Development and testing of an Arduino data logger to record four temperature readings, Voltage, Current and Power

A report by:

Cecilia N. Naule

Master of Science in Renewable Energy

(University of Namibia (UNAM), Department of Physics, Chemistry & Material Science, Namibia)

Completed under the supervision of:

Prof. Ole Jorgen Nydal

At

(Norwegian University of Science and Technology (NTNU), Department of Energy and Process Engineering, Norway)

July 2024

Acknowledgements

Firstly, I would like to extend my sincere gratitude to **SANORD** for supporting my research visit to NTNU through the **Brian O'Connell Scholarship**. This important part of my research would not be achievable if it was not for this scholarship.

I would further like to express my acknowledgement to my supervisor at NTNU, **Prof. Ole Jorgen Nydal**, for providing me with all the necessary tools and components to enable me to come up with a functional data logger. His passion to explain as much as he could helped me understand the fundamental concepts concerning Arduino data loggers, especially those that deal with temperature, current, voltage, and power.

A special thanks to **Jimmy Chasiga** from Makerere University, Uganda, for helping me with my data logger during the initial stages of its construction and for being my first soldering instructor 😊. My first week at NTNU was a lot smoother thanks to you.

I would also like to extend my appreciation to my supervisor at UNAM, **Dr. Chiguvare Zivayi,** for his constant support, encouragement and faith in me.

Last but not least, I would like to express my appreciation to **Tsige Gebregergs** from Mekele University, Ethiopia, for helping me during the start of my data logging journey. Your assistance was greatly appreciated.

Table of Contents

List of figures

List of tables

1. Introduction and background

Temperature is one of the top five parameters measured in the world annually, which contributes to the advancement of its measuring devices yearly for minute purposes to large scaled purposes [1]. In modern world, practically there are no systems in which some kind of temperature monitoring is not needed, while the measurement accuracy offered by today's most advanced temperature data loggers rivals the performance of many higher priced, computer-based data acquisition systems [2]. Moreover, the need for collecting high quality data exponentially increases, as better information on performance can improve understanding dynamics of system energy use, thermal comfort, indoor environmental quality, microbiology of the built environment, etc.

While modern management systems can gather extensive data on system operations, precise characterizations of several parameters sometimes rely on proprietary hardware/software, which negatively impacts costs, flexibility, and data integration in decision-making and control [2]. Measurement in the engineering sector is rigorously governed by the precision and efficacy of measuring instruments. This further influences the elevated selling price of measuring devices (sensors), data collecting systems, or data loggers manufactured by measurement equipment producers. Due to their elevated cost, numerous laboratories in educational institutions lack the instrumentation for these measurement devices [3].

A Data Logger is a programmed electronic device that gives room for measurement, documentation, analysis and authentication of various parameters like (voltage, current, humidity, temperature and pH) over a period with desirable time intervals. The basic requirement for a data logging system is acquisition, online analysis, logging, offline analysis, display and data sharing [1]. Data loggers accept data inputs from sensors implemented in the circuit depending on the target purpose and the parameter being measured. Common examples of such input sensors are temperature sensors, sound sensors, pressure sensors, flame sensors, light sensors, electrical sensors, contact sensors, water sensors and flames sensors [1].

The main components of a data logger are a microcontroller (which could be an Arduino, Universal Synchronous/Asynchronous Receiver/Transmitter (USART); Microprocessor; Peripheral Interface Controller (PIC); Integrated Electronics (INTEL) 8051, 8052, AT89s52); a sensor to capture values; a Serial Data (SD) (which serves as an internal memory for storing data) and a power supply mechanism (i.e., battery powered, Universal Serial Bus (USB) and Alternate current supply) [4, 1, 5, 6, 7]. It works with the sensors to convert physical phenomena into electronic signals. Then these electronic signals are converted into binary data, which is easily analyzed by software and stored for later process analysis [5].

The major consequence of data loggers over some measurement device is their ability to capture data automatically within a specified period depending on its source of power [1]. This feature minimizes human efforts in monitoring and errors in recording and documentation of values. There exist different types of data loggers, which include Wi-Fi data logger, universal input data logger, Bluetooth data logger, remote data logger, Radio Frequency Identification data logger, Modbus data logger, High speed data logger, multi-channel data logger, paperless data logger, mechanical and electrical data logger [1].

A wide range of factors could affect the choice of a data logger ranging from reliability, cost, usability, timeliness and high data accuracy, efficiency, alarm indication of preferred value limit, ability to withstand high temperature if used in hot regions and available storage space among others [1, 8]. Being mobile because of their small size is one of the advantages of a datalogger system. Another advantage is the feature of automatically collecting data without human surveillance for a long time [1, 6]. Datalogger systems are designed according to the needs of the specific environments or applications. In addition, they can be used in remote areas or dangerous situations. They are more accurate because there is no possibility of human error when recording. With the help of graphics obtained from their records, they help to better understand scientific experiments and scientific concepts [6].

Although they have numerous advantages, dataloggers have some disadvantages. They are expensive and their initial investment costs are high for small businesses [6]. Usually, they do not have all the features required by the user, so changes may be required in the software or application. As a result, some data may be lost or not saved if the data logger fails. In addition, some dataloggers can only take readings in the initially configured fixed intervals. Furthermore, basic training is required to use them [6]. It is therefore important to develop low-cost, user-customizable and re-programmable datalogger systems for specific purposes in order to record the desired parameters.

In recent years, numerous researchers have employed Arduino as a mechanical controller or for data gathering [3]. An Arduino is a physical programming platform that uses an Advance Technology for Memory and Logic (ATMEL) microcontrollers and has a variety of digital and analogue inputs and outputs [9, 6]. A microcontroller is a small computer on a single integrated circuit containing a processor, memory, and programmable input/output peripherals [4]. The Arduino platform combines electronics hardware and software into a cohesive system that is easy for novices to use in many different applications including lab and field-based research [9]. Arduino perceives the environment through input from various sensors and influences its surroundings by regulating lights, motors, and other actuators.

Examples of employing Arduino as a mechanical controller include stepper motor drives and data collecting time configurations based on frequency or duration [3]. Arduino may serve as a data acquisition system to obtain temperature measurements via thermocouples. In addition to data collecting, Arduino can function as an autonomous data logger with an extended data retrieval duration. Arduino can acquire data from 64 temperature measurement points by utilizing a multiplexer to connect several devices [3]. Another great strength of the Arduino is the cross-platform Integrated Development Environment (IDE) which presents a simplified C++ programming interface that leverages extensive code libraries without requiring the user to know low-level details for common-case implementations [9].

A data logger based on the Arduino has many features such as: Built with low-cost components compared to commercial data loggers that are usually expensive; These components can be easily obtained and purchased; Low power consumption; The adjustability of operating parameters; The ability to connect with a computer to collect and analyze data [5]. The recent development in energy sector have shown that solar-energy market is one of the most rapidly expanding renewable energy markets in the world. Presently there is significant increasing in demands for remote monitoring and control equipment for solar-energy applications [10]. The need for this current data logger project arose as a result of the fact that most temperature loggers are beyond the reach of most researchers in developing countries due to the high cost of these systems and the difficulty in accessing fund prevalent in these regions.

This project chose an Arduino data logger because of its open-source character, simplicity of use, and great community support. The modular architecture of the platform enables simple integration of other components including sensors, shields, and displays as well as its capacity to be driven with a tiny power bank in case of distant/remote places where a computer cannot run it. This adaptability makes Arduino a great alternative for bespoke data logging solutions fit for particular requirements. Furthermore, the large spectrum of accessible libraries streamlines the development process and makes fast prototyping and implementation possible. This work therefore produced a tailored and re-programmed datalogger system ready for measuring four temperature values from a heat storage energy system.

2. Objectives

The objectives of this thesis component were to:

1. Build an Arduino-uno data logger that can read four temperature values from Type-K thermocouples.
2. Modify the data logger to read current, voltage and power for when a PV system is connected.
3. Test the data logger to successfully log the temperature, current, voltage and power readings to an SD card while displaying real-time data on an LCD shield.

3. Literature review on Arduino data loggers

3.1 Introduction

In the last decade, Arduino-based data loggers have attracted much interest as they are customizable, flexible, and cheap. Such systems have been incorporated into different sectors such as environmental monitoring, solar energy systems, and agricultural applications. This literature review is focused on the key development and applications of data loggers based on Arduino boards, with the indication of the boards used and the purposes they have.

3.2 Arduino-based data loggers for environmental monitoring

Temperature, humidity and pressure are among the most common parameters that are monitored through Arduino-based data loggers, particularly in environmental applications. For example, an

Arduino-based Cave Pearl data logger was constructed to serve as a multi-purpose monitoring platform [9]. This system used an Arduino Uno board, which is very reliable and can work with different types of sensors; it was used to record the changes in cave conditions such as drip rates and water flow in a flooded cave. The design flexibility was also given much importance, so that the researchers could easily change the system as per different environmental requirements and for different types of sensors.

Furthermore, a data logger system with a sole purpose of measuring temperature and humidity using the Arduino Uno board and the DHT11 sensor was developed [11]. This system also used a real-time clock (DS3231) for time-stamping the data, the LCD display for displaying real-time data, and a piezo buzzer as an alarming system. The study was informed by the urgent need to have a system that can monitor temperature and humidity in one system, with real time data logging capability and an embedded alarming system that would notify the user any time the set temperature and humidity limits were exceeded. This work established the feasibility of the Arduino Uno board in real-time measurement of environmental conditions.

In a similar study, a low cost, multi-sensor Arduino based system for monitoring the dynamics in stream headwater catchments in mountainous regions was developed [12]. The system developed utilized an Arduino Pro Mini board together with combined multi-sensors to measure factors such as water depth and temperature, demonstrating the versatility of Arduino platforms for multi-parameter environmental sensing. In the field tests, the researchers discovered that the monitoring system was power efficient; it was powered by four AA batteries and ran for nine months at a five-minute logging interval. The used Arduino Pro Mini board has similar pins to the Arduino Uno board which is reported to have a higher number of input/output pins. This was convenient because it allowed multiple sensors to be incorporated and offer an extensive array of data regarding the environment.

3.3 Arduino data loggers in energy systems

Another promising area of research is the application of Arduino data loggers in energy systems. They are used to control and manage energy systems. For instance, an Arduino-based data logger was designed to measure photovoltaic (PV) systems' parameters [8]. This system was meant to capture values like current and voltage generated by the solar panels. The system employed an Arduino Mega board. This study illustrated how such a system could be efficiently utilized in observing and enhancing the performance of PV systems, thus making it a useful tool in the utilization of solar energy.

Building on this, another study [10] developed an Arduino-solar power parameter-measuring system with a built-in data logger. This system incorporated an Arduino Uno R3 board; this is because it has the processing capability and memory to support several sensors besides its ability to record data for long durations. It was developed to measure the amount of solar irradiance, the panel temperature, current and voltage, as well as the atmospheric pressure. The validity of the constructed device was verified by comparing the measured parameters with the standard measuring instruments which are found to be in close agreement.

In addition, a multichannel data logger using Arduino Uno for thermal measurement in solar still system was developed [3]. This system was able to record temperature data from several different locations within the solar still, giving a more accurate picture of the thermal process occurring. This application brought out the flexibility of Arduino platforms in handling thermal systems of high complexities.

