate the end of transmission. As previously mentioned the LED requires a clock input, which means that this method of communication is serial and synchronous. This model of LED has a feature in which one could daisy chain up to five LEDs together and set their colors independently, but for this application only one LED is used. It would theoretically be trivial to add several LEDs to indicate the status of several stocks.



Webserver

One integral part of this project is accessing a remotely hosted webserver which supplies the necessary data.   Yahoo! Finance provides essentially an API for finance data where a user or device can send parameters in the URL that correspond to data and stock symbol(s). The server returns a comma-separated values file (CSV) with the stock symbols and the requested up-to-date numbers. For this application, four example stock tickers are used: ^GSPC (S&P 500), GOOG (Google), GE (General Electric), and DIS (Walt Disney Co.). The relevant data is the daily percent change which is represented by the special tag “p2.” The returned data will have an HTTP header, and the data in the form “TICKER”,“+p.pp%”. A simple finite state machine processes these characters and then converts the percent change from a string to an integer.

Initializations

On the microprocessor the initializations necessary to operate the peripherals are all related to USART. USART1 and USART3 are configured as in previous labs, using the transmit and receive ports. USART1 is used to communicate with the serial terminal running on the computer to enter commands for the wireless module and to echo back responses from the wireless chip. The primary purpose for setting up USART1 like this is debugging. USART3 is used to communicate with the wireless module, which expects and sends standard UART commands. Interrupts are also enabled for USART1 and 3. USART2, however, is configured differently to communicate with the LED. As mentioned, the LED requires a clock for synchronous communication. The USART on the M4 has a clock out pin which allows for this mode of communication. In order to connect to the LED some modifications must be made to the default USART settings. Basically, the clock must bet set to work as in the timing diagram above. The settings for the USART clock are in Control Register 2. There are four settings that must be set appropriately: clock enable (on), clock polarity (steady high), clock phase (first data transfer is on second clock change), and last bit clock pulse (sends an additional clock pulse). Interrupts are not enabled for USART2.

As in the labs, the LEDs are enabled on GPIOD and the user button is simply activated by enabling GPIOA.

USART init methods (USART1 and 3 init methods are essentially identical):

```
void USART2_init(void){
   RCC->AHB1ENR |= RCC->AHB1ENR | RCC_AHB1ENR_GPIODEN;    /* Enable port d clock */
   RCC->APB1ENR |= RCC_APB1ENR_USART2EN;    /* Enable USART2 Periph. clock */
   GPIOD->MODER = (GPIOD->MODER & (0xfffff3ff)) | 0x800; /* Set Pin 5 as AF */
   GPIOD->MODER = (GPIOD->MODER & (0xffff3fff)) | 0x8000; /* Set Pin 7 as AF */
   GPIOD->AFRL = (GPIOD->AFRL & 0x0f0fffff) | (0x70700000); /* Set pins 5 and 7 as AF7 */
   USART2->CR1 |= 0x2000;
   USART2->BRR = 1667;
   USART2->CR1 |= (0x8); /* Enable transmit */

   USART2->CR2 |= (0x400); /* Clock polarity high 0x400*/
   USART2->CR2 |= (0x200); /* clock phase */
   USART2->CR2 |= (0x100); /* last bit clock pulse */
   USART2->CR2 |= (0x800); /* Enable Clock for synch */
}

void USART1_init(void){
   uint32_t *nvic_iser1 = 0xE000E104;
   RCC->AHB1ENR |= RCC->AHB1ENR | 0x2;    /* Enable port B clock */
   RCC->APB2ENR |= RCC_APB2ENR_USART1EN;    /* Enable USART1 Periph. clock */

   GPIOB->MODER = (GPIOB->MODER & 0xffff0fff) | 0x0000a000; /* Set pins 6 and 7 as AF */
   GPIOB->AFRL = (GPIOB->AFRL & 0x00ffffff) |  (0x77000000); /* Set both as AF7 */

   USART1->CR1 |= (0x2000);
   USART1->BRR = 1667;
   USART1->CR1|= 0x2c; /* Enable transmit, receive and interrupts */
   *nvic_iser1 |= 0x20;
}
```

Problems

Working on this project created a number of complications in getting the three devices to communicate correctly. The first step of sending color codes to the LED was initially difficult because the datasheet was difficult to locate. Initially the only information I had on the LED was a segment of Arduino code, from which I was able to mostly understand how to send data to the device. However, it was not until finding the timing diagram that I was actually able to get the device working. Ultimately, I had to hookup the data pin and clock to the oscilloscope and compare it to the provided diagram. Adjusting the USART settings as described above finally resulted in a working light.

