Real-time clock-based data logging subsystem in a 5 DOF robotic arm system

Tál Dominik
Neptun code: RHNFAE
Lecturer: Dr. Magyar Attila

Table of contents

Task description

Real-time clock-based data logging subsystem in a 5 DOF robotic arm system

The student shall develop a data logging subsystem based on an Arbotix-M robocontroller, a 5 degrees of freedom robotic arm, a micro SD card shield and a real time clock module.
The newly implemented subsystem should be able to log all the parameters of the robotic arm in real time, with at least a 100 ms resolution. The created log files’ format should be chosen with a focus on simple data analysis. The data structure should be designed by the student after negotiation with the lecturer. Detailed testing documentation should be included too in the semester closing documentation.
Required activities during the semester:

- choose the proper hardware for being able to determine the actual time and date
- choose the proper hardware for interfacing a micro SDHC card
- develop an interface to the Arbotix-M Robocontroller to be able to send out the internal values stored in its RAM
- design the data structure of the log files
- implement the secure data logging to the micro SDHC card
- execute and document the testing procedure

Introduction

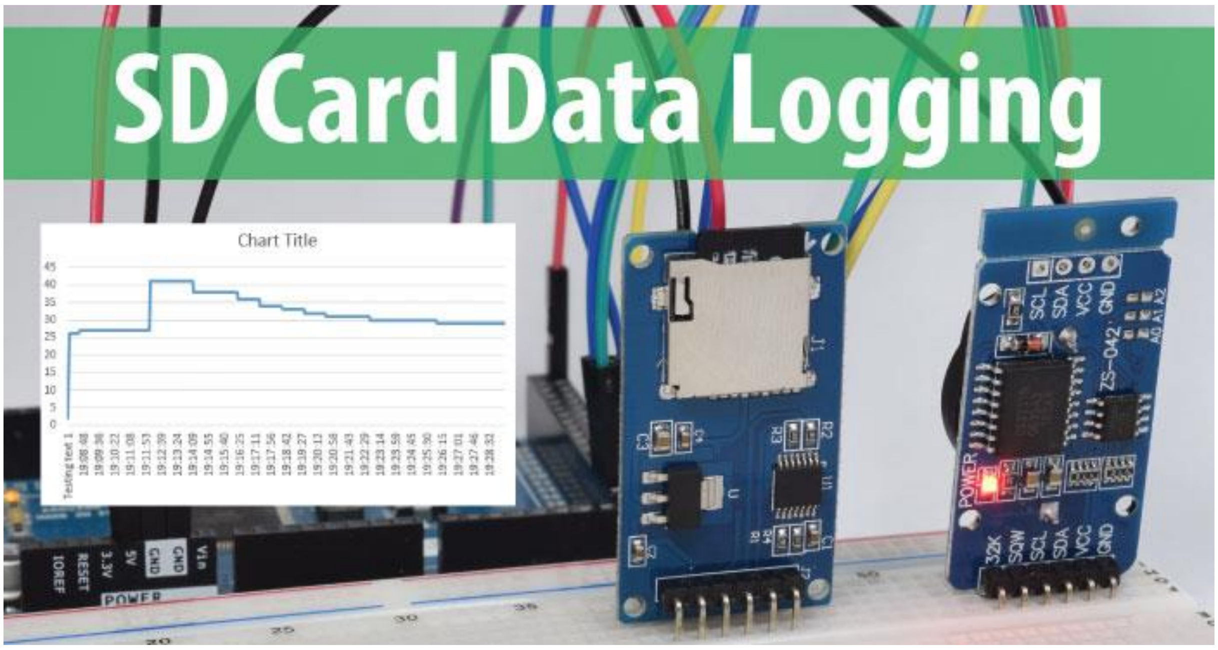
Nowadays in industrial systems data logs are one of the biggest treasure what mankind can gather to be able to develop processes and make them more effective. In the age of IoT (Internet of Things) we are blind without these tiny but really useful details which can be gathered from our systems. Just please imagine how terribly much data could be collected from our microcontroller based systems. Obviously, some percent of them still have data logging, but what about the remaining part? This gap must be decreased and according to my vision it will reach zero.

Data logging in real-time systems

The ability of the data logger system is that it can collect data in real-time after the logging subsystem is activated, then it will log all the information needed during the monitoring period.