Furthermore, a data logger was developed to measure thermal conductivity of building materials using Arduino Uno [5]. The performance of the system in terms of temperature logging was the main area of interest in this study since temperature fluctuations are integral in determining the effectiveness of TES materials. Arduino Uno was chosen because it is simple and has enough computing power for thermal measurements required in experimental thermal energy research.

3.4 Arduino data loggers in Agriculture

Arduino based data loggers have also been used in agriculture especially monitoring of environmental conditions that influence crop production. In Turkey, an Arduino based low-cost data logger system was developed to record air temperature, humidity and air pressure in an agricultural environment [6]. The study employed an Arduino uno R3 board to record the environmental factors that are essential in enhancing irrigation and other practices in agriculture. The system was exposed to outdoor conditions for one week in spring and one week in summer and it was discovered that the system could collect data for durations of one-hour intervals. Such cost effective and adaptable instruments were underscored by this study as crucial for improvement of yield in the field of agriculture particularly in the developing world.

In contrast, a study explored the feasibility of using Arduino data loggers in the irrigation systems [13]. Their design was specifically devoted to the assessment of in-canopy sprinklers installed in center pivot irrigation systems. The goal was to design an in-canopy sprinkler monitoring system for center pivot irrigation system which will be capable of identifying the correct location and time of an in-canopy sprinkler separation from the center pivot span. The Arduino Uno board was employed and also the Arduino MKR GSM 1400 board was selected due to the fact that this board has built-in 3G cellular compatible modem. This made it easier for the microcontroller to enable and disable the sending and receiving of the SMS text messages through the Arduino MKRGSM library. This board also encompasses greater flash and dynamic memory as compared with the Arduino Uno. This application depicted how possible it is to design data loggers using Arduino, that suit the needs of agriculture, through providing real-time data that would help in proper usage of water and management of crops.

3.5 Conclusion

The analyzed works as a whole reflect the flexibility and efficiency of Arduino-based data loggers for different contexts, ranging from the environmental to the solar energy to agricultural applications. The choice of Arduino board – whether it is the Arduino Uno, the Arduino Mega or the Arduino Nano – is based on the number of sensors that are needed as well as the complexity of the data processing that is required and the need for data logging. Besides, these systems do not only point to a cheaper means of data collection but also enable the flexibility that can be required especially when working in different conditions and with varying goals and aims. As research on these fields advances, Arduino-based data loggers will remain very useful tools in acquiring precise and thorough data that will fuel development of renewable energy, environmental discipline and agriculture.

4. Materials and methods

In this project, an Arduino data logger was built using an Arduino-UNO R3 board and Type-K temperature sensors. In addition, a current sensor and voltage regulator was used for additional, possible measurements with a PV panel. **Figure 1** shows the block diagram of the built data logger with four temperature sensors interfaced between the heat storage system and the Arduino board, RTC Module, SD Card, LCD display and a laptop or power supply. The acquired data from sensors are analogue, thus the conversion to digital equivalent was performed within the Arduino UNO analogue-to-digital converter module programmed in C- language.

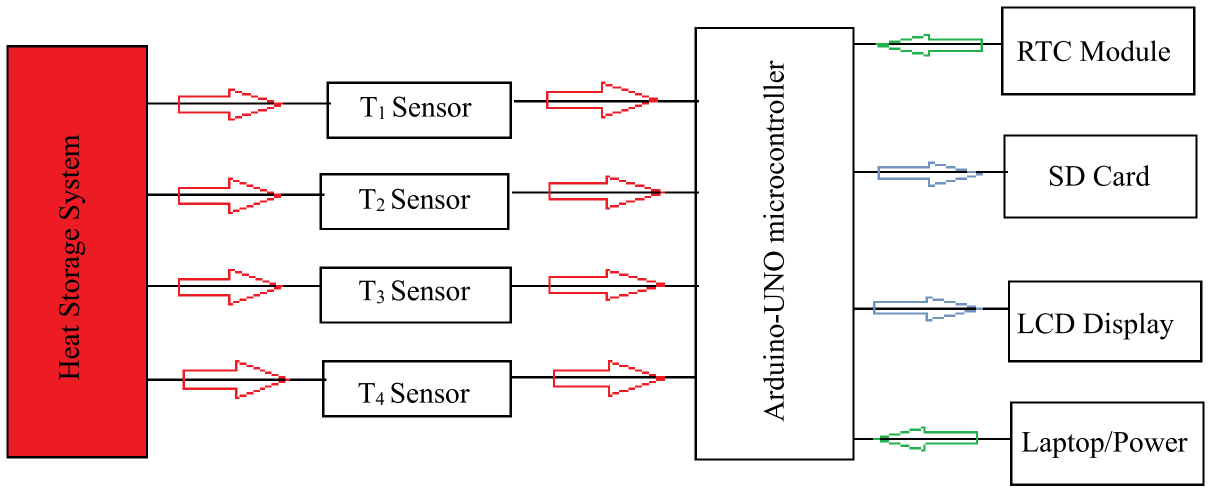

Figure 1: Block diagram of the built Arduino-Uno data logger and connections from the heat storage

4.1 Components used

4.1.1 Arduino UNO R3 board

The Arduino UNO R3 is a circuit board utilizing the ATmega328P microcontroller [9]. The ATmega328P functions as the central processing unit of the data logger, managing all components and processing data from the sensors. **Figure 2** illustrates the board, which features 14 digital input/output (I/O) pins that can be connected to other expansion boards (shields) and other circuits. Six (6) digital pins are capable of functioning as Pulse Width Modulation (PWM) outputs. The board further features six analog inputs, a 16 MHz crystal oscillator, a USB connection serving as both a power source and communication channel, a power jack, an ICSP header, and a reset button [3;9].

The board is programmable with the Arduino IDE (Integrated Development Environment), via a type B USB cable. The IDE program is free to download from the Arduino software webpage, depending on the operating system of the user. The board can be powered by the USB cable or by an external 9-volt battery, though it accepts voltages between 7 and 20 volts [4]. The

ATmega328P on the board is preprogrammed with a bootloader that facilitates the uploading of fresh code without requiring an external hardware programmer [4]. The main characteristics of an Arduino Uno R3 microcontroller are given in Table 1 [9;10] and the board with the labelled parts is shown in **Figure 2** below.

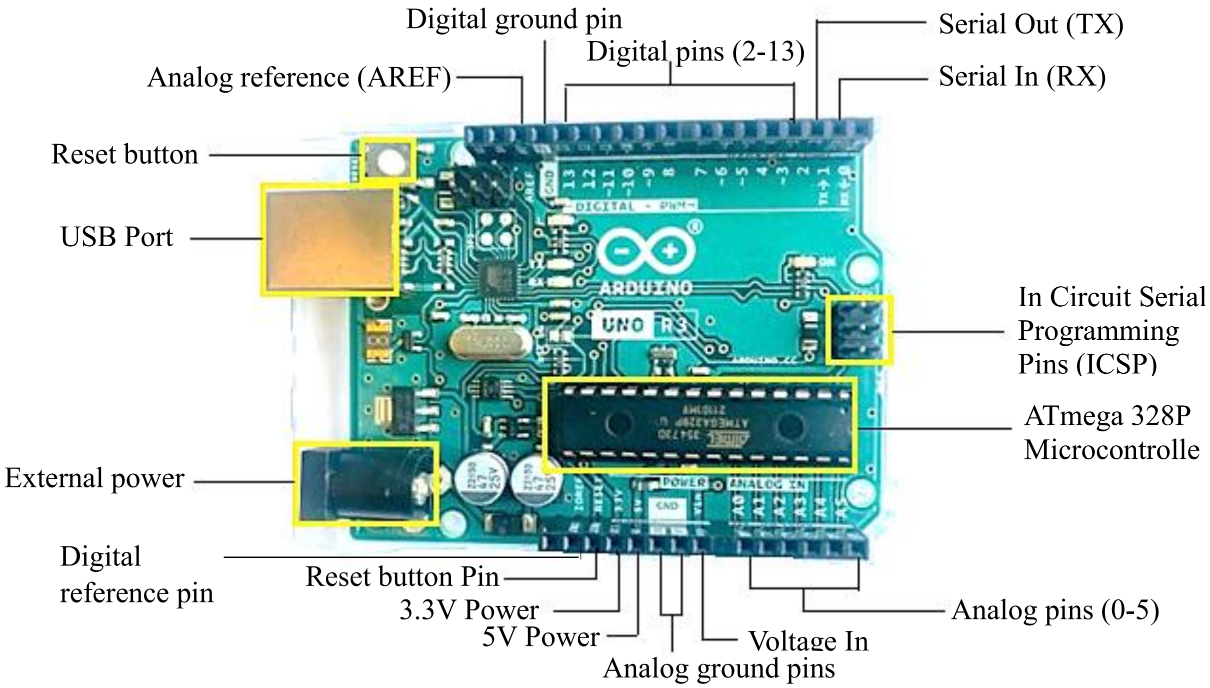

Figure 2: Arduino-Uno R3 board

All 14 digital pins on the Arduino Uno board can operate as either inputs or outputs, utilizing the pinMode(), digitalWrite(), and digitalRead() methods. All of them function at 5V. Each pin may provide or receive a maximum of 40 mA and is equipped with an inbuilt pull-up resistor ranging from 20 to 50 kΩ, which is disabled by default. Furthermore, certain pins possess specialized capabilities, as indicated below [9]:

i. Serial: 0 (Receive) and 1 (Transmit). Facilitates the reception (RX) and transmission (TX) of TTL serial data. The pins are linked to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.

ii. External Interrupts: 2 and 3. These pins can be set to initiate an interrupt upon a low value, a rising or falling edge, or a change in value.

iii. PWM: 3, 5, 6, 9, 10, and 11. Utilize the analogWrite() method to generate 8-bit PWM output.

iv. SPI: 10 (Slave Select), 11 (Master Out Slave In), 12 (Master In Slave Out), 13 (Serial Clock). These pins facilitate Serial Peripheral Interface (SPI) bus connectivity through the SPI library.

v. LED: 13. A built-in LED is connected to digital pin 13. When the pin is at a HIGH value, the LED is illuminated; when the pin is at a LOW value, it is turned off.

Furthermore, the Arduino Uno possesses 6 analog inputs, designated A0 to A5, each offering 10 bits of resolution (i.e., 1024 distinct values). By default, these inputs can measure from ground to 5 Volts; however, the upper limit of their range can be modified using the AREF pin and the AnalogReference() function.

In addition to digital inputs, certain pins provide particular functions:

vi. TWI: A4 (SDA) or A5 (SCL) pin. Facilitate TWI communication utilizing the Wire library. The remaining pins on the board comprises of:

vii. AREF. Reference voltage for the analog inputs. Utilized in conjunction with analogReference().

viii. Reset. Connects a LOW line to reset the microcontroller, commonly employed to incorporate a reset button on shields that obstruct the reset button on the board [9].