USART typically sends data with the least significant bit (LSB) first and the most significant bit (MSB) last. The LED however requires the MSB first. To overcome this difference, I used a function to reverse the bits of each 8-bit integer sent to the light. I implemented this function with a statically declared array in which each value corresponds to the reverse of its index. This results in a very fast function for reversing a number, which would otherwise take linear time. There is not a huge improvement in performance because only 8-bit numbers are reversed for this application, but I still found it to be an interesting solution to reversing the bits of a number.

The wireless module also had a number of complications. The first difficulty was that I expected that the device needed more power than the microprocessor's board could provide. The transmit function of the device requires a fairly large amount of current, which is greater than the max current provided by the board at most transmit rates. However, I did not end up running into this problem as this project mostly just receives rather than sends data.

Code

main.c

```
#include "stdint.h"     /* uint32_t, etc... */
#include "stm32f4xx.h"  /* Useful definitions for the MCU */
#include "LED.h"        /* C routines in LED.c */
#include "USART2.h"     /* assembly routines in USART.S */

#include "systick.h"
#include "lock.h"

uint8_t send_flag = 0;
uint8_t terminal_flag = 0;
uint8_t data_done = 0;

uint8_t reverse(uint8_t);
void push_ticker(uint8_t);
void push_percent(uint8_t);
int strtoi();
uint8_t data_in;
uint8_t quotes = 0;
int pi = 0;
char percent[10];
char stock[10];
int ti = 0;

void __attribute__ ((interrupt)) USART3_handler(void)
{
  data_in = USART3_recv();
  USART1_send(data_in);
  if (data_in == '"')
    quotes++;
    if(quotes == 6)
      push_ticker('\0');
    else if (quotes == 8)
      push_percent('\0');

  switch (quotes){
  case 1:
    data_done = 0;
    break;
  case 5:
    if (data_in != '"')
      push_ticker(data_in);
    break;
  case 7:
    if (data_in != '"')
      push_percent(data_in);
    break;
  case 8:
    quotes = 0;
    pi = 0;
    ti = 0;
```

```
        data_done = 1;
        break;
    }

}

void __attribute__ ((interrupt)) USART1_handler(void)
{
    uint8_t data;
    data = USART1_recv();
    if(data == '<')
        terminal_flag = 0;

    if(terminal_flag != 0)
        USART3_send(data);

    if(data=='>')
        terminal_flag = 1;
}

int strtoi(){
    int val=0;
    char tmp[10];
    uint8_t i = 1;
    uint8_t place = 0;
    uint8_t ch;
    while(percent[i] != '.'){
        ch = percent[i++];
        place++;
    }

    if (place == 1){
        tmp[0] = percent[0];
        tmp[1] = 48;
        tmp[2] = percent[1];
        tmp[3] = '.';
        tmp[4] = percent[3];
        tmp[5] = percent[4];

        percent[0] = tmp[0];
        percent[1] = tmp[1];
        percent[2] = tmp[2];
        percent[3] = tmp[3];
        percent[4] = tmp[4];