"Data logging is the process of collecting and recording data from sensor output automatically which aims for archiving or data analysis

Data logging can be used in research monitoring systems that require fast and precise data collection time. Data collected is data from sensors that will convert physical quantities into electrical signals that can be measured automatically and sent to computer devices or microprocessors for processing. To support fast and precise data collection, data logging requires an important element, the data logger. By using a data logger, users can monitor or do data logging for 24 hours continuously and real-time, this allows users to get comprehensive information about the condition of something being monitored. In addition, the data that has been recorded will be stored in offline storage or even online, so that the data can be monitored and analysed easily and efficiently"



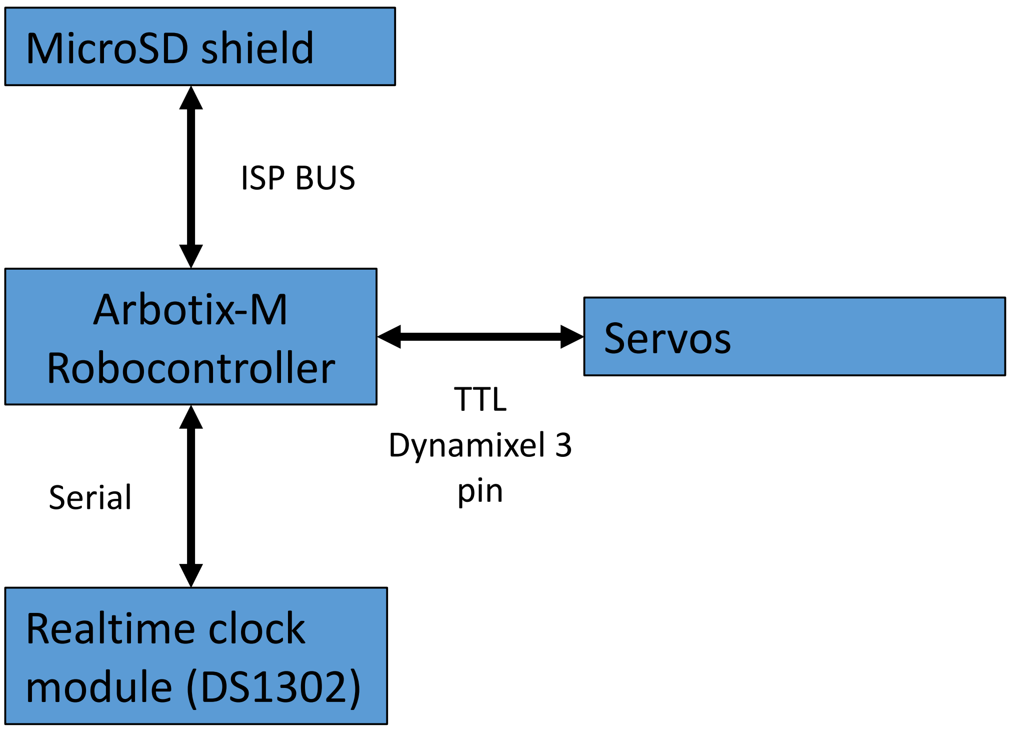
While you are designing a data logging system you have to take care about all the tiny details of the system. Basic rule is that your logging system must not influence the inspected system's behaviour. I mean it must not modify some internal values, runtime should not be increased radically and stability has to be kept while integrating a new module which will log internal or external data in the system.

Let's go in some details regarding the runtime. Arduino system can provide pretty good loop time, but it can be radically increased after applying some resource required processes. Data logging is not that activity which can cause this problem if you do not want to save the Arduino's whole RAM in each and every millisecond of runtime.

System and software requirements

In this chapter I will introduce the schematic of the system plan, the system level requirements and the software level requirements.
It is important to divide these two groups of requirements as later the two layers can be modified independently.

System plan



System requirements

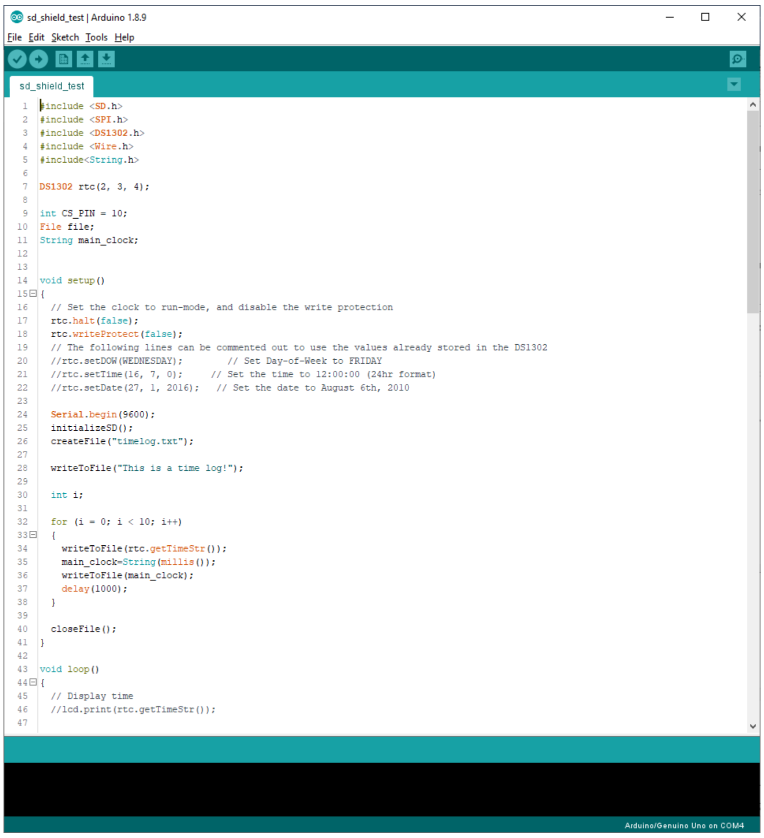
1. The time and date values shall be provided by a DS1302 timekeeping chip.

a.