Table 1: Technical specifications of an Arduino UNO board

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage	7-9V
Input Voltage (limits)	6-20V
Digital Input/Output (I/O) Pins	14 (6 provide PWM output)
Analog Input Pins	6 (No output pins)
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) (0.5 KB used by bootloader)
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock/processor Speed	16 MHz
Serial communication protocols	UART – I^{2}C –SPI
Analog-Digital Converter (ADC)	10 [bit]

4.1.2 Data logging shield

This is an add-on board that is used to do the data logging functions. It provides functionalities such as an SD card interface for data storage and a Real-Time Clock (RTC) for timestamping data.

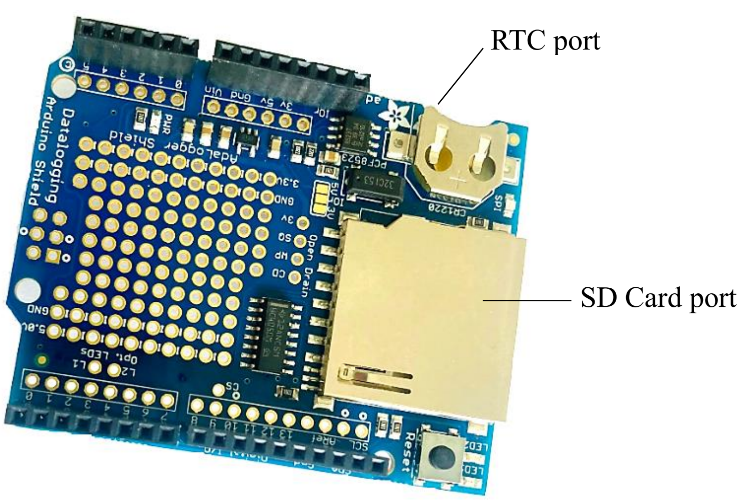

Figure 3: Data logging shield

The shield provides the following features/functionalities [15]:

a) Able to use any SD card with a FAT16 or FAT32format. The 3.3V level shifter circuit enables fast data reading and writing, and prevents damages on SD Card.
b) The RTC ensures that the time will still be ongoing even when the Arduino board is not connected to a power source.
c) A 3.3V voltage on-board regulator can be used as the reference potential (Vref) and to power up the SD card that needs a lot of power to work. This is needed in case one uses a PV system for instance.
d) It uses an "R3 layout" for Inter-Integrated Circuit (I^2C) bus dan ICSP SPI ports, so it will suit many types of Arduino boards

4.1.3 Bi-directional level shifter

A device used to safely interface different voltage levels between components. Here, it is used to interface the voltage between the Arduino (5V logic) and the thermocouple amplifiers (3.3V logic). The level shifter is the connection between the Arduino and the amplifiers for the thermocouples, and enables them to have a single data line in to the Arduino.

Since the Arduino R3 board uses a 5 V logic and the amplifiers use 3.3 V logic all wires connecting the board to the Inter-Integrated Circuit (I^2C) bus needed to pass through a bi-direction logic level converter. The bi-directional logic level converter steps down all outgoing 5V signals to 3.3V while simultaneously stepping up incoming 3.3V signals to 5V. To accomplish this, the converter requires voltage inputs of 5 V and 3.3 V [17]. The bi-directional is shown in Figure 4 [Adafruit].

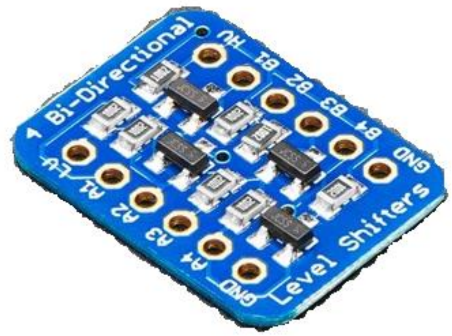

Figure 4: Bi-directional level shifter

The level shifter has a low voltage side (on the left 'A1-LV') and high voltage side (on the right 'B1-HV') as seen in Figure 4. Low voltage side on the left and high voltage on the right. Since the level shifter works on the I^2C communication bus, there is only a need of one data line (SDA) to the amplifiers in addition to ground and power.

4.1.4 Max31850 Type-K Amplifier

Thermocouples are very low-level signals and often require amplification or a high-resolution transducer to process the signals, and since the signal is analog, an analog-digital converter must be present to convert these analog signals into digital signals that are compatible with the Arduino inputs [10]. The amplifiers condition the small voltage output from the thermocouples, amplifying it to a level that can be read by the Arduino.

Each of the amplifiers come with a 2-pin terminal block (for connecting to the thermocouple), a 4.7kΩ data line pullup resistor and a pin header (to plug into any breadboard or perfboard). [Adafruit.com] see **Figure 5** below.

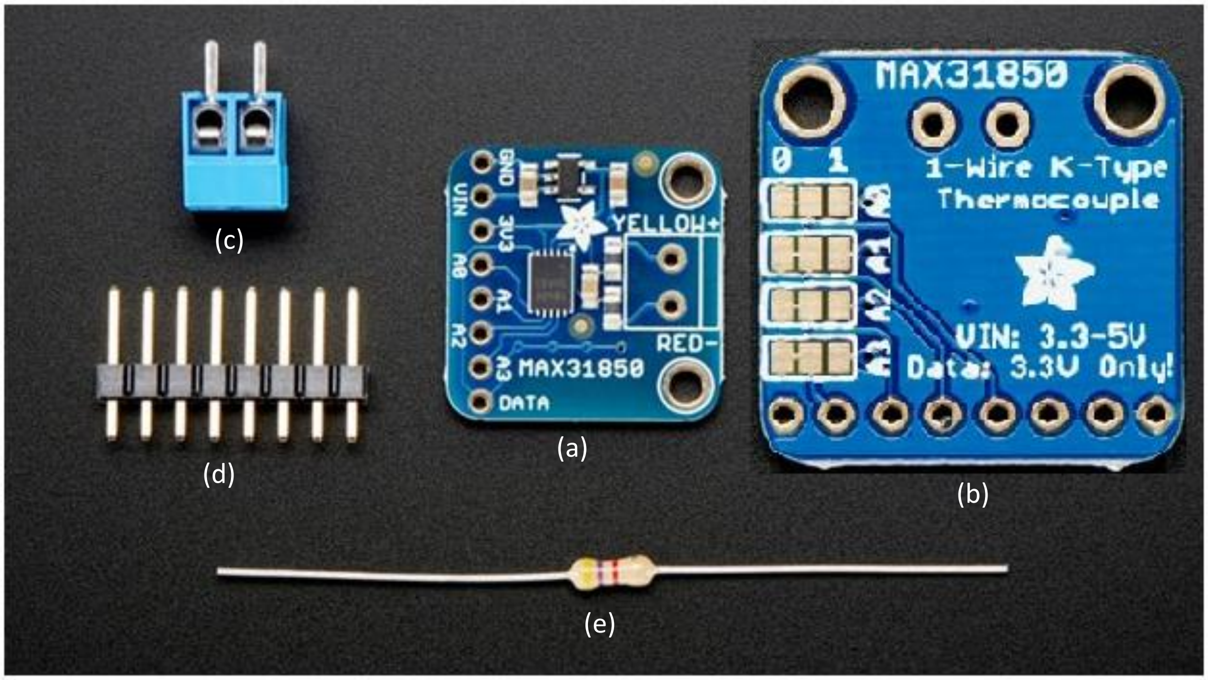

Figure 5: Max31850 Amplifier a) Front look b) Back look c) Terminal block d) pin header e) Resistor Source: [Core Electronics]

Table 2: Features of Max31850 Amplifiers

	Features
1.	Only work with K-type thermocouple (Any)
2.	-270°C to +1370°C output in 0.25 degree increments
3.	Internal temperature reading
4.	3.3 to 5v power supply - Data line is 3V only
5.	1-Wire interface allows any number of thermocouple amps on a single data line

4.1.4.1 Resistor

The resistor (shown in **Figure 5**) is used as a single data line between the level shifter and the amplifiers to limit the current flow and protect the components (such as SD Card) from damage.

4.1.5 Type-K Thermocouples

These are sensors that measure temperature. They produce a voltage proportional to the temperature difference between two junctions. The type-K thermocouple senses the room temperature changes and sends an electric signal to the amplifier, who will amplify the signal and sends it to the Arduino [15].

4.1.6 LCD Shield

A display module that shows real-time data, system status and other information. It allows users to monitor the temperature and other data directly on the device, without having to have a pc/laptop open. The LCD shield is equipped with five programmable buttons as seen at the bottom left of **Figure 6**, a reset button and a display adjustment rheostat (Orange button) on the bottom right. This shield also works on the I^2C communication bus.

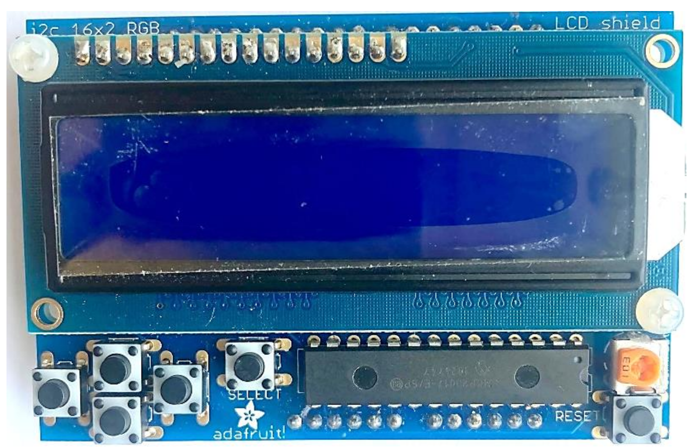

Figure 6: LCD Shield

The LCD buttons were programed as shown in **Table 3** below. To change the functionalities of these buttons, changes need to be made to the Arduino IDE program/codes in Appendix A. **NB:** The buttons need to be pressed for about a second in order to activate.

Table 3: The LCD buttons and their programmed functions

Button	Function
Left	File number readout
Up	Date and log number readout display
Down	Current, Voltage & Power readout display
Right	Turns off display/screen
Select	To show the 4 temperature readings
Orange	LCD adjustment
Reset	Resets the Arduino and starts the program from the beginning

4.1.7 SD Card

The SD card is used to store the temperature readings (or other data) recorded by the data logger over time. After the Arduino receives the signals, the signals will be processed into data that will be written onto the SD Card in a file with **.txt** format which can be opened using a spreadsheet application, like Microsoft Excel. A 16GB card was used to store all data received from sensors.

4.1.8 Real-Time Clock (RTC)

The Real-Time Clock (RTC) Monitors the current time, enabling the data logger to precisely timestamp each recorded data point. The RTC provides the capability to maintain the current time even while the microcontroller is inactive. The real-time clock is powered by a specific battery that is independent of the power supply. Consequently, the date and time for each data entry will remain unaffected when power is disconnected from the circuit [1]. The RTC operates on a 3V lithium coin cell battery, ensuring continuous functionality even when the shield is not powered.

4.1.9 Voltage divider and shunt

In the case where the heat storage system is being heated with a PV panel, a voltage divider and shunt are needed. This is because the PV panel produces a varying voltage due to the variations in solar variation throughout the day. Hence, the Arduino will not be able to use this power directly.

It is in need of a converter to produce a stable 5 V between the power generated by the PV panel and the power input of the Arduino. This voltage converter will need a positive and negative input of power from the PV panel. Since it is desirable to have the Arduino to be able to measure the voltage and current produced by the PV system that will power the heating elements, a shunt will be needed. This shunt is for current measurements.

