Technical University of Cluj-Napoca

Design with Microprocessors

Smart clock

Radu Dragos Imbuzan
group 30434

Contents

Chapter 1

Summary

The objective of this project is to build a digital smart clock, capable of displaying the current date and time. Besides this, the clock displays the current temperature and humidity of the room. The time and also the date can be set by the user using the push buttons or with a serial connection for sending commands.

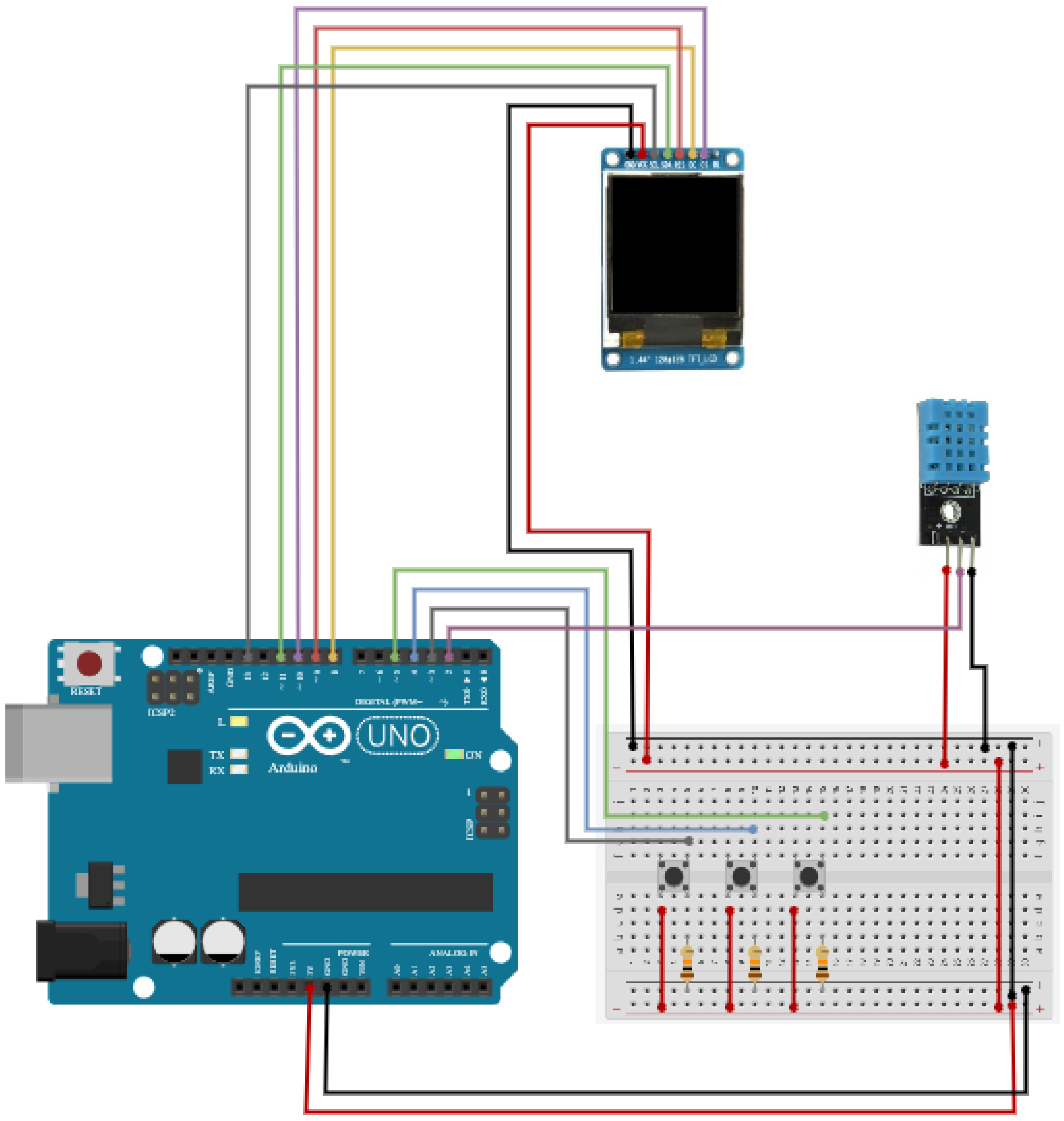
This project can be used at home, at work, or in any location where we need to know not only the time but also information about the current state of the room.