        percent[5] = tmp[5];
    }

    val = val + ((percent[1])-48) * 1000;
    val = val + ((percent[2])-48) * 100;
    val = val + ((percent[4])-48) * 10;
    val = val + ((percent[5])-48) * 1;

    if(percent[0]=='-')
        val *= -1;
```

```
  return val;
}


char ticker[10];
void push_ticker(uint8_t c){
    ticker[ti++] = c;
}



void push_percent(uint8_t c){
    percent[pi++] = c;
}

void wait(uint32_t secs){
    int i = 0;
    float j=5;
    for(;secs>0;secs--){
        i = 0;
        for(; i<200000; i++){}
    }
}

void send32zero(){
    USART2_send(0);
    USART2_send(0);
    USART2_send(0);
    USART2_send(0);
}
uint8_t TakeAntiCode(uint8_t data){
    uint8_t temp = 0;
    if ((data & 0x80) == 0)
        temp |= 0x02;
    if((data & 0x40)== 0)
        temp|=0x01;
    return temp;
}
void processAndSend(uint8_t r, uint8_t g, uint8_t b){
    uint8_t b1=0;
    b1 |= 0x03 << 6;
    b1 |= TakeAntiCode(b) << 4;
    b1 |= TakeAntiCode(g) << 2;
    b1 |= TakeAntiCode(r);

    send32zero();
    USART2_send(reverse(b1));
    USART2_send(reverse(b));
    USART2_send(reverse(g));
    USART2_send(reverse(r));
    send32zero();
}

uint8_t reverse(uint8_t num){
    uint8_t ret;
    static const unsigned char BitReverseTable256[] =
    {
      0x00, 0x80, 0x40, 0xC0, 0x20, 0xA0, 0x60, 0xE0, 0x10, 0x90, 0x50, 0xD0, 0x30, 0xB0, 0x70, 0xF0,
      0x08, 0x88, 0x48, 0xC8, 0x28, 0xA8, 0x68, 0xE8, 0x18, 0x98, 0x58, 0xD8, 0x38, 0xB8, 0x78, 0xF8,
      0x04, 0x84, 0x44, 0xC4, 0x24, 0xA4, 0x64, 0xE4, 0x14, 0x94, 0x54, 0xD4, 0x34, 0xB4, 0x74, 0xF4,
      0x0C, 0x8C, 0x4C, 0xCC, 0x2C, 0xAC, 0x6C, 0xEC, 0x1C, 0x9C, 0x5C, 0xDC, 0x3C, 0xBC, 0x7C, 0xFC,
      0x02, 0x82, 0x42, 0xC2, 0x22, 0xA2, 0x62, 0xE2, 0x12, 0x92, 0x52, 0xD2, 0x32, 0xB2, 0x72, 0xF2,
      0x0A, 0x8A, 0x4A, 0xCA, 0x2A, 0xAA, 0x6A, 0xEA, 0x1A, 0x9A, 0x5A, 0xDA, 0x3A, 0xBA, 0x7A, 0xFA,
      0x06, 0x86, 0x46, 0xC6, 0x26, 0xA6, 0x66, 0xE6, 0x16, 0x96, 0x56, 0xD6, 0x36, 0xB6, 0x76, 0xF6,
      0x0E, 0x8E, 0x4E, 0xCE, 0x2E, 0xAE, 0x6E, 0xEE, 0x1E, 0x9E, 0x5E, 0xDE, 0x3E, 0xBE, 0x7E, 0xFE,
      0x01, 0x81, 0x41, 0xC1, 0x21, 0xA1, 0x61, 0xE1, 0x11, 0x91, 0x51, 0xD1, 0x31, 0xB1, 0x71, 0xF1,
      0x09, 0x89, 0x49, 0xC9, 0x29, 0xA9, 0x69, 0xE9, 0x19, 0x99, 0x59, 0xD9, 0x39, 0xB9, 0x79, 0xF9,
      0x05, 0x85, 0x45, 0xC5, 0x25, 0xA5, 0x65, 0xE5, 0x15, 0x95, 0x55, 0xD5, 0x35, 0xB5, 0x75, 0xF5,
      0x0D, 0x8D, 0x4D, 0xCD, 0x2D, 0xAD, 0x6D, 0xED, 0x1D, 0x9D, 0x5D, 0xDD, 0x3D, 0xBD, 0x7D,
0xFD,
      0x03, 0x83, 0x43, 0xC3, 0x23, 0xA3, 0x63, 0xE3, 0x13, 0x93, 0x53, 0xD3, 0x33, 0xB3, 0x73, 0xF3,
      0x0B, 0x8B, 0x4B, 0xCB, 0x2B, 0xAB, 0x6B, 0xEB, 0x1B, 0x9B, 0x5B, 0xDB, 0x3B, 0xBB, 0x7B, 0xFB,
      0x07, 0x87, 0x47, 0xC7, 0x27, 0xA7, 0x67, 0xE7, 0x17, 0x97, 0x57, 0xD7, 0x37, 0xB7, 0x77, 0xF7,
      0x0F, 0x8F, 0x4F, 0xCF, 0x2F, 0xAF, 0x6F, 0xEF, 0x1F, 0x9F, 0x5F, 0xDF, 0x3F, 0xBF, 0x7F, 0xFF
    };
    ret = BitReverseTable256[num];
    return(ret);
}

void send_string(char *str){