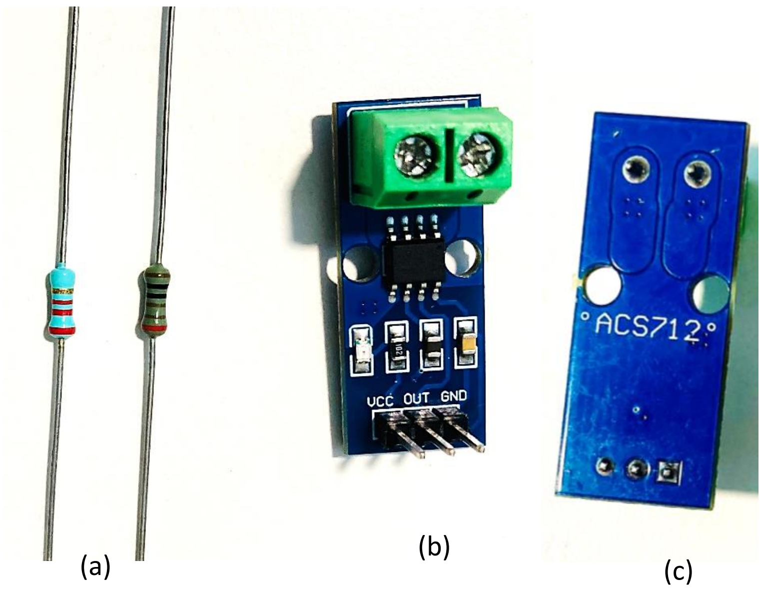

Figure 7: *The resistors for voltage divider (a) and current sensor (b) front (c) back*

The shunt/current sensor has a built-in ACS712 sensor, these sensors use the Hall effect principle to measure current (See) [14]. The current moving through the shunt creates a magnetic field which then is translated to a proportional voltage in the integrated circuit of the sensor. The 2-pin terminal block is soldered to the board and the wires of the external circuit is fastened to this block. To send this information to the Arduino, there are three header pins: VCC, OUT and GND. VCC and GND are for power and ground connection respectively, while OUT is the data line. The ACS712 sensor used has a capacity at 30 amperes and a sensitivity of 66 mV/A.

The solar PV voltage was measured by employing a voltage divider. A voltage divider is a simple circuit that reduces the voltage of the PV panel to a level that can be safely measured by the Arduino. The voltage divider principle implies that: when two resistors are connected in series across a voltage source, the voltage drop across each resistor is proportional to its resistance. The voltage divider takes advantage of this property to "divide" the input voltage into smaller, measurable voltages.

The current sensor interface circuit comprised of two series resistors R1 and R2, obtained as 2.2kΩ and 1kΩ, this could allow an input voltage of up to 16V. In the case of input voltages greater than 16V, several other resistor combinations were added (680Ω & 3.3 kΩ; 330Ω & 3.3 kΩ; 330Ω & 5.1 kΩ) to measure voltages of up to 29V, 55V and 80V respectively. These resistor options were made in such a way that the resulting current does not exceed the accepted/safe value for the Arduino. The highest permissible current for the atMega328 Arduino is 200 mA in total across all pins, with a limit of 20 mA per individual I/O pin, see.

The resistance factor (Rf) was obtained from equation (3) and is responsible for converting the voltage back to the original solar panel value to be displayed on the PC and LCD shield. The measured solar panel output voltage is given by equation (2) [10].

The voltage scaled factor (voltage at the divider junction) is given by:

$$V_f = \frac{R_1 + R_2}{R_1} \qquad \textbf{\textit{(1)}}$$

Where, R_1 is the smaller resistor (closest to the ground) and R_2 is the bigger resistor (closest to the input voltage)

$$Measured\ Voltage = \frac{Voltage\ divider\ analog\ value + Reference\ voltage\ (5V\ for\ arduino)}{Resistance\ factor\ (R_f)} \qquad \textbf{\textit{(2)}}$$

$$R_f = \frac{1023}{R_1/(R_1+R_2)} \qquad \textbf{\textit{(3)}}$$

The measured solar panel current is given by equation (3) [3] and the power is computed from equation (4).

$$Measured\ current = \frac{(Analogue\ value \times Analogue\ factor) - AC\ offset}{Sensitivity} \qquad \textbf{\textit{(4)}}$$

Where: Analogue factor = 5/1023, AC offset = 2500mA, and Sensitivity = 66mV/A

The power of the PV panel was calculated as:

$$Power = Measured\ voltage \times Measured\ current \qquad \textbf{\textit{(5)}}$$

4.1.10 Solar charger shield and Voltage regulator

In addition, a solar charger shield and voltage regulator were added. The solar charger shield is a power system, capable of accepting power from solar cells, and via micro-USB. It is used to charge a Lithium-Ion battery which will provide power to the Arduino when no other power source is connected and it will be charged when external power is available. The battery used in this study is a LiFe 3.7V and 1 200 mAh. This solar shield is suitable for field work, in cases where no electrical power connection is available, the data logger can run on this battery while at the same time being charged by a small PV panel.

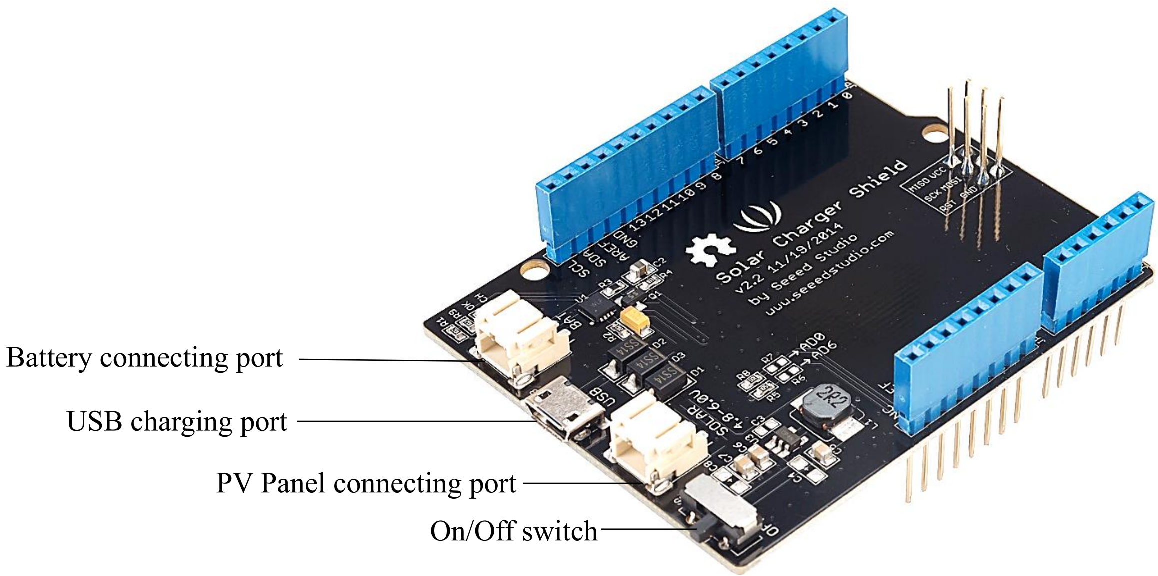

Figure 8: Solar charger shield   Source: Studica

Figure 8 shows the Solar charger. The shield has an on/off switch. If this switch is turned on, the battery and PV panel will be powering the Arduino, and if it is off the Arduino must be connected to an external power through the USB port to stay on. In the case where the Arduino is connected to power through either of the ports and the switch is on the battery will be charging.

In the case where the Arduino is being charged with PV panel of voltage greater than 5V, a voltage regulator will be needed to ensure that no matter the input of the PV panel used, the output will always be 5V (suitable for the Arduino). An LM2596S DC-DC Adjustable Voltage Power Module, from Luxorparts, was chosen in this study (although will not be used). It takes the input power from the PV panel and send out a stable 5 V feed to the Arduino input port.

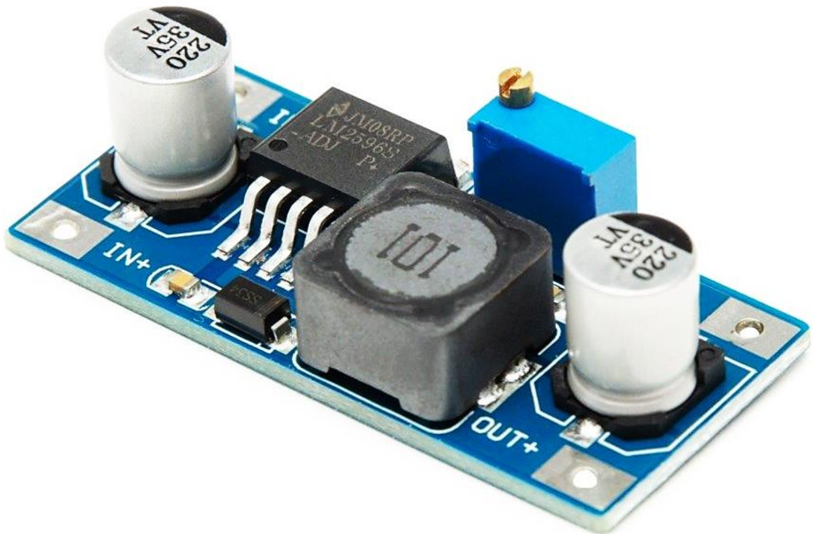

Figure 9: DC-DC Voltage regulator Source: Kjell & Company

5. Soldering and pin configuration

Soldering is the process, commonly used in electronics, that uses a filler metal with a low melting point, also known as solder, to join metal surfaces. The solder is usually made up of an alloy consisting of tin and lead whose melting point is around 235°C and 350°C, respectively. The alloy is melted by using a hot iron (soldering gun) at above 316 °C. As the solder cools, it creates a strong electrical and mechanical bond between the metal surfaces. This bond allows the metal parts to achieve electrical contact while it is held in place [15]. Please (see and this) for a step-by-step guide on soldering.

In this project, soldering was done only for the data logging shield, level shifter and the amplifiers as the other components (Arduino board and LCD shield) were already soldered.

5.1 Data logging shield

The first step to soldering the data logging shield is getting the right (sized) stacking headers and soldering them on the shield as shown in **Figure 10**. Stacking headers were used in order to allow the stacking of other shields (e.q LCD shield) on top of the data logging shield.