The components used to build this project are:

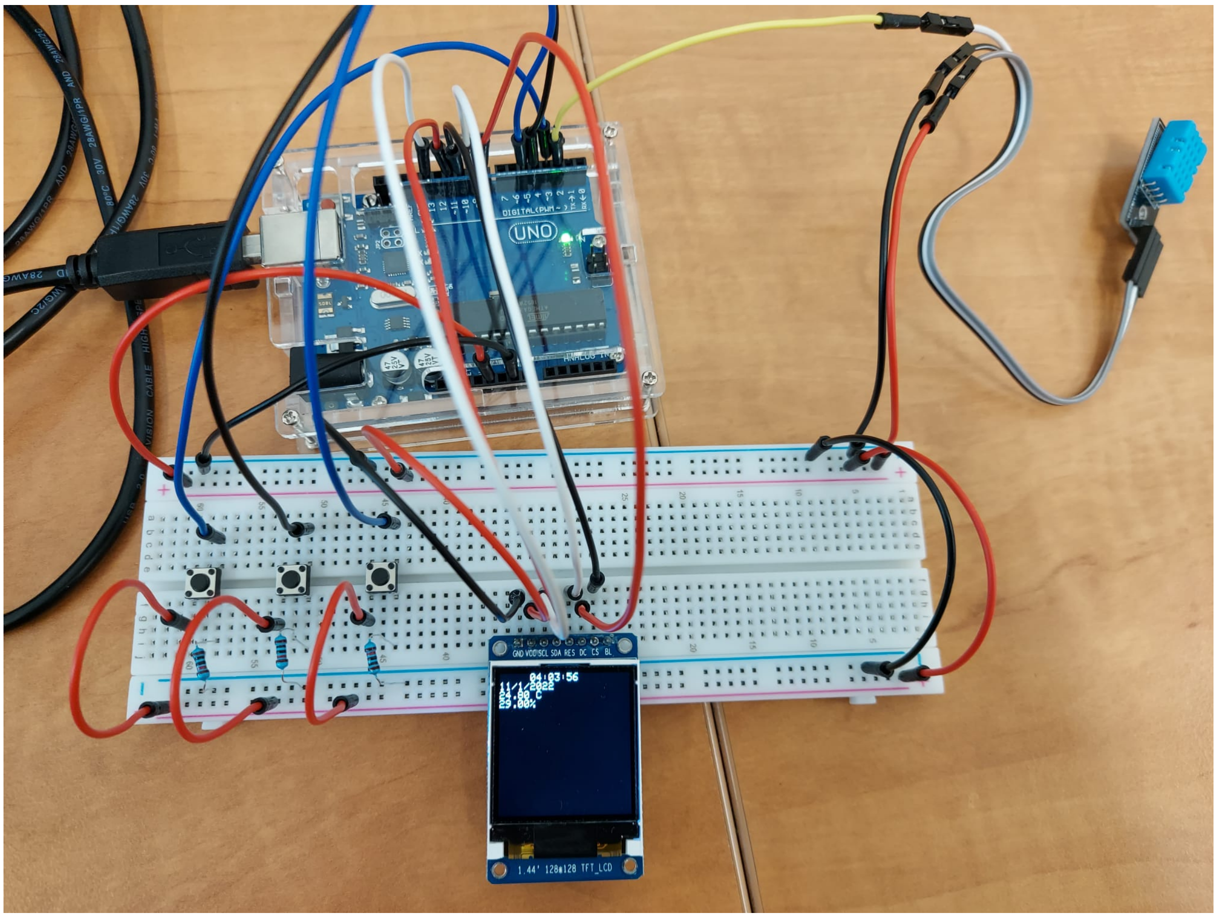
- An Arduino Uno
- DHT11 temperature and humidity sensor - this is an inexpensive component that is connected using only 3 wires, 1 for Vcc, 1 for GND and one for data.
- 1.44” 128x128 LCD module with SPI and ST7735 Controller - the display has a good resolution, useful for displaying data on multiple rows
- 3 push buttons
- 3 10K ohm resistors