2. The micro SD card standard shall be used as a data storage.
3. The micro SD card shall be interfaced by an LVC125A chip
4. The micro SD card shield should use an SPI bus for communication with the microcontroller
5. All the connector wires must be insulated by PVC
6. The microcontroller must be placed on an insulator which can help avoiding short circuit.
7. The extension shields must be placed on an insulator which can help avoiding short circuit.
8. The micro SD card shall provide at least 2 GiB of free storage place
9. The file system shall be chosen to provide compatibility on most operating systems
10. The file format shall be .csv
11. The file name shall follow the given pattern: <YYYYMMDD>;<HHWWSS>.csv
    a. where YYYY equals to the year in 4 digits
    b. where MM equals to month in 2 digits
    c. where DD equals to the day in 2 digits
    d. HH equals to the hour in 2 digits
    e. WW equals to the minutes in 2 digits
    f. SS equals to the seconds in two digits
12. The log file shall be automatically created at system start up.
13. The system has to ensure data protection for unwanted power off situations
14. The log file shall be closed securely when the robotic arm has finished all of its tasks.
15. A new data line has to be added in every 50 milliseconds.
16. The data logging process should not interrupt a request for the robotic arm.

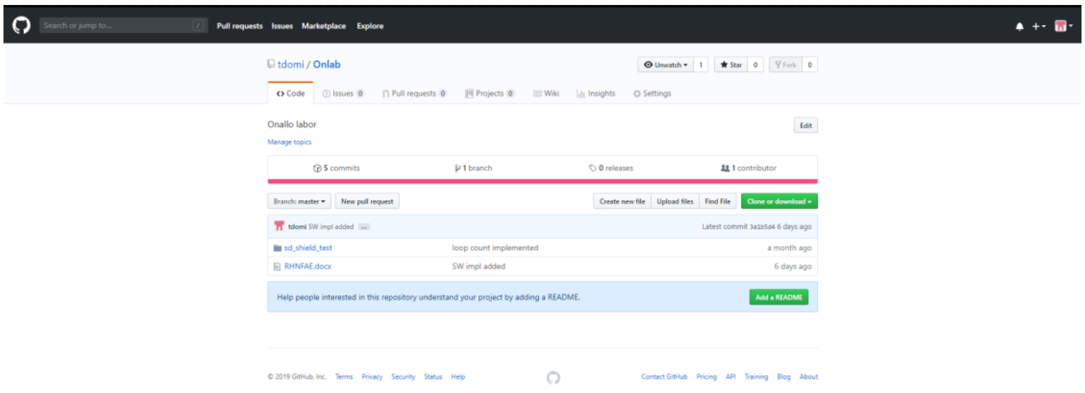
Software requirements

1. Arduino IDE should be used for development.

a. 
sd_shield_test | Arduino 1.8.9
File Edit Sketch Tools Help
sd_shield_test
Arduino/Genuino Uno on COM4

```
#include <SD.h>
#include <SPI.h>
#include <DS1302.h>
#include <Wire.h>
#include<String.h>

DS1302 rtc(2, 3, 4);

int CS_PIN = 10;
File file;
String main_clock;


void setup()
{
  // Set the clock to run-mode, and disable the write protection
  rtc.halt(false);
  rtc.writeProtect(false);
  // The following lines can be commented out to use the values already stored in the DS1302
  //rtc.setDOW(WEDNESDAY);        // Set Day-of-Week to FRIDAY
  //rtc.setTime(16, 7, 0);     // Set the time to 12:00:00 (24hr format)
  //rtc.setDate(27, 1, 2016);   // Set the date to August 6th, 2010

  Serial.begin(9600);
  initializeSD();
  createFile("timelog.txt");

  writeToFile("This is a time log!");

  int i;

  for (i = 0; i < 10; i++)
  {
    writeToFile(rtc.getTimeStr());
    main_clock=String(millis());
    writeToFile(main_clock);
    delay(1000);
  }

  closeFile();
}

void loop()
{
  // Display time
  //lcd.print(rtc.getTimeStr());

```

2. Every software component should be included in different header files.
3. The SPI.h header shall be used for the SPI bus communication which isp provided by the Arduino default libraries
4. The DS1302.h shall be used for the real time clock module for interfacing the hardware securely.
5. The SD.h directory shall be developed by the student which shall contain the following functions
    a. CreateFile
    b. WriteToFile
    c. CloseFile
    d. OpenFile
    e. Readline
6. GitHub shall be used as a version control system.



tdomi / Onlab

Onallo labor

tdomi SW impl added — Latest commit 3a1b5d4 6 days ago

sd_shield_test	loop count implemented	a month ago
RHNFAE.docx	SW impl added	6 days ago

Help people interested in this repository understand your project by adding a README.