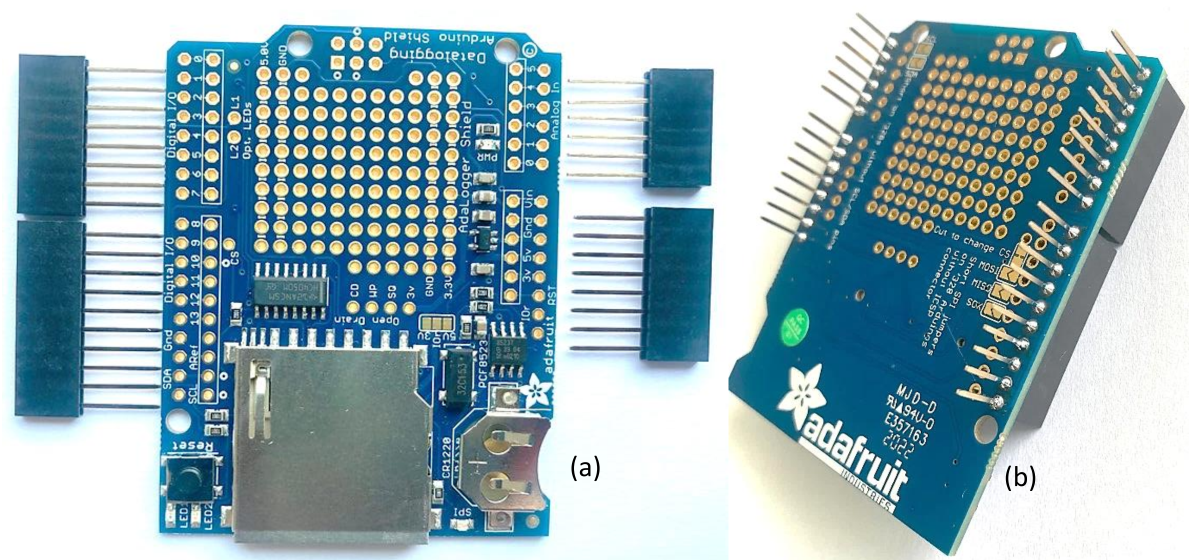

Figure 10: Data logging shield (a) before and (b) after soldering

5.1.1 Level shifter

The second step of soldering the logging shield was adding the level shifter. The wires that connect the level shifter to the data logging shield need to be soldered on the level shifter before soldering it to the data logging shield. It would be best if different color-coded wires could be used for the different pins, for easy identification. In this work, a single color ‘yellow’ was used as those were the only wires available and appropriate for soldering on the level shifter as shown in **Figure 11**.

On the low voltage side LV (first, top-left) is connected to the 3.3 V of the data logging shield. A1 (second, top-left) is connected to the 4.7 kΩ resistor, which also goes to the 3.3 V. On the high voltage side, HV (first, top-right) is connected to 5.0 V on the data logging shield while, B1 (second, top-right) is connected to one of the of 14 digital inputs. B1 is for the data line to the

Arduino. According to the Arduino IDE program that was used in this project, this need to be connected to pin 2. If a different pin is desirable, then the program needs to be changed as well. Ground (bottom wire) on both sides go to the ground of the data logging shield.

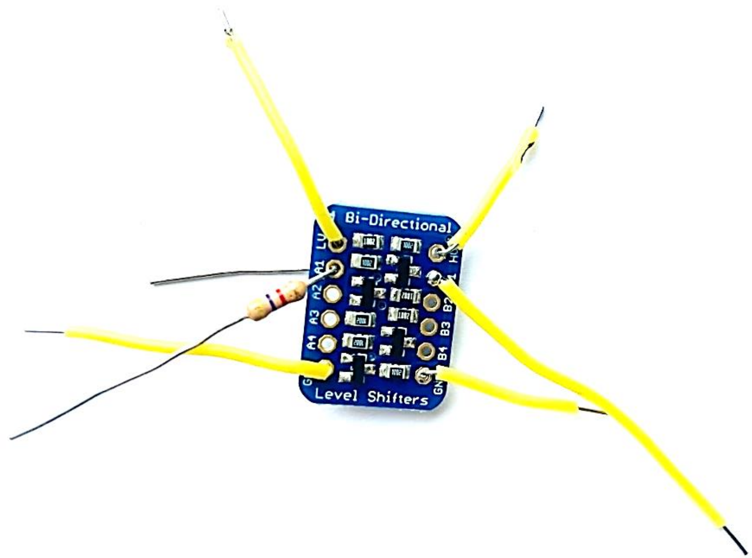

Figure 11: The level shifter after soldering

After soldering the wires to the level shifter, it is then soldered on the data logging shield according to the descriptions given above. In order to prevent a short circuit, after soldering the wires to the level shifter, all the wires (**Except the resistor**) need to be cut as short as possible. The resistor needs to be long enough, so that it can pass through the hole on the logging shield to the other side. The resistor needs to first be soldered on the data logging shield, with the send side passing through the hole to the other side and then soldering of the rest of the wires follows.

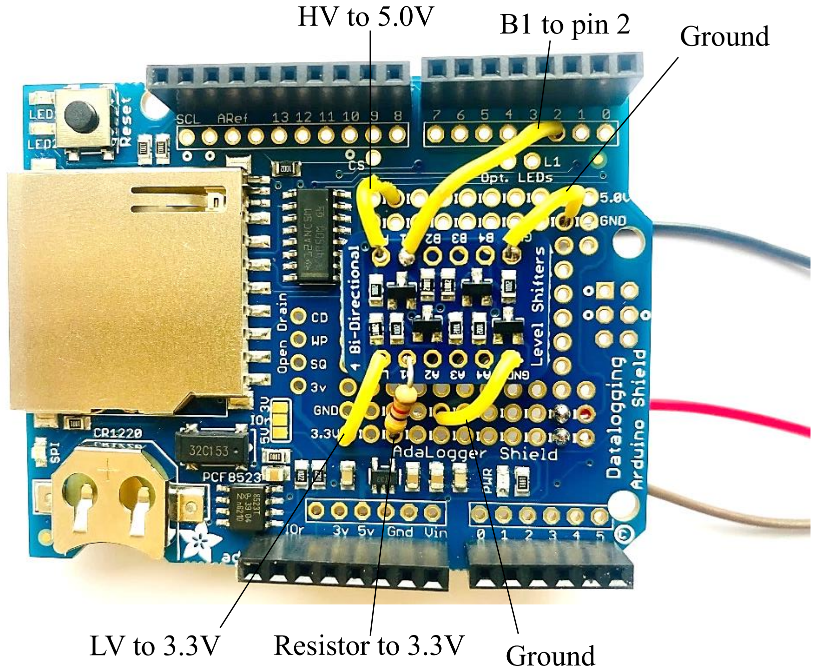

Figure 12: Data logging shield with the Bi-directional level shifter

The resistor is used as a data line from the amplifiers and it is the black wire in **Figure 12** and **Figure 13**. In **Figure 13**, the Brown wire is 3.3 V power line to the amplifiers, while the red wire is for ground.

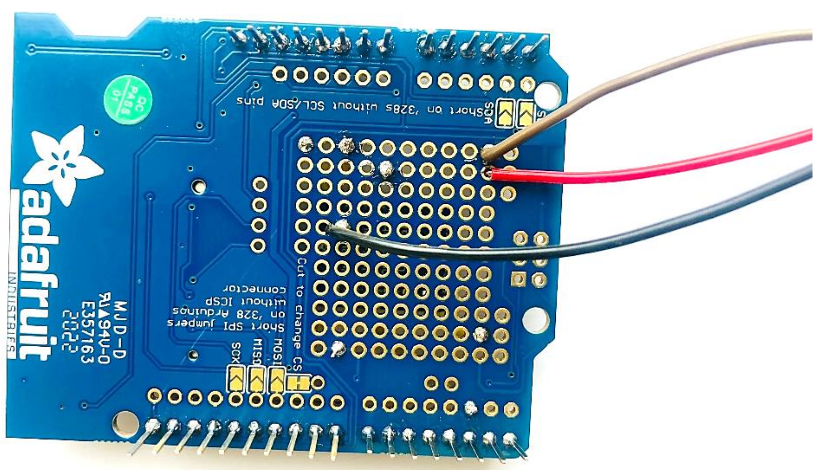

Figure 13: Back side of the data logging shield with the level shifter

5.2 Max31850 Amplifiers

The amplifiers (as shown in **Figure 14**) are mounted on a stripboard, this enables all the four amplifiers to send data to the same line output, as long as they are connected in the same way and on the same lines on the stripboard. The stripboard mounting makes it easy for a connection of only three (3) wires between it and the Arduino: Ground, power and a data line. The amplifier is connected to the stripboard using header pins and an extra header pin is mounted at the top of the amplifiers on the stripboard for the connection of the three wires to the Arduino.

The orientation of the amplifier is important as one need to know which header pin is needed for ground, power and data line. In addition, one need to confirm that the correct terminals (+ve and -ve) of the 2-pin terminal block is connected/soldered correctly to the amplifiers, this will ensure a correct connection to the thermocouple sockets. In **Figure 14**, some amplifiers are connected front-side down (See **Figure 5** (b)) on the stripboard.

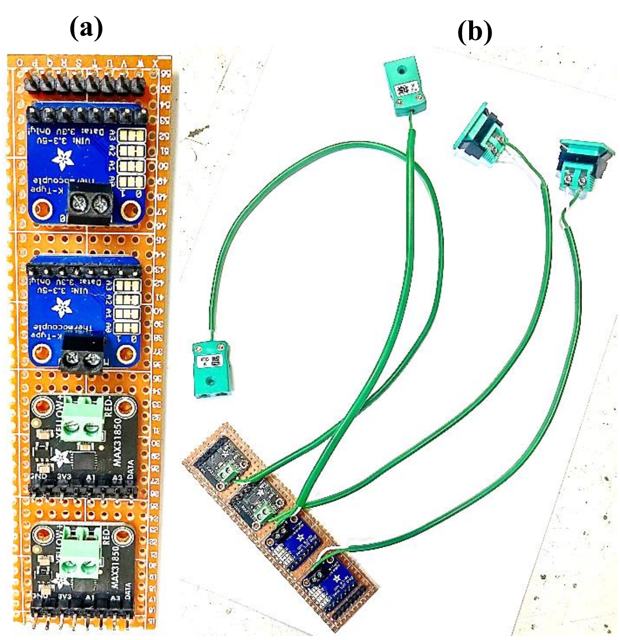

Figure 14: The four amplifiers a) on the Stripboard, b) connected to type-K thermocouple sockets

5.3 Shunt and voltage divider interface

The final part of soldering was for the current sensor interface circuit (voltage divider and shunt) onto a stripboard as shown in **Figure 15**.

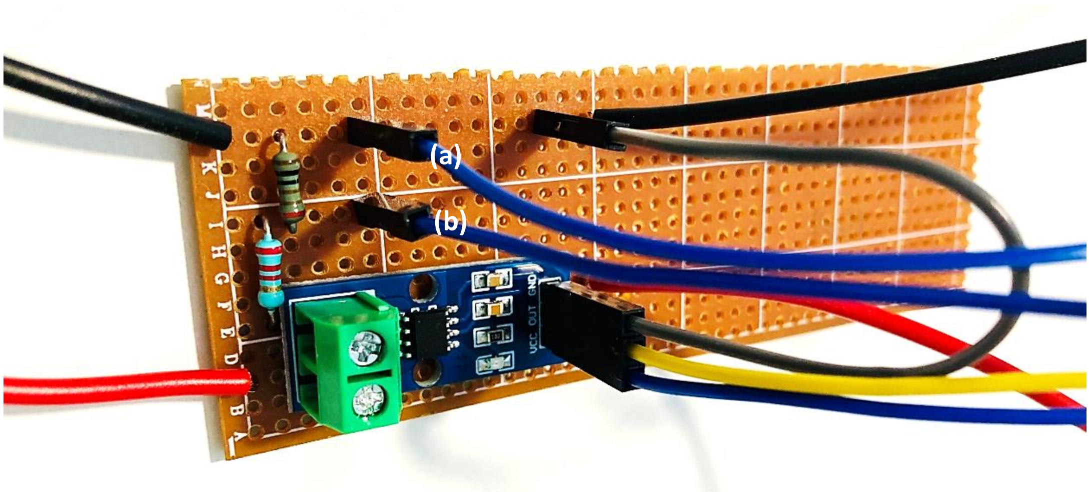

Figure 15: *The* current sensor and voltage divider interface

In **Figure 15**, the wires connected to the current sensor's three pins (VCC-Blue, Out-Yellow and GND-Grey) are color coded. The Yellow wire will connect to pin 2 on the Arduino (for current measurement), the blue wire will be connected to the analog pin 5V (data line) and Grey is for ground connection. Moving to the top (the two blue wires) a) is for ground connection while b) is for voltage measurement from the voltage divider. The two thick wires (red, +ve and black, -ve) are for connections to the power source (PV panel). Likewise, the two thick (black and red) wires at the back are for connections to the heat storage system (heating elements).