Chapter 2

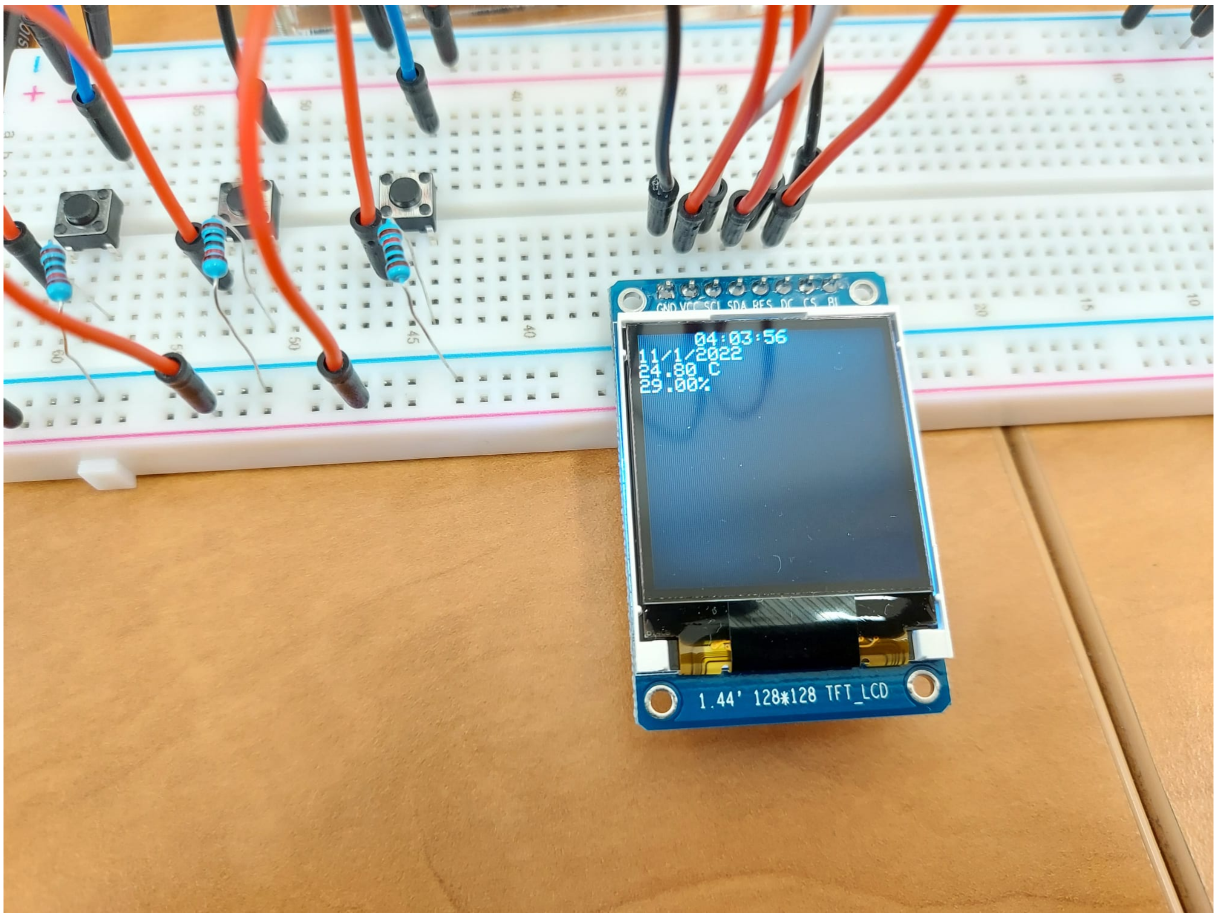
Schematic



Project schematic



The complete circuit



Display connections

Chapter 3

Flow of the program

In the setup function, we set the baud rate of the serial transmission, we initialize the TimerOne to call the function that increments the seconds, attach the interrupt to the setup pin, initialize the display and the temperature and humidity sensor, and in the end, we build the time and date string, with the initial values, to be displayed on the screen.

