```
/*
 * lab6Demo.c
 *
 * Created: 12/9/2018 7:21:32 PM
 * Author : Kaveh Pezeshki and Christopher Ferrarin
 *
 * MCU backend for Lab 6. Generates webpages as requested by the ESP8266, and
  interfaces with the onboard PIO and ADC peripherals to allow user control of
  an LED and to display the voltage on ADC channel 2.
 *
 * The LED on PA17 is controlled by the webpage, active low
 * The LED on PA18 is active when the webpage is transmitted to the ESP8266
  from the MCU
 *
 * In more detail:
 * 1) The MCU initializes peripherals
 * 2) The MCU loads any incoming bytes from UART into a buffer, and scans for
  the sequence '< ... >' Held within the angle brackets is the request from the
  microcontroller. Steps 3-5 are executed if a request is detected
 * 3) The MCU turns on PA17 if 'on' is within the request, and turns off PA17
  if 'off' is within the request
 * 4) The MCU reads the CH2 ADC voltage
 * 5) The MCU generates and sends the webpage to the ESP8266 over UART
 */

#include "easySamIO.h"   //peripheral header file
#include <string.h>       //string operations for parsing incoming requests
#include <stdio.h>        //float -> string conversion for ADC
#define LED_PIN 17        //LED controlled by webpage. Active low
#define TRANSMIT_PIN 18   //LED indicated webpage transmission. Active low
#define VOLTAGE_CHARS_TO_TRANSMIT 5 //The number of characters in the string
 representation of the CH2 voltage to transmit as a part of the webpage


//The webpage is separated into a 'start' and 'end' section. These sandwich the
 CH2 voltage, which is inserted between the two webpage arrays. The webpage is
 raw HTML
//Note: as " terminates the string, we escape the " with \". This is
 interpreted as the raw character '"' rather than as an escape character
const char* webpageStart = "<!DOCTYPE html><html>\n    <head>\n
 <title>E155 Web Server Demo Webpage</title>\n            </head>\n    <body>\n
 <h1>E155 Web Server Demo Webpage</h1>\n        <p>Current Microcontroller ADC:
 </p>\n        ";
//ADC CH2 voltage is printed between these
const char* webpageEnd   = "\n        <p>LED Control:</p>\n        <form
 action=\"on\">\n            <input type=\"submit\" value=\"Turn the LED on!
 \" />\n        </form>\n        <form action=\"off\">\n                <input
 type=\"submit\" value=\"Turn the LED off!\" />\n        </form>\n    </
 body>\n</html>\n";
```

```
//as we do not dynamically calculate webpage size for the constant start and
 end arrays, it is given as constant
const int webpageStartChars = 215;
const int webpageEndChars = 264;


void transmitWebpage() {
    digitalWrite(TRANSMIT_PIN, LOW);
    //first transmitting the initial section of the webpage
    for (int charCount = 0; charCount < webpageStartChars; charCount++) {
        uartTx(webpageStart[charCount]);
        if (webpageStart[charCount] == '\n') {uartTx('\r');} //some
         interpreters want a carriage return and line feed. Adding a carriage
         return if line feed is detected
    }

    //reading the ADC
    float ch2Voltage;
    char ch2VoltageStr[VOLTAGE_CHARS_TO_TRANSMIT];
    ch2Voltage = adcRead(CH2);
    //converting ADC voltage as a float to a string
    snprintf(ch2VoltageStr, VOLTAGE_CHARS_TO_TRANSMIT, "%f", ch2Voltage);
    //Transmitting the voltage string to the webpage. Note: snprintf transmits
     a null terminator as its last character. This breaks many string parsing
     functions, so we do not transmit it.
    for (int charCount = 0; charCount < VOLTAGE_CHARS_TO_TRANSMIT-1; charCount+
     +) {
        uartTx(ch2VoltageStr[charCount]);
    }
    //finally transmitting the final section of the webpage
    for (int charCount = 0; charCount < webpageEndChars; charCount++) {
        uartTx(webpageEnd[charCount]);
        if (webpageEnd[charCount] == '\n') {uartTx('\r');}
    }
    digitalWrite(TRANSMIT_PIN, HIGH);
}


int main(void) {
    /* Initialize the SAM system */
    samInit(); //peripheral initialization. See easySamIO.h
    pinMode(LED_PIN, OUTPUT); //LED_PIN is controlled by the webpage
    digitalWrite(LED_PIN, HIGH);
    pinMode(TRANSMIT_PIN, OUTPUT); //TRANSMIT_PIN is active when a webpage is
     being transmitted
    digitalWrite(TRANSMIT_PIN, HIGH);
    uartInit(4, 25); //initializing the UART with no parity, 9600 baud
    adcInit(ADC_BITS_12); //initializing the ADC with a precision of 12 bits
    adcChannelInit(CH2, ADC_GAIN_X1, ADC_OFFSET_OFF); //initializing channel 2
     of the ADC with no gain or offset
```

```
int currentRxState = 0;                //1 when there is an unread byte in
 the UART RX register, 0 otherwise
char character;                        //character read by the UART
char request[14] = "              ";  //14-character buffer to store the
 webpage request
int requestFound = 0;                  //0 only if '<...>' not in the
 request
int currentRequestChar = 0;            //The number of valid characters in
 request
const char requestStart = 0x3c;        //hex representation of character
 '<'
const char requestEnd = 0x3e;          //hex representation of character
 '>'

transmitWebpage();                     //on boot, transmit the webpage.
 This mitigates any potential request - response timing errors on system
 initialization



while (1)
{
    //checking whether there is a byte to read in the UART RX buffer
    currentRxState = UART_REGS->UART_SR.RXRDY;
    //process if there is an unread byte
    if (currentRxState == 1) {
        if(currentRequestChar == 14) {
            //if buffer is filled, wrap to start
            currentRequestChar = 0;
        }
        //read in unread character
        character = uartRx();
        //add character to buffer
        request[currentRequestChar] = character;
        //searching for substring
        int startInString = strchr(request, requestStart); //0 only if '<'
         not in the request
        int endInString   = strchr(request, requestEnd); //0 only if '>'
         not in the request
        //if a request start character is past the start of the buffer,
         empty the buffer with the start character in position 0
        if (startInString != 0 && currentRequestChar >= 2 && request[0] !=
         requestStart) { //SHOULD THIS BE >=1??
            request[0] = '<';
            for (int i = 1; i < 14; i++) {
                request[i] = ' ';
            }
            currentRequestChar = 0;
        }
        //if the buffer contains both the start and end characters, then
         the request is loaded. Process the request and return a webpage
```

```
            if (startInString != 0 && endInString != 0) {
                int ledOnInString = strstr(request, "on");    //Turn LED on if
                 'on' is in the request
                int ledOffInString = strstr(request, "off");  //Turn LED on if
                 'off' is in the request
                if(ledOnInString != 0) {
                    digitalWrite(LED_PIN, LOW);
                }
                if(ledOffInString != 0) {
                    digitalWrite(LED_PIN, HIGH);
                }
                //the request has been processed, and is therefore cleared
                for (int i = 0; i < 14; i++) {
                    request[i] = ' ';
                }
                //finally, transmitting the webpage
                transmitWebpage();
            }
            currentRequestChar += 1; //preparing to write in the next buffer
             index
        }

    }
}
```