5.4 The complete data logger

The four shields (Arduino board, Data logging shield, Solar charging shield and the LCD shield) are stacked together to make an Arduino stack, while the other components are either connected to the shields or the Arduino to make up the total system as seen in **Figure 16** below and all components were well arranged in a box with some small openings to allow connections to the heat storage system. The USB cable (black in cable in bottom right picture) is used for connecting the Arduino to a laptop (or Power bank), where it can get power and the program that is written can be downloaded and stored on the device.

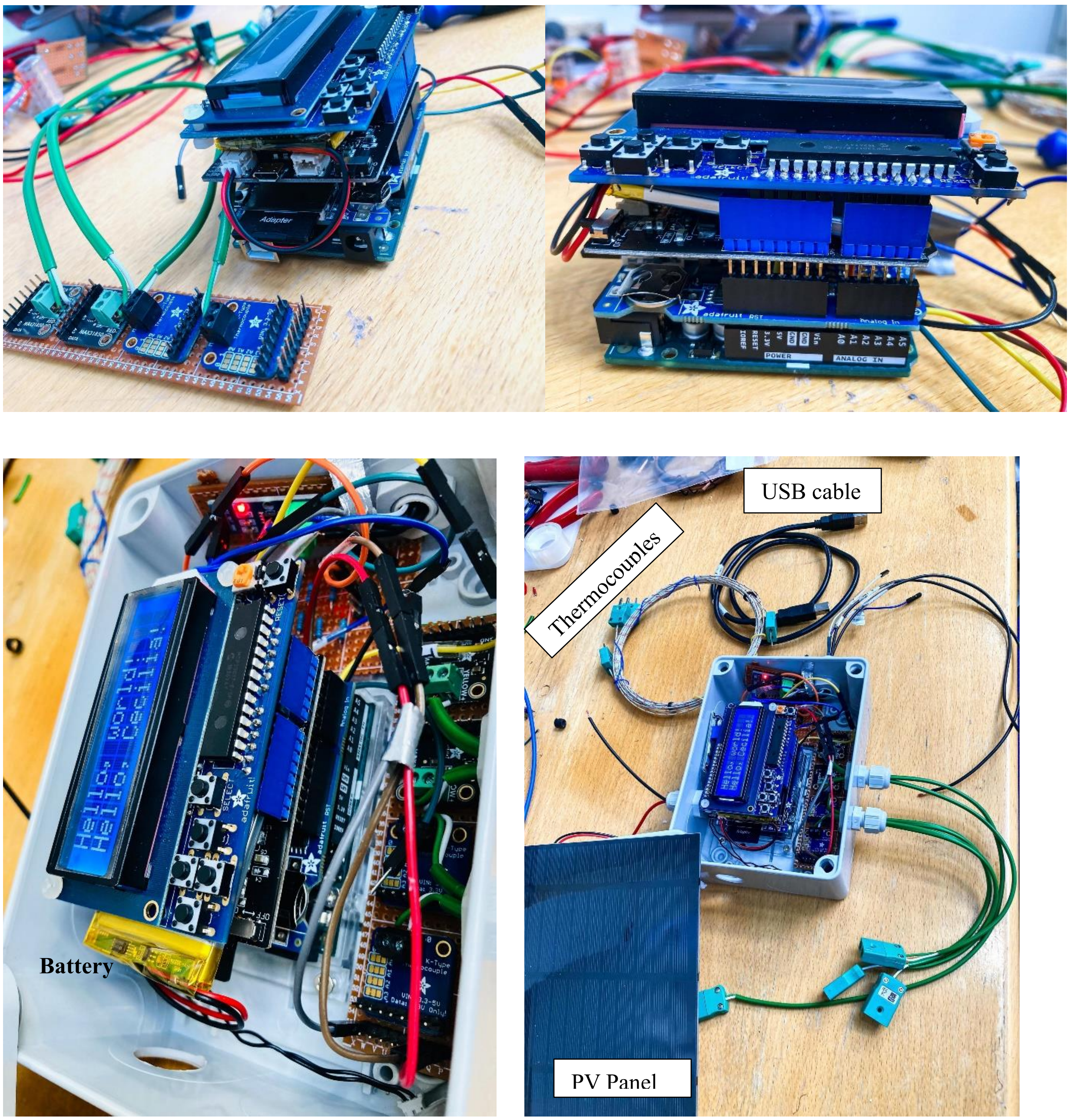

Figure 16: The complete Arduino data logger

The thermocouple sockets are wired with (green and white wires) to the 2-pin terminal blocks of the amplifiers. When connecting the sockets to the terminal block, one need to ensure that the correct poles are connected for each of the wires. The three wires: ground (brown), power (black) and data line (red) are connected from the stripboard to the data logging shield.

6. Arduino IDE Program

When starting the Arduino for the first time, it needs to be connected to the laptop via a USB Type A to B 2.0 cable and the Arduino IDE program is run on the laptop. The codes (in Appendix A) which are uploaded to the Arduino, from the IDE program, is called a “sketch”. These codes stay in memory until it is replaced with new codes, even when the power is off. Thus, it is possible to develop and upload code when Arduino board is connected to a computer and then run that code with some other power supply (like a power bank or battery) when it is no longer connected to a computer. This is an essential feature for outdoor systems [9]. The Arduino in this study was programmed to perform the following functions: To read the thermocouple signals; show the temperature readings on the LCD screen and record and save the readings on the SD Card.

The thermocouples sense when the room temperature changes, and sends an electric signal to the amplifier, that will amplify the signal and sends it to the Arduino. After the Arduino receives the signal, the signal will be processed into a data that will be written onto the SD Card in a file with .txt format which can then be opened using a spreadsheet application, like Microsoft Excel [15]. **Figure 17** shows a flow chart of the steps from when one starts the Arduino to saving and displaying the data.

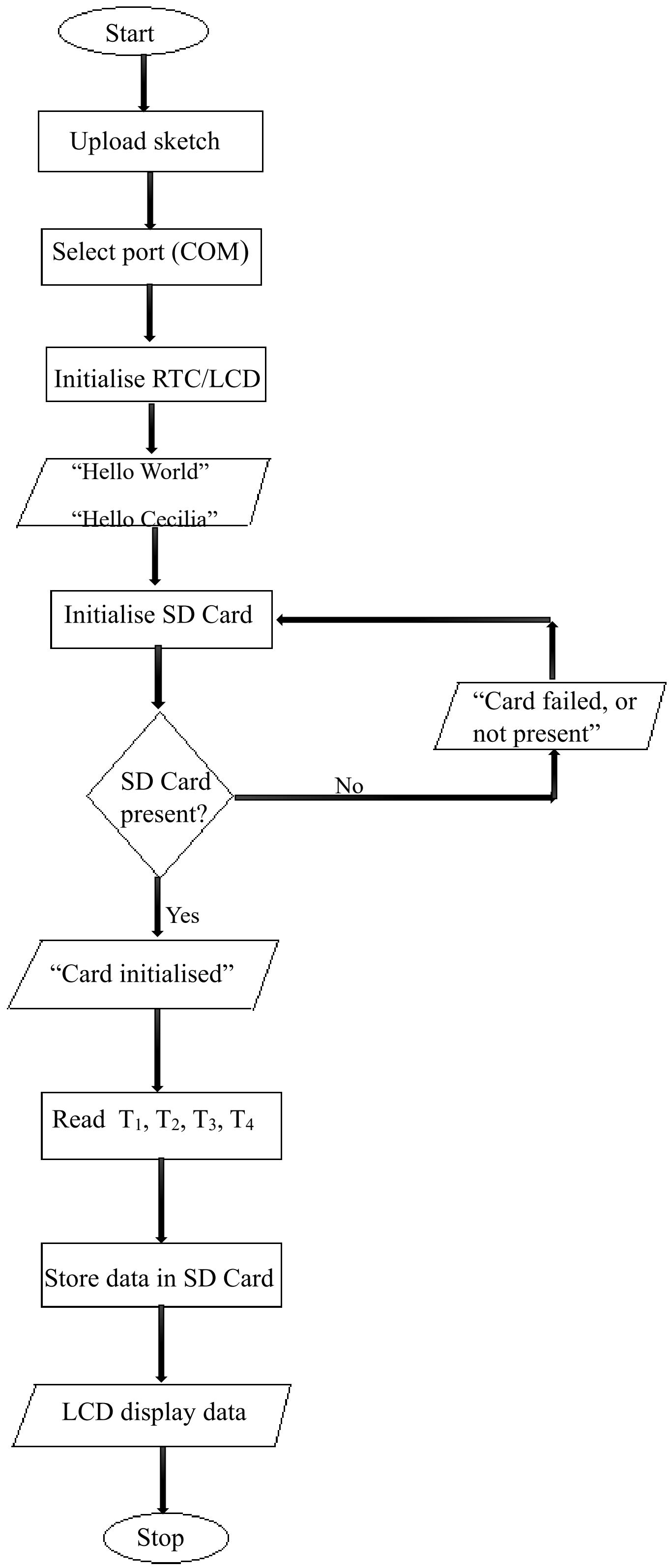

Figure 17: Simplified flowchart of steps taken by the Arduino IDE to acquire and store data

7. Testing of the developed data logger

Before being used on the heat storage system, the data logger was tested by boiling water with heating elements. This test was done in order to confirm whether all sensors were working properly, if the data was being logged properly and in a correct format. The temperature readings were done in an interval of 5 seconds. Some results are shown below.