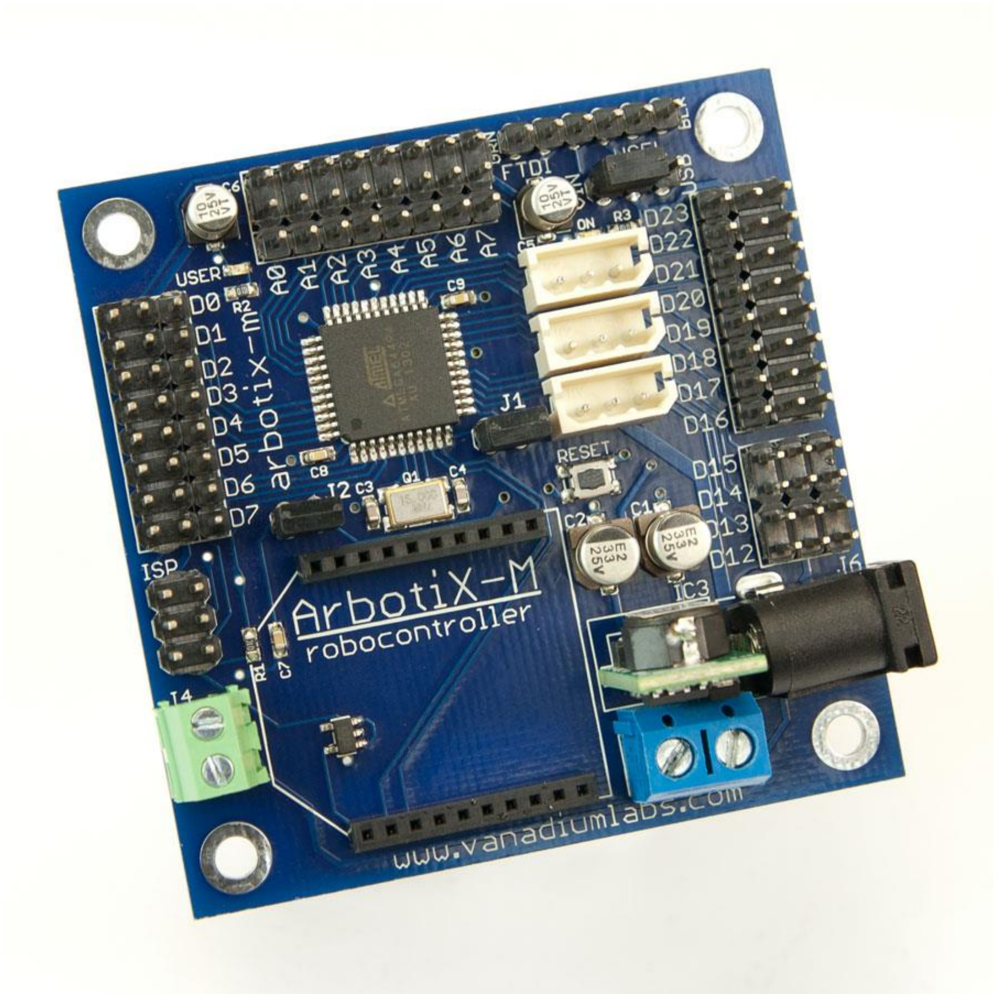
a.

7. An exception shall be raised and handled properly while the real time clock module is disconnected while the system is running.
8. An exception shall be raised and handled properly while the micro SD card shield's data wires are disconnected from the Arduino microcontroller.
9. The successful file opening shall be communicated on the default serial port for debugging purposes.
10. The unsuccessful file opening shall be communicated on the default serial port for debugging purposes.

Implementation

Hardware implementation

Luckily the Arbotix-M Robocontroller has exactly the same input and output pins as and Arduino Uno microcontroller. That makes the whole wiring process much easier. The pin layout is really user-friendly too, so everything can be wired in easily and safely.



Pins used by the DS1302 module (left hand side is the microcontroller side):

- Digital pin 2: RST
- Digital pin 3: DAT
- Digital pin 4: CLK

Pins used by the microSD card shield:

- Digital pin 10: CS
- Digital pin 11: MOSI
- Digital pin 12: MISO

- Digital pin 13: SCK

Of course both devices need a parallel line from the VCC (+5V) and the GND to operate.

File system and data structure

The used interface can handle complex folder structures, but representing this is not the aim of my project. So I just put a file in the root folder. I have chosen .csv as it is simple, can be automatically analysed with many tools, and Microsoft Excel supports it too. The filename is automatically generated from the RTC module's date and time data. The secure file handling is provided by the implemented file opening and closing functions.