The main states of our program are the display state, in which we display the data on the screen every second, the setup state, in which we set the time from the buttons and the last state is the execute state, in which we execute the commands received from the serial transmission.

In the loop function, we first check if the command that we receive from the serial monitor is complete, and if it is then we call the executeCommand function. In the executeCommand we extract the name of the command and the parameter, and after that, we set the day, month, or year with the new value given as a parameter, if the value is valid. Because every component of the date is saved both as a string and as a number we have to set both of them. At the end of the function, we clear the row of the date text and write the new date. To clear the row, we use the clearCharPosition function, which takes as parameters, the x and y position, the length of the row, and the height of the row on the screen. To clear that portion of the screen we just draw a filled rectangle with the color equal to the color of the background.

To enter in the setup state we have to press the setup pin which triggers an interrupt and calls the changeSetupState function. In this function we enter or leave the setup state by setting the isInSetupState variable to 1 or 0, then we reset the seconds. If we enter the setup state then we stop the timer and if we exit the setup state we restart the timer. When we are in the setup state we debounce the remaining two push buttons. When a button is pressed and its state is set to HIGH in the debounce function, we call the buttonPressed function which increments the minutes or hours, based on the number of the button which was pressed.

The last state is the display state in which we display the information on the screen every second. We have distinct functions for displaying the time and date, the temperature, or the humidity.