20240906 - Notepad

File Edit Format View Help

6-9-2024	10:23:25	log	T1	T2	T3	T4	Ampere	Voltage	power
6-9-2024	10:23:25	0	21.50	21.50	20.75	21.25	0.08	0.00	0.00
6-9-2024	10:23:30	1	21.25	21.00	20.50	21.00	0.08	0.00	0.00
6-9-2024	10:23:35	2	21.00	21.00	20.50	21.25	0.06	0.00	0.00
6-9-2024	10:23:40	3	20.75	21.00	20.50	21.00	0.03	0.00	0.00
6-9-2024	10:23:45	4	20.75	20.75	20.50	21.00	0.03	0.00	0.00
6-9-2024	10:23:50	5	21.00	20.75	20.50	21.25	0.06	0.00	0.00
6-9-2024	10:23:55	6	21.00	20.75	20.50	21.25	0.05	0.00	0.00
6-9-2024	10:24:0	7	21.00	21.00	21.00	20.00	0.06	0.00	0.00
6-9-2024	10:24:5	8	21.00	21.00	24.75	19.75	0.08	0.00	0.00
6-9-2024	10:24:10	9	21.00	21.00	26.00	19.75	0.03	0.00	0.00
6-9-2024	10:24:15	10	21.25	21.00	26.00	19.50	0.08	0.00	0.00
6-9-2024	10:24:20	11	21.25	21.00	26.50	20.00	0.07	0.00	0.00
6-9-2024	10:24:25	12	21.25	21.00	26.50	19.75	0.04	0.00	0.00
6-9-2024	10:24:31	13	21.25	21.00	26.75	19.75	0.00	0.00	0.00
6-9-2024	10:24:36	14	21.25	21.00	26.75	19.75	0.07	0.00	0.00
6-9-2024	10:24:41	15	21.50	20.75	26.75	19.50	0.04	0.00	0.00
6-9-2024	10:24:46	16	21.50	20.75	26.75	19.50	0.05	0.00	0.00
6-9-2024	10:24:51	17	21.50	20.75	26.75	19.75	0.06	0.00	0.00
6-9-2024	10:24:56	18	21.50	20.75	26.75	19.75	0.03	0.00	0.00
6-9-2024	10:25:1	19	21.25	20.75	26.75	19.50	0.06	0.00	0.00
6-9-2024	10:25:6	20	21.50	20.75	26.75	19.50	0.03	0.00	0.00
6-9-2024	10:25:11	21	21.25	20.75	26.50	19.50	0.06	0.00	0.00
6-9-2024	10:25:16	22	21.25	20.75	26.50	19.50	0.05	0.00	0.00
6-9-2024	10:25:21	23	21.25	20.75	26.75	19.50	0.02	0.00	0.00
6-9-2024	10:25:26	24	21.50	20.75	26.25	19.25	0.03	0.00	0.00
6-9-2024	10:25:31	25	21.25	20.50	26.00	19.25	0.02	0.00	0.00
6-9-2024	10:25:37	26	21.50	20.50	25.25	19.25	0.03	0.00	0.00
6-9-2024	10:25:42	27	21.50	20.50	25.00	19.25	0.05	0.00	0.00
6-9-2024	10:25:47	28	21.25	20.50	25.50	19.25	0.06	0.00	0.00
6-9-2024	10:25:52	29	21.50	20.50	25.25	19.25	0.03	0.00	0.00
6-9-2024	10:25:57	30	21.50	20.75	24.50	19.25	0.06	0.00	0.00
6-9-2024	10:26:2	31	21.50	20.75	24.25	19.00	0.07	0.00	0.00
6-9-2024	10:26:7	32	21.50	20.75	24.25	19.25	0.03	0.00	0.00
6-9-2024	10:26:12	33	21.50	20.75	24.00	19.25	0.07	0.00	0.00
6-9-2024	10:26:17	34	21.25	20.75	24.50	19.25	0.05	0.00	0.00

Figure 18: *Testing the data logger*

References

[1] Abdulsalam KA, Adebisi JA and Oluwasanjo OB 2023 Development of an Arduino temperature data logger - A framework *FUW Trends in Science & Technology Journal* **8** pp. 139 – 146.

[2] Medojevic M, Medojevic M, Radakovic N, Lazarevic M and Sremcev N 2018 A conceptual solution of low-cost temperature data logger with relatively high accuracy *International Journal of Industrial Engineering and Management* **9** pp. 53-58.

[3] Roihan I and Koestoer RA 2020 Data logger multichannel based on Arduino-Uno applied in thermal measurement of solar still Carocell L3000 *In AIP Conference Proceedings* **2314**.

[4] Obi AI, Udosen AN and Anyaoha CO 2019 Design, construction and testing of multipoint humidity, temperature data Logger *In Proceedings of the 1st International Multidisciplinary Conference on Technology*, *Nsukka, Nigeria* **1** pp. 101-109.

[5] Saliby A and Kovács B 2022 Thermal data logging used in thermal conductivity apparatus based on Arduino-Uno *Academic Journal of Manufacturing Engineering* **20** pp. 109-114.

[6] Polat MY 2020 A low-cost microcontroller-based Air Temperature, Humidity and Pressure datalogger system design for Agriculture *Yuzuncu Yıl University Journal of Agricultural Sciences*, **30** pp. 211-219.

[7] Wahyudi RNF and Santoso I 2023 Design of Temperature data logger using Thermocouple *International Research Journal of Engineering and Technology (IRJET)* **10** pp. 566-570.

[8] Mahzan NN, Omar AM, Rimon L, Noor SM and Rosselan MZ 2017 Design and development of an arduino based data logger for photovoltaic monitoring system *International Journal of Simulation: Systems, Science and Technology* **17** p. 15.1.

[9] Beddows PA and Mallon EK 2018 Cave pearl data logger: A flexible arduino-based logging platform for long-term monitoring in harsh environments *Sensors* **18** p. 530.

[10] Oladimeji I., Adediji YB, Akintola JB, Afolayan MA, Ogunbiyi O, Ibrahim SM and Olayinka SZ 2020 Design and construction of an Arduino-based solar power parameter-measuring system with data logger *Arid Zone Journal of Engineering, Technology and Environment* **16** pp. 255-268.

[11] Kuria KP, Robinson OO and Gabriel MM 2020 Monitoring temperature and humidity using Arduino Nano and Module-DHT11 sensor with real time DS3231 data logger and LCD display *International Journal of Engineering Research & Technology (IJERT)* **9** pp. 416-422.

[12] Assendelft RS and Van Meerveld HI 2019 A low-cost, multi-sensor system to monitor temporary stream dynamics in mountainous headwater catchments *Sensors* **19** p. 4645.

[13] Akin AA, Rogers DH and Aguilar J 2019 Design of an in-canopy sprinkler monitoring system for center pivot irrigation *In World Environmental and Water Resources Congress, Reston, VA: American Society of Civil Engineers* pp. 28-39.

[14] Allegro MicroSystems. ACS712: Hall-Effect-Based Linear Current Sensor IC. Online: https://www.allegromicro.com/en/products/sense/current-sensor-ics/zero-to-fifty-amp-integrated-conductor-sensor-ics/acs712.

[15] Sild S 2022 Soldering Explained – Definition, Process, Types Online: https://fractory.com/soldering-explained/.

Appendices

Appendix A: Arduino IDE codes

```
// Program for logging temperatures, current, voltage and power
// Builds on version for PTC switching
// Compute averaged power values after each reading
// At printout time, reset reading counter and print the values
// temperatures not averaged
// By: Prof Ole J Nydal and Cecilia Naule, August 2024, at NTNU, Norway

//Libraries included

#include <DallasTemperature.h>
#include <OneWire.h>
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <RTClib.h>
#include <Adafruit_RGBLCDShield.h>
#include <utility/Adafruit_MCP23017.h>

//Definitions, where the various pins are connected, allocating memory to arrays and variables.
#define ONE_WIRE_BUS 2        // Data is connected to pin 2
#define SD_CS 10              // pin to write to data logger shield pin 10
#define TEMPERATURE_PRECISION 9
#define SERIAL_PRINT_ON 1     // put to 0 if no print of data to serial
#define SELECT 1              // LCD buttons
#define UP 2
#define DOWN 3
#define RIGHT 4
#define LEFT 5
#define PINAMPERE A2         // Shunt connected to A2
#define PINVOLTAGE A0        // Voltage after voltage splitter connceted to A0

// The shield uses the I2C SCL and SDA pins. On classic Arduinos
// this is Analog 4 and 5 so you can't use those for analogRead() anymore
// However, you can connect other I2C sensors to the I2C bus and share
// the I2C bus.

OneWire oneWire(ONE_WIRE_BUS); // Setup a oneWire instance to communicate with any OneWire devices
                               //(not just Maxim/Dallas temperature ICs)
// DeviceAddress insideThermometer; // arrays to hold device addresses,  not used?
DallasTemperature sensors(&oneWire);

//#define seconds() (millis()/1000)   // help function to get seconds from start, not used
RTC_PCF8523 rtc;                      // This is the RTC on the Adafruit Logger shield

char dateChar[50];                    // used for lcd display
char temperatureChar[10];             // used for lcd display

//R1 =1, R2 =2.2  gives R1/(R1+R2)= 0.5
//float voltageCalibration=3.0;  // calibration depending on voltrage splitter.
//float voltageCalibration=3.22;           //from voltage divider calculations  //Fist (small) splitter
float voltageCalibration=5.85;           //from voltage divider calculations  //second splitter
//float sensitivityAmpere=0.185;  // 0.185 for 5A sensor, 0.100 for 20 A sensor and 0.066 for 30A sensor
// used with Ampere=(analogRead(PINAMPERE)*(5.0 / 1023.0)-2.5)/sensitivityAmpere;

//float offsetAmpere = 511.5; // calibrate zero current alternatively
float offsetAmpere = 509.0;
//float offsetAmpere = 510.5;

//float spanAmpere = -0.07315;
float spanAmpere = -0.073;