I have faced a challenging technical problem while I started designing the data structure. The DS1302 real time clock module can only output seconds as the biggest time resolution. Therefore I needed a more precise time data too. I have discovered in the Arduino documentation that the microcontroller is counting the elapsed milliseconds since the last restart. This can be requested by the call of Millis() function.
After solving the timestamp issue I could start defining which data is necessary from the robotic arm. First of all, the system voltage should be continuously monitored. Then the encoder values of each servos. Additionally the 3D coordinates can be stored using the calculations presented in the attached project laboratory documentation.

Software implementation

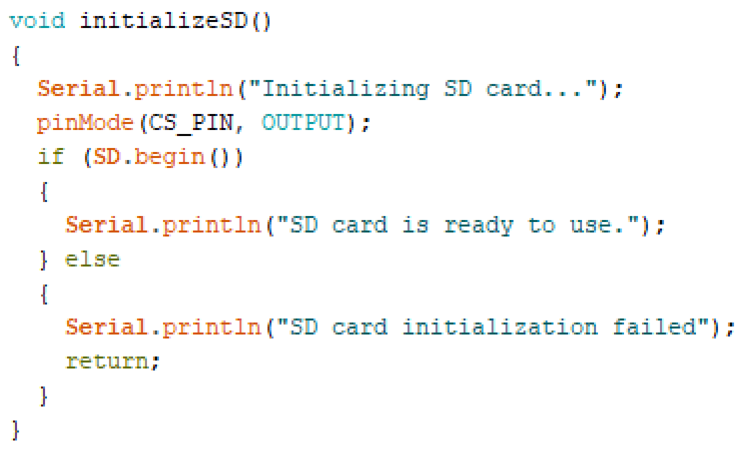
The software implementation has been made driven on the component based development. I have created a separate header file for the real time clock module components, and another for the SD card shield.
Firstly, let's see the basic functions of the RTC.h header:

- void setTimeAndDate ();
- void getTimeAndDate();
- void printTimeStamp();

Here the setter function is used for adjusting the date and the time and the week of the day in the RTC module memory. The getter of course handles getting the actual date and time, and there is a separate printer function because the transformation of the format is being done in this component, the SD card shield's software module will only handle the prepared data.

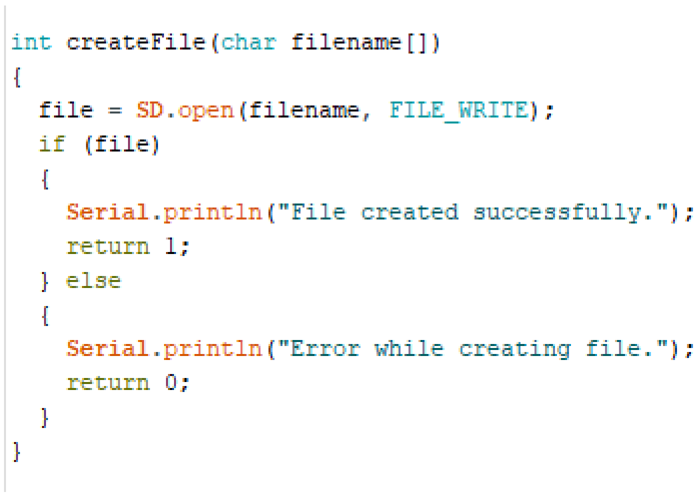
The software module for the SD card shield is created in a separate header file too, and it has the following functions which are the high level implementation of the requirements based on the lower level functions provided by the manufacturer.

- void initializeSD();



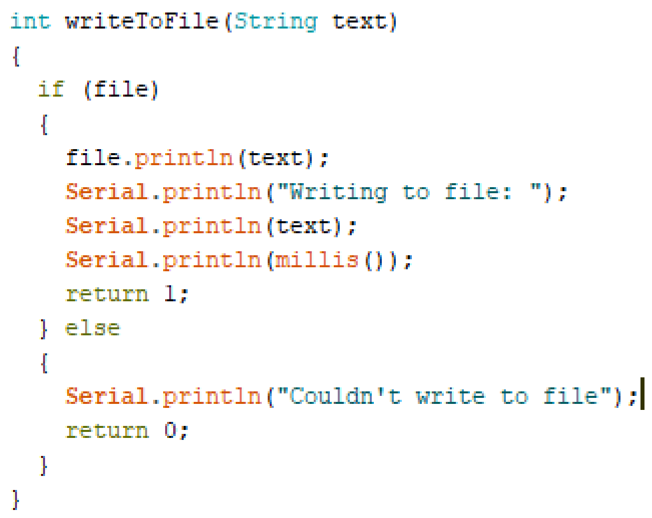
```
void initializeSD()
{
  Serial.println("Initializing SD card...");
  pinMode(CS_PIN, OUTPUT);
  if (SD.begin())
  {
    Serial.println("SD card is ready to use.");
  } else
  {
    Serial.println("SD card initialization failed");
    return;
  }
}
```

- int createFile(char filename[])



```
int createFile(char filename[])
{
  file = SD.open(filename, FILE_WRITE);
  if (file)
  {
    Serial.println("File created successfully.");
    return 1;
  } else
  {
    Serial.println("Error while creating file.");
    return 0;
  }
}
```