Chapter 4

Code

Listing 4.1: Project code

```
#include <TimerOne.h>

#include "DHT.h"

#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_ST7735.h> // Hardware-specific library for ST7735
#include <SPI.h>

#define DHTPIN 2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

#define TFT_CS        10
#define TFT_RST        9
#define TFT_DC         8

 Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);

#define time_text_position_x 30
#define time_text_position_y 2

#define date_text_position_x 0
#define date_text_position_y 10

#define temperature_text_position_x 0
#define temperature_text_position_y 18

#define humidity_text_position_x 0
#define humidity_text_position_y 26

#define font_width_pixels 5
#define font_height_pixels 7

 unsigned int seconds = 55, minutes = 59, hours = 23;
 unsigned int day = 30, month = 12, year = 1000;

 unsigned int lastSecond = 61;
 char secondString[] = "00";
 char minuteString[] = "00";
 char hourString[] = "00";
 char dayString[] = "00";
 char monthString[] = "00";
 char yearString[] = "0000";

 char timeUpdateMask = 0;
 bool dateStringUpdate = 0;
 char time[9];
 char date[11];

 // Temperature and humidity variables
 float currentTemperature, pastTemperature;
 float currentHumidity, pastHumidity;

 // Command from Serial Monitor
 String command = "";
 boolean commandComplete = false;

 /// Control buttons
 const int setupPin = 3;
 int isInSetupState = 0;

 const int incButtonPin1 = 4;
 const int incButtonPin2 = 5;

 const int buttonPin[2] = {incButtonPin1, incButtonPin2};
 int buttonState[2];
 int lastButtonState[2] = {LOW, LOW};
 unsigned long lastDebounceTime[2] = {0, 0};
 unsigned long debounceDelay = 60;

 void setup() {
   Serial.begin(9600);

   command.reserve(200);

   Timer1.initialize(1000000);
   Timer1.attachInterrupt(incrementSeconds);

   /// Attach interrupt to the setup pin
   attachInterrupt(digitalPinToInterrupt(setupPin), changeSetupState, RISING);

   pinMode(incButtonPin1, INPUT);
   pinMode(incButtonPin2, INPUT);

   // Init the TFT display
   tft.initR(INITR_GREENTAB);
   tft.fillScreen(ST77XX_BLACK);

   dht.begin();

   intToString(hours, hourString, 2);
   intToString(minutes, minuteString, 2);
   intToString(seconds, secondString, 2);

   intToString(day, dayString, 2);
   intToString(month, monthString, 2);
   intToString(year, yearString, 4);

   sprintf(time, "%s:%s:%s", hourString, minuteString, secondString);
   sprintf(date, "%s/%s/%s", dayString, monthString, yearString);
 }

 void loop() {
   if (commandComplete) { // Check if command received
     char char_command[200];
     command.toCharArray(char_command, command.length() + 1);

     executeCommand(char_command);


     command = "";
     commandComplete = false;
   }
   else {
     if (!isInSetupState)
     {
       if (lastSecond != seconds) {
         displayTimeAndDate();
         displayTemperature();
         displayHumidity();

         lastSecond = seconds;
       }
     }
     else {
       debounce(0);
       debounce(1);
     }
   }
 }

 void serialEvent() {
   while (Serial.available()) {
     char inChar = (char)Serial.read();

     if (inChar != '\n')
       command += inChar;

     if (inChar == '\n') {
       commandComplete = true;
     }
   }
 }

 void executeCommand(char commandString[]) {
   char* pch;
   pch = strtok(commandString, " ");

   int counter = 0;

   char command[20];
   char parameter[20];

   // extract the command and the parameter strings from the received command
   while (pch != NULL)
   {
     if (counter == 0) {
       strcpy(command, pch);
       counter++;
     }
     else {
       strcpy(parameter, pch);
     }
     pch = strtok (NULL, " ");
   }

   if (strcmp(command, "setDay") == 0)
   {
     int newDay = atoi(parameter);
     if (newDay >= 1 && newDay <= 31)
     {
       strcpy(dayString, parameter);
       day = newDay;
     }
   }
   else if (strcmp(command, "setMonth") == 0)
   {
     int newMonth = atoi(parameter);
     if (newMonth >= 1 && newMonth <= 12)
     {
       strcpy(monthString, parameter);
       month = newMonth;
     }
   }
   else if (strcmp(command, "setYear") == 0)
   {
     int newYear = atoi(parameter);
     if (newYear >= 1000 && newYear <= 9999)
     {
       strcpy(yearString, parameter);
       year = newYear;
     }
   }

   clearCharPosition(date_text_position_x, date_text_position_y, 128,
     font_height_pixels);
   sprintf(date, "%s/%s/%s", dayString, monthString, yearString);
   tft.setCursor(date_text_position_x, date_text_position_y);
   tft.print(date);
 }

 void debounce(int buttonNumber) {
   int reading = digitalRead(buttonPin[buttonNumber]);

   if (reading != lastButtonState[buttonNumber]) {
     lastDebounceTime[buttonNumber] = millis();
   }

   if ((millis() - lastDebounceTime[buttonNumber]) > debounceDelay) {
     if (reading != buttonState[buttonNumber]) {
       buttonState[buttonNumber] = reading;

       if (buttonState[buttonNumber] == HIGH) {