```

```
//float spanAmpere = -0.1;  // calibrate 30A ACS712 (https://forum.arduino.cc/t/how-to-calibrate-acs712-properly/540323/2)
// used with Ampere=(analogRead(PINAMPERE)-offsetAmpere)*spanAmpere;

unsigned int long intervalWrite= 5000; // time between writing to file (milli seconds) 5s
unsigned int long prevWrite = 0;     //
unsigned int read_numbers =0;        // counter for number of reading for averaging
unsigned long nowMillis = 0;         // milliseconds
int delayReading=100;                // delay between the reading of sensors

String dateString;              // for output

unsigned int logNumber=0;       // counter for each log

int LCD_button=0;               // active button on lcd
int fileNumber=0;               // number of writing to files
bool NEWFILE=0;
char filename[] = "YYYYMMDD.txt";

DateTime now;                   // current time
String timeNow;                 // time string for writing

Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();   // lcd shield

struct powerStruct {            // Voltage, Ampere and Power lumped in a struct
    float Ampere; //
    float Voltage;
    float Power;
};

powerStruct power, power_now, power_accumulated;   // power variables

File myFile; //  Creating empty file for the SD functions


// FUNCTIONS

// function to get power
//powerStruct powerRead() {
//  powerStruct powerStructVar;
//  float auxAmp = analogRead(PINAMPERE)*(5.0 / 1023.0);  // read Ampere signal
//  powerStructVar.Voltage = analogRead(PINVOLTAGE)*(voltageCalibration*5.0 / 1023.0);  // read Volt and calibrate
//  powerStructVar.Ampere=(auxAmp-2.5)/sensitivityAmpere;  // calibrate ampere
//  powerStructVar.Power=powerStructVar.Ampere*powerStructVar.Voltage;   // new power
//  return powerStructVar;
//}
// ---------------------------------------------------

// function to get power
powerStruct powerRead() {
  powerStruct powerStructVar;
  float auxAmp = analogRead(PINAMPERE);  // read Ampere signal
  //powerStructVar.Voltage = analogRead(PINVOLTAGE)*(voltageCalibration*5.0 / 1023.0);  // read Volt and calibrate
  powerStructVar.Voltage = analogRead(PINVOLTAGE)*(voltageCalibration*5.0 / 1023.0);
  powerStructVar.Ampere=(auxAmp-offsetAmpere)*spanAmpere;  // calibrate ampere
  powerStructVar.Power=powerStructVar.Ampere*powerStructVar.Voltage;   // new power
  return powerStructVar;
}
// ---------------------------------------------------
```

```

// function to get power and accumulate for averaging
powerStruct powerReadAndAccumulate(powerStruct p) {
  powerStruct powerlogged, powerStructVar;
  powerlogged = powerRead();    // read the power
  powerStructVar.Power = powerlogged.Power + p.Power;
  powerStructVar.Ampere = powerlogged.Ampere + p.Ampere;
  powerStructVar.Voltage = powerlogged.Voltage + p.Voltage;
  return powerStructVar;
}
// ---------------------------------------------------

// Function to set date and time stamp
void dateTime(uint16_t* date, uint16_t* time) {    // This funciton is used to set date and time stamp for the file creation
  DateTime now = rtc.now();     // return date using FAT_DATE macro to format fields
  *date = FAT_DATE(now.year(), now.month(), now.day());
  *time = FAT_TIME(now.hour(), now.minute(), now.second());
}
// ----------------------------------------------------


// Function to write to file
void writeToFile(){
  // Set file name from date and time
  filename[0] = '0' + (now.year() / 1000);
  filename[1] = '0' + (now.year() / 100) % 10;
  filename[2] = '0' + (now.year() / 10) % 10;
  filename[3] = '0' + (now.year() % 10);
  filename[4] = '0' + (now.month() / 10) % 10;
  filename[5] = '0' + (now.month() % 10);
  filename[6] = '0' + (now.day() / 10) % 10; // use minute below if new file every minute
  filename[7] = '0' + (now.day() % 10);
  //filename[6] = '0' + (now.minute() / 10) % 10;
  //filename[7] = '0' + (now.minute() % 10);
  //Serial.println (filename);

  dateString = String(now.day())+"-"+String(now.month()); // modify this?
  dateString= dateString+"-"+ String(now.year());

  //String hours = String(now.hour());
  //if(now.minute()<10) {              //If the minute counter is less than 10, add a 0 in front.
  //  hours = hours+":0"+String(now.minute());
  //  }
  //  else{
  //    hours = hours+":"+String(now.minute());
  //}
                                       // try direct, but we lose constant length in date-time
  timeNow= String(now.hour())+":"+String(now.minute())+":"+String(now.second());

```

```
  //hours.toCharArray(timeChar,100);      // lcd use char arrays?
  //dateString.toCharArray(dateChar,100);  // ??   check if used

  //
  SdFile::dateTimeCallback(dateTime);  // function to set the date time for the file creation
                                        //  this was needed to get the correct time stamp on the file
  if (! SD.exists(filename)) {         // flag if new file to be made. The filename has the time
                                       // here we include day in file name, so new file every day
                                       // Could be changed to include new folders also (e.g. every month)
    NEWFILE=1;
    //fileNumber=fileNumber+1;   // for LCD display

  };
  myFile=SD.open(filename,FILE_WRITE);

  if(! myFile){                        // see if the card is present and can be initialized:
    Serial.println("Error opening data.txt file."); // Error message if opening fails.
    Serial.println("Card failed, or not present");
    lcd.setCursor(0, 0);               // write also to LCD
    lcd.print("   no SD card!");
    lcd.setCursor(0,1);
    lcd.print("insert and reset");
    while(1);                          // Hold if opening fails.
  }
    // Saving data to SD card
    // If a new file has been made, a header is printed, followed by the data in columns
  if (NEWFILE){                        // a new file has been opened
    myFile.print(dateString);
    myFile.print(" ");
    myFile.print(timeNow);
    myFile.print("      ");
    myFile.println("log    T1    T2     T3      T4      Ampere   Voltage   power  ");
    //myFile.println(" ");          // new line
    NEWFILE=0;                    // reset  new file flag
    lcd.clear();                  // clear screen as counters are reset
    //myFile.close();  //Close the new file with printed header. NB closing and reopening right after caused failure
  }
                                // print columns with data for each logging
  myFile.print(dateString);
  myFile.print("  ");
  myFile.print(timeNow);
  myFile.print("  ");
  myFile.print(String(logNumber));
  for (uint8_t i = 0; i < 4; i++){  // Temperature values to SD card
    myFile.print("    ");
    myFile.print(sensors.getTempCByIndex(i));
  }
  myFile.print("   ");
  myFile.print(power.Ampere,2);
  myFile.print("   ");
  myFile.print(power.Voltage,2);
  myFile.print("   ");
  myFile.print(power.Power,2);
  myFile.println(" ");   // new line
  myFile.close();  //Close the file.
}
// ----------------------------------------------------
```

```
// functio to update the LCD
void updateLCD() {
  // Colors for backlight, Many are not used
  #define RED 0x1
  #define YELLOW 0x3
  #define GREEN 0x2
  #define TEAL 0x6
  #define BLUE 0x4
  #define VIOLET 0x5
  #define WHITE 0x7
  // define the content for each button
  if(LCD_button==SELECT){
    //lcd.clear();
    for (uint8_t i = 0; i < 4; i++){
      //Diplsay the temperature values to the LCD.
      if (i==0)lcd.setCursor(0,0);
      if (i==1)lcd.setCursor(8,0);
      if (i==2)lcd.setCursor(0,1);
      if (i==3)lcd.setCursor(8,1);
      //lcd.print("T");
      lcd.print(i+1);
      lcd.print(": ");
      lcd.print(sensors.getTempCByIndex(i),1);
      lcd.setBacklight(WHITE);
    }
  }
  if(LCD_button==UP){
     //lcd.clear();
    lcd.setCursor(0,0);
    //lcd.print(dateChar);
    lcd.print(dateString);
    lcd.setCursor(0,1);
    lcd.print("log number");
    lcd.setCursor(12,1);
    lcd.print(String(logNumber));
    lcd.setBacklight(WHITE);
  }
  if(LCD_button==DOWN){
     //lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("A ");
    lcd.print(String(power.Ampere));
    lcd.setCursor(8,0);
    lcd.print("V ");
    lcd.print(String(power.Voltage));
    lcd.setCursor(0,1);
    lcd.print("P ");
    lcd.print(String(power.Power));
    lcd.setCursor(8,1);
    lcd.setBacklight(WHITE);
  }
  if(LCD_button==RIGHT){
    //lcd.clear();
    lcd.setBacklight(TEAL);
  }
  if(LCD_button==LEFT){
    //lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("file number");
```

```
      lcd.setCursor(12,0);
      lcd.print(String(fileNumber));
      lcd.setBacklight(WHITE);
    }
}
// ----------------------------------------------------

// Function to read which button is active
void setLCDbuttons(){      // read which button is active
  uint8_t buttons = lcd.readButtons();
  if (buttons) {
    //lcd.clear();
    if (buttons & BUTTON_UP) {
      lcd.clear();
      LCD_button = UP;
    }
    if (buttons & BUTTON_DOWN) {
     lcd.clear();
      LCD_button = DOWN;
    }
     if (buttons & BUTTON_LEFT) {
      lcd.clear();
      LCD_button = LEFT;
    }
     if (buttons & BUTTON_RIGHT) {
      lcd.clear();
      LCD_button = RIGHT;
    }
     if (buttons & BUTTON_SELECT) {
       lcd.clear();
      LCD_button = SELECT;
    }
  }
}
// ----------------------------------------------------
// ----------------------------------------------------

void setup(){
// This only runs one time after startup.
// We use many global varaibles, which are set up front

  lcd.begin(16, 2);  // set up the LCD's number of columns and rows:

  // Print a message to the LCD at startup
  lcd.setCursor(0, 0);
  lcd.print("Hello, world!");
  lcd.setCursor(0, 1);
  lcd.print("Hello, Cecilia!");
  lcd.setBacklight(WHITE);

  pinMode(SD_CS, OUTPUT); // set for the SD card on Uno
  Serial.begin(9600);
  Wire.begin();      // initiation for logger
  rtc.begin();
  sensors.begin();

  // Set the pins for input
  pinMode(PINAMPERE, INPUT);   // Set A and V for input pins
  pinMode(PINVOLTAGE, INPUT);   //

```

```
  fileNumber=0;                // number of writing to files
  NEWFILE=0;
  logNumber=0;
  power={0,0,0};
  power_now=power;
  read_numbers=0;
  prevWrite=0;


// Below is only if battery was off to set the date and time again. Uncomment to adjust.
// Check your PC date and time is correctly set. It adjusts to the date and time of the PC.
//rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));

  Serial.print("Initializing SD card...");

  // see if the card is present and can be initialized:
  if (!SD.begin(SD_CS)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    lcd.setCursor(0, 0);
    lcd.print("   no SD card!");
    while (1);
  }
  Serial.println("card initialized.");
// SD is successfully initialized
delay(1000); //1 second delay after setup, to ensure that all processes have time to begin.

}


// ---------------------------------------------------
// ---------------------------------------------------

void loop()
{
  // This runs continously in a loop.
  now = rtc.now();              // get current time

  nowMillis = millis();     // get current milliseconds since start

  power_accumulated = powerReadAndAccumulate(power_accumulated);   // get accumulated power values
  sensors.requestTemperatures();  //get temperature from the sensors
  ++read_numbers;     // increment number of readings
  delay(delayReading);


  if ((nowMillis - prevWrite) >= intervalWrite) {   // test for time for writing to file (logging time)
    power.Power=power_accumulated.Power/read_numbers;  // Average values. We have read read_numbers times
    power.Ampere=power_accumulated.Ampere/read_numbers;
    power.Voltage=power_accumulated.Voltage/read_numbers;


    read_numbers=0;            // reset counter for number of readings

    power_accumulated.Power=0;
    power_accumulated.Ampere=0;
    power_accumulated.Voltage=0;


```

```
    writeToFile();                  // write to file, power (averaged frpm readings) and temperatures (actual)
    updateLCD();                    // update lcd
    prevWrite = nowMillis;            // reset counter limit for next interval
    logNumber=logNumber+1;  // printed to file, starting on zero for each file
    // Showing data to the Serial monitor, if flag is defined at the top of the file
    // The format is different than for the SD card, which needs to be suitable for plotting
    if (SERIAL_PRINT_ON){
      Serial.print(dateString);
      Serial.print("  ");
      Serial.print(timeNow);
      Serial.print(" ");
      Serial.print(String(logNumber));
      Serial.print(" ");
      for (uint8_t i = 0; i < 4; i++){
        Serial.print("  T");
        Serial.print(i+1);
        Serial.print(":");
        Serial.print(sensors.getTempCByIndex(i));
      }
      Serial.print("  A:");
      Serial.print(power.Ampere,2);
      Serial.print("  V:");
      Serial.print(power.Voltage,2);
      Serial.print("  P:");
      Serial.print(power.Power,2);
      Serial.println(" ");  // new line
    }
  }

    setLCDbuttons();  // check which lcd button

} ;   // continuous loop end

```