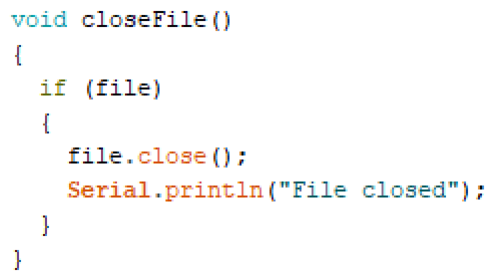
- int writeToFile(String text)



```
int writeToFile(String text)
{
  if (file)
  {
    file.println(text);
    Serial.println("Writing to file: ");
    Serial.println(text);
    Serial.println(millis());
    return 1;
  } else
  {
    Serial.println("Couldn't write to file");
    return 0;
  }
}
```

  -

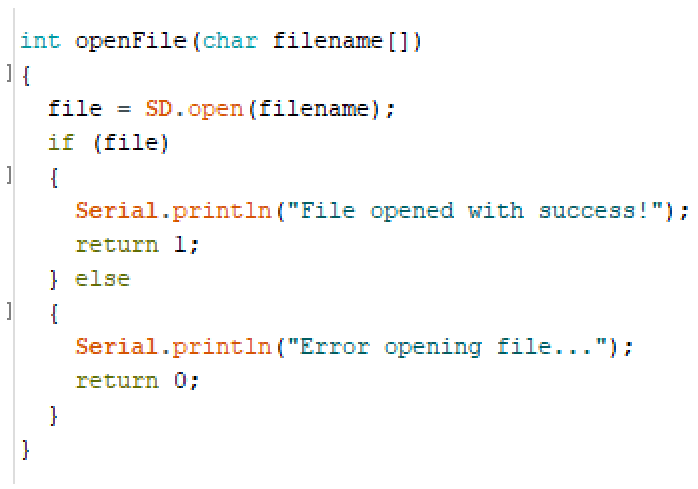
- void closeFile()



```
void closeFile()
{
  if (file)
  {
    file.close();
    Serial.println("File closed");
  }
}
```

  -

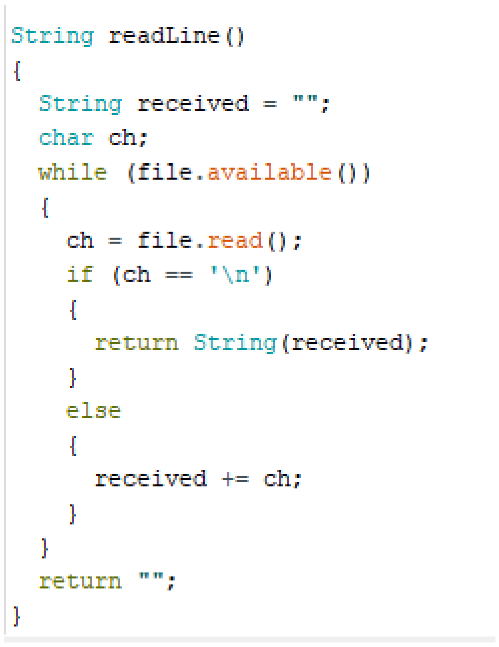
- int openFile(char filename[])



```
int openFile(char filename[])
{
  file = SD.open(filename);
  if (file)
  {
    Serial.println("File opened with success!");
    return 1;
  } else
  {
    Serial.println("Error opening file...");
    return 0;
  }
}
```

  -

- String readLine()



```
String readLine()
{
  String received = "";
  char ch;
  while (file.available())
  {
    ch = file.read();
    if (ch == '\n')
    {
      return String(received);
    }
    else
    {
      received += ch;
    }
  }
  return "";
}
```

After introducing the function prototypes let’s see their purpose and working. The initialization function performs a secure initialization for the file handling. The create file creates a new file in the set directory with the given name. The writer is the interface which can write the string in the parameter to the opened file. The closing function is necessary for the secure file handling after the writing activities. Open file is responsible for a previously crated file’s secure opening and positioning to its end to append it. Readline is used for read the next line from the actually opened file.

Testing

In my project every test is executed against the previously defined requirements and the actual software version. Functional requirements can be tested directly by test cases written based on the requirements, non-functional requirements can be proven with the actual implementation and standards.

System test

Detailed test results will be provided in the following semester's documentation for the Thesis Laboratory subject.

Software test

Detailed test results will be provided in the following semester's documentation for the Thesis Laboratory subject.

Summary

In general, this semester was successful from that point of view that I knew two very useful devices for Arduino systems, their hardware specification and of course their software. I have implemented some new features based on the given libraries' basic functions, and I have designed a data structure for logging. The testing and its documentation did not reach the maturity which I have planned, but without 100% test coverage the project can still run and work properly in real life, not just in the plans.