         buttonPressed(buttonNumber);
       }
     }
   }
   lastButtonState[buttonNumber] = reading;
 }

 void buttonPressed(int buttonNumber) {
   if (buttonNumber == 0) { // Increase minutes
     minutes = (minutes + 1) % 60;

     intToString(minutes, minuteString, 2);
   } else if (buttonNumber = 1) { // Increase hour
     hours = (hours + 1) % 24;

     intToString(hours, hourString, 2);
   }

   // Print the new time on the display
   sprintf(time, "%s:%s:%s", hourString, minuteString, secondString);
   clearCharPosition(0, time_text_position_y, 128, font_height_pixels);
   tft.setCursor(time_text_position_x, time_text_position_y);

   tft.print(time);
 }

 void incrementSeconds() {
   timeUpdateMask = 0;
   dateStringUpdate = 0;

   seconds += 1;
   timeUpdateMask = timeUpdateMask | 1;

   if (seconds >= 60) {
     minutes++;
     timeUpdateMask = timeUpdateMask | 2;

     seconds = 0;

     if (minutes >= 60)
     {
       hours++;
       timeUpdateMask = timeUpdateMask | 4;

       minutes = 0;
       intToString(hours, hourString, 2);

       if (hours >= 24) {
         hours = 0;
         day++;

         if (day >= 31) {
           day = 1;
           month++;

           if (month >= 13)
           {
             month = 1;
             year++;

             intToString(year, yearString, 4);
             dateStringUpdate = 1;
           }

           intToString(month, monthString, 2);
           dateStringUpdate = 1;
         }

         intToString(day, dayString, 2);
         dateStringUpdate = 1;
       }
       intToString(hours, hourString, 2);
     }
     intToString(minutes, minuteString, 2);
   }
   intToString(seconds, secondString, 2);

   sprintf(time, "%s:%s:%s", hourString, minuteString, secondString);

   if (dateStringUpdate) {
     intToString(day, dayString, 2);
     intToString(month, monthString, 2);
     intToString(year, yearString, 4);

     sprintf(date, "%s/%s/%s", dayString, monthString, yearString);
   }
 }

 void changeSetupState() {
   isInSetupState = (isInSetupState + 1) % 2;

   // Reset the seconds
   seconds = -1;

   if (isInSetupState == 1)
   {
     Timer1.stop();
   }
   else {
     Timer1.restart();
   }
 }


 // DISPLAY FUNCTIONS
 // -----------------
 void displayTimeAndDate() {
   // Clear the seconds position
   if (timeUpdateMask & 1)
   {
     clearCharPosition(time_text_position_x + 42, time_text_position_y, font_width_pixels, font_height_pixels);
     clearCharPosition(time_text_position_x + 36, time_text_position_y, font_width_pixels, font_height_pixels);
   }

   // Clear the minutes position
   if (timeUpdateMask & 2)
   {
     clearCharPosition(time_text_position_x + 24, time_text_position_y, font_width_pixels, font_height_pixels);
     clearCharPosition(time_text_position_x + 18, time_text_position_y, font_width_pixels, font_height_pixels);
   }

   // Clear the hour position
   if (timeUpdateMask & 4)
   {
     clearCharPosition(time_text_position_x + 6, time_text_position_y, font_width_pixels, font_height_pixels);
     clearCharPosition(time_text_position_x, time_text_position_y, font_width_pixels, font_height_pixels);
   }

   if (dateStringUpdate) {
     clearCharPosition(date_text_position_x, date_text_position_y, 128, font_height_pixels);
   }

   tft.setCursor(time_text_position_x, time_text_position_y);
   tft.print(time);

   tft.setCursor(date_text_position_x, date_text_position_y);
   tft.print(date);
 }

 void displayTemperature() {
   currentTemperature = dht.readTemperature();

   // Update the temperature text line
   if (currentTemperature != pastTemperature) {
     pastTemperature = currentTemperature;

     // Clear the entire temperature line
     clearCharPosition(0, temperature_text_position_y, 128, font_height_pixels);

     tft.setCursor(temperature_text_position_x, temperature_text_position_y);
     tft.print(currentTemperature);
     tft.print(" C");
   }
 }

 void displayHumidity() {
   currentHumidity = dht.readHumidity();

   // Update the humidity text line
   if (currentHumidity != pastHumidity)
   {
     pastHumidity = currentHumidity;

     // Clear the entire humidity line
     clearCharPosition(0, humidity_text_position_y, 128, font_height_pixels);

     tft.setCursor(humidity_text_position_x, humidity_text_position_y);
     tft.print(currentHumidity);
     tft.print("%");
   }
 }
 // ----------------------


 // UTILITY FUNCTIONS
 // ----------------------

 /// Convert int to string, starting from right to left
 /// and pad with '0' untile end
 char* intToString(int nr, char retString[], int stringSize)
 {
   int index = stringSize - 1;
   while (nr) {
     retString[index] = '0' + (nr % 10);
     nr = nr / 10;
     index--;
   }

   while (index >= 0) {
     retString[index] = '0';
     index--;
   }

   retString[stringSize] = '\0';
 }

 void clearCharPosition(int x0, int y0, int width, int height)
 {
   tft.fillRect(x0, y0, width, height, ST77XX_BLACK);
 }

 // --------------------
```