    int i = 0;
    while (str[i]!='\0'){
        USART3_send(str[i++]);
    }
}

void send_string_term(char *str){
    int i = 0;
    while (str[i]!='\0'){
        USART1_send(str[i++]);
    }
}
```

```
void set_string(char *str){
    int i = 0;
    while (str[i]!='\0'){
        stock[i] = (str[i++]);
    }
    stock[i] = '\0';
}

void change_ticker(uint8_t ticker){
    uint8_t t = ticker%4;

    switch(t){
    case 0:
        toggle_red();
        set_string("^GSPC");
        toggle_orange();
        break;
    case 1:
        toggle_orange();
        set_string("GOOG");
        toggle_green();
        break;
    case 2:
        toggle_green();
        set_string("GE");
        toggle_blue();
        break;
    case 3:
        toggle_blue();
        set_string("DIS");
        toggle_red();
        break;
    }
}

uint8_t calc_color(int change){
    uint8_t green;
    if (change > 100)
        green = 255;
    else if (change < -100)
        green = 0;
    else
        green = 128 + (change * 128)/100;
    return green;
}
```

```
void send_cmds(){
    toggle_blue();
    send_string("\r exit\r");
    wait(5);
    send_string("$$$");
    wait(5);
    send_string("open\r");

    wait(5);
    send_string("GET /d/quotes.csv?s=");
    send_string(stock);
    send_string("&f=sp2 HTTP/1.1\n");
    send_string("Host: download.yahoo.finance.com\n");
    send_string("Connection: close\n");
    USART3_send('\n');

    while(data_done != 1){}
    send_string("\r exit\r");
    toggle_blue();
}

int main()
{
    LED_init();
    systick_init();
    button_init();


    LED_update(LED_BLUE_ON);
    LED_update(LED_BLUE_OFF);
    LED_update(LED_RED_ON | LED_BLUE_ON | LED_ORANGE_ON | LED_GREEN_ON );
    LED_update(LED_RED_OFF | LED_BLUE_OFF | LED_ORANGE_OFF | LED_GREEN_OFF);

    /* Initialize the USART for 9600, 8N1, send '!' - calls into USART2.S */
    /* NOTE: must set USART2 interrupt config register to enable TX/RX interrupts */
    USART2_init();
    USART3_init();
    USART1_init();

    __asm ("  cpsie i \n" );


    uint8_t R = 0;
    uint8_t G = 255;
    uint8_t B = 0;
    processAndSend(0,255,255);

    B=0;
    char c;
    int per;

    set_string("^GSPC");

    uint8_t tick = 0;
    uint32_t i = 1;
    uint32_t j = 0;
    uint32_t k = 0;
    uint32_t rel=0;

    wait(20);
    toggle_orange();
    while(1){
        data_done = 0;
        send_cmds();
        per = strtoi();
        G = calc_color(per);
        R = 255 - G;
        B = 0;
        while(data_done == 0){}
        processAndSend(R,G,B);

        i = 1;
        j = 0;
        k = 0;
        while(j++<1200){
            while(k++ <  9000000){
                i = check_button(i);
                if(rel>i && (rel>1000)){
                    change_ticker(++tick);
                    j = 1200;
                    k = 9000000;
                    i = 0;
                }
                rel = i;
            }
        }
    }
    /* We'll never reach this line */
    return 0;
}
```