Future plans

This project has been planned and executed as a crucial part of my thesis written in the following two semesters. As the logging subsystem has been integrated successfully in the system, now I can go on developing a so called learning function in the software. That means after a specific command the robotic arm will be released and can be moved freely. The SD module will log the movement and it will be able to replay these predefined movements. This plan has many challenges too, I will have to define the speed of replay, or maybe make it depend on a parameter. A filtering will be needed too in the recorded data.
As this system from hardware point of view is complete and robust I am not planning to modify it, only the software will be radically changed.

With the upper mentioned learning function my project should support our life in many fields. The robotic arm's structure is the same as the human hands, so it could memorize physiotherapic exercises to help the rehabilitation of the injured. Another interesting field would be recording movements and replay them million times in industrial fields. Of yourse, you can not move a huge robotic arm on your own, but my project is scalable up to the mid size robotic arms which are the most common in co-robot workplaces and processes.

Bibliography

Dr. Magyar Attila. (2017). Robotics (VEMIVIB112R/2017/18/1) learning materials.

Ervianto, N. L. (2012). Data Logger Sensor Suhu Berbasis Mikrokontroler ATmega 8535 dengan PC sebagai Tampilan. J. Ilm. Elit. Elektro.

Margolis, M. (2011). Arduino Cookbook, 2nd Edition. O'Reilly Media.

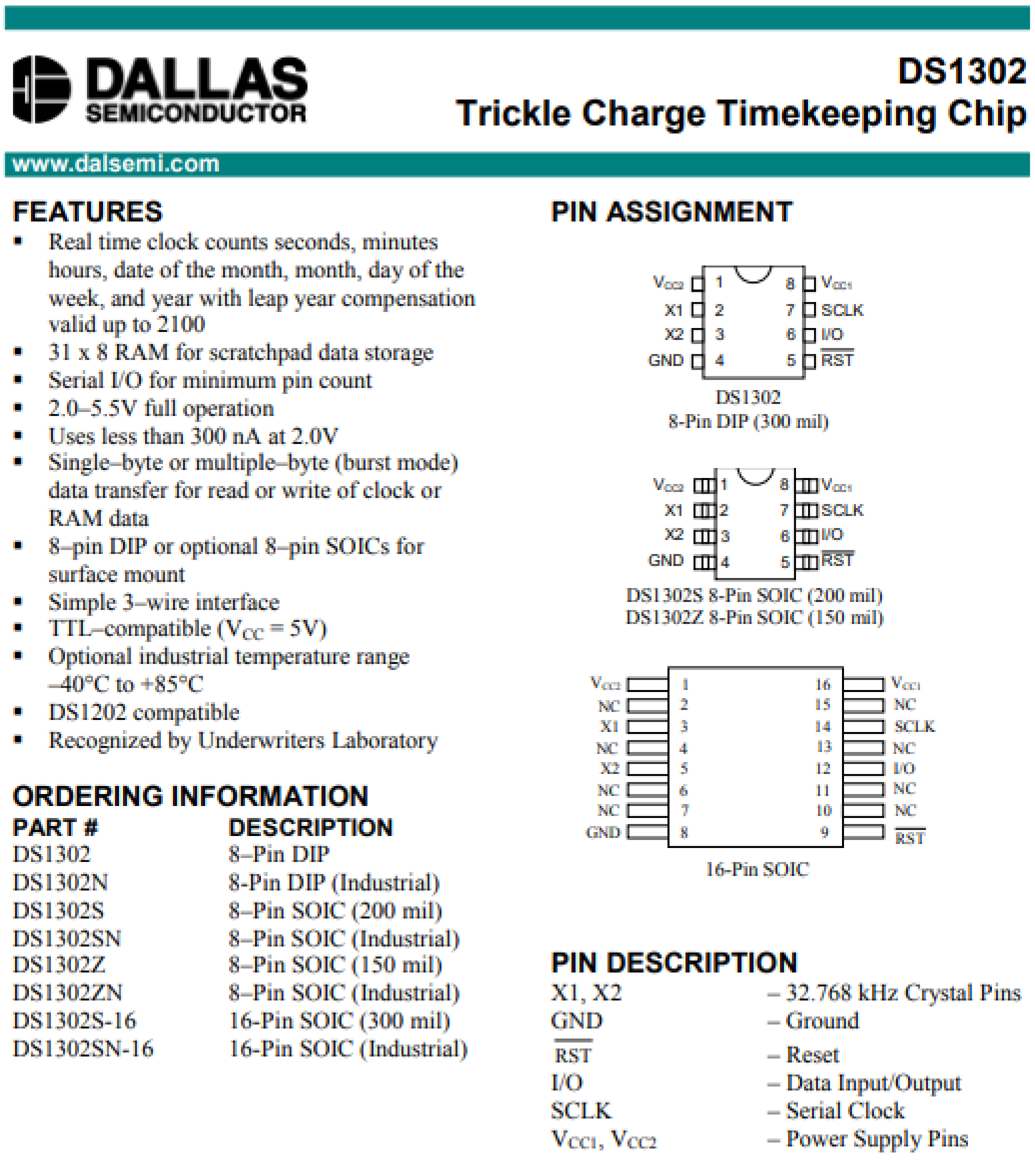
Niku, S. B. (2011). Introduction to Robotics: Analysis, Control, Applications. San Luis: Pearson Education Inc.

Arduino. (2017). Source: Wikipedia: https://en.wikipedia.org/wiki/Arduino

Bräunl, T. (2003). Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems. Springer Verlag: Berlin.

Appendix

DS1302 datasheet (extract)



DALLAS SEMICONDUCTOR

DS1302
Trickle Charge Timekeeping Chip

www.dalsemi.com

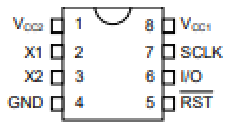
FEATURES

- Real time clock counts seconds, minutes hours, date of the month, month, day of the week, and year with leap year compensation valid up to 2100
- 31 x 8 RAM for scratchpad data storage
- Serial I/O for minimum pin count
- 2.0–5.5V full operation
- Uses less than 300 nA at 2.0V
- Single–byte or multiple–byte (burst mode) data transfer for read or write of clock or RAM data
- 8–pin DIP or optional 8–pin SOICs for surface mount
- Simple 3–wire interface
- TTL–compatible (V_{CC} = 5V)
- Optional industrial temperature range –40°C to +85°C
- DS1202 compatible
- Recognized by Underwriters Laboratory

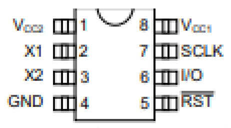
ORDERING INFORMATION

PART #	**DESCRIPTION**
DS1302	8–Pin DIP
DS1302N	8-Pin DIP (Industrial)
DS1302S	8–Pin SOIC (200 mil)
DS1302SN	8–Pin SOIC (Industrial)
DS1302Z	8–Pin SOIC (150 mil)
DS1302ZN	8–Pin SOIC (Industrial)
DS1302S-16	16-Pin SOIC (300 mil)
DS1302SN-16	16-Pin SOIC (Industrial)

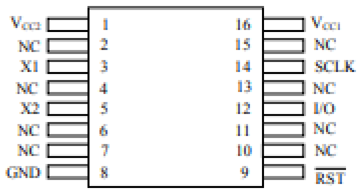
PIN ASSIGNMENT



DS1302
8-Pin DIP (300 mil)



DS1302S 8-Pin SOIC (200 mil)
DS1302Z 8-Pin SOIC (150 mil)



16-Pin SOIC

PIN DESCRIPTION

X1, X2	– 32.768 kHz Crystal Pins
GND	– Ground
$\overline{\text{RST}}$	– Reset
I/O	– Data Input/Output
SCLK	– Serial Clock
V_{CC1}, V_{CC2}	– Power Supply Pins

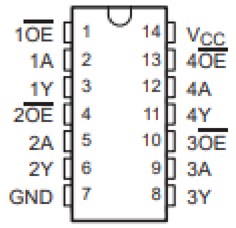
LVC125A datasheet (extract)

SN74LVC125A QUADRUPLE BUS BUFFER GATE WITH 3-STATE OUTPUTS

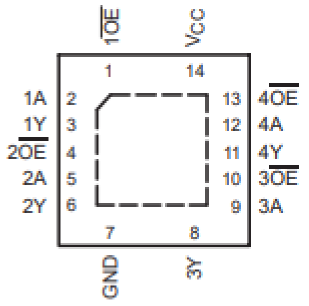
SCAS290N – JANUARY 1993 – REVISED FEBRUARY 2004

- **Operates From 1.65 V to 3.6 V**
- **Specified From −40°C to 85°C and −40°C to 125°C**
- **Inputs Accept Voltages to 5.5 V**
- **Max t_{pd} of 4.8 ns at 3.3 V**
- **Typical V_{OLP} (Output Ground Bounce) <0.8 V at V_{CC} = 3.3 V, T_A = 25°C**
- **Typical V_{OHV} (Output V_{OH} Undershoot) >2 V at V_{CC} = 3.3 V, T_A = 25°C**
- **Latch-Up Performance Exceeds 250 mA Per JESD 17**
- **ESD Protection Exceeds JESD 22**
  - **2000-V Human-Body Model (A114-A)**
  - **200-V Machine Model (A115-A)**
  - **1000-V Charged-Device Model (C101)**

D, DB, NS, OR PW PACKAGE
(TOP VIEW)



RGY PACKAGE
(TOP VIEW)



description/ordering information

This quadruple bus buffer gate is designed for 1.65-V to 3.6-V V_{CC} operation.

The SN74LVC125A features independent line drivers with 3-state outputs. Each output is disabled when the associated output-enable ($\overline{OE}$) input is high.

To ensure the high-impedance state during power up or power down, $\overline{OE}$ should be tied to V_{CC} through a pullup resistor; the minimum value of the resistor is determined by the current-sinking capability of the driver.

Inputs can be driven from either 3.3-V or 5-V devices. This feature allows the use of this device as a translator in a mixed 3.3-V/5-V system environment.