TECHNISCHE UNIVERSITÄT
CHEMNITZ

Department of Electrical Engineering and Information Technology

Chair of Measurement and Sensor Technology

Project Documentation

„Project Lab Embedded Systems “

Group: 27-A

Members: Sharad Sirsat (549787)

Animesh Mukherjee (558741)

Giriraj Pawar (551205)

Project: Real Time Measurement of Temperature on a System on Chip with Consideration of Ambient Temperature

Date: 2019-06-14

Contents

1. Abstract

In Wireless Sensor Networks (WSNs), the communication presents the most complex and computational functionality of the system. This creates an increasing demand for the microcontroller to process communication information. In fact, nodes communication is maintained via radio signal propagation and then, once a reliable communication link is established between nodes, relative information is exchanged. Hence, during nodes communication, continuous data exchange enables an increase on the microcontroller workload and therefore, the temperature of the System on Chip (SoC)/microcontroller may show a non-stable behaviour. Moreover, this variation may lead to certain changes in the obtained results of relative data measurement and in the stability of the system. For this, it is critical to study the internal temperature variation of the circuit. In this work, three panStamp nodes were used to investigate the effect of communication task on the hardware internal temperature and hence to give an overview on its effect on the signal propagation. PanStamp nodes use received signal strength (RSS) to estimate the power of communicated signal, where each node communicates an RSSI value to the base station via CCradio Communication Protocol. SI7021 temperature sensor is used to measure the temperature of the SoC of panstamp, and continuous measurement were carried out while transmitting RSSI packets. An android application enables to visualize and monitor the actual state of connected sensor nodes. To study the impact of the temperature, other ambient temperature measurement were carried out with Arduino board and DHT 11 and external SI7021 sensors.

2. Introduction

Sometimes it becomes difficult to manage when the sensor is deployed in some extreme condition site where human cannot reach, or it is difficult to visit there frequently. To rectify this kind of problems we always investigate the ways where we want to monitor the sensor data in real time from anywhere without any physical presence at that place. Also, sometime to study variation of temperature or behaviour of system on chip in at different temperature at real time we need real time temperature measurement.

Sometime, System can get into a problem due to undefined behaviour caused by extremely low or high temperature. To overcome this problem, system need to monitor the internal temperature of SoC and to take corrective action. Currently, Wireless sensor networks have been developed for machinery condition-based maintenance (CBM) as they offer significant cost savings and enable new functionality.[8] Currently, there is an increasing number of applications in various fields, where even slight temperature changes at the level of a few tens of millikelvin need to be captured either to dynamically update domain knowledge models supporting precise decision-making functionality or to avoid unacceptable errors [12]

Wireless sensors can be placed in locations difficult or impossible to reach with a wired system, such as rotating machinery and untethered vehicles. Here, three panstamp has been used for measuring the health of monitoring system of SoC over the cloud database. Continuous, accurate temperature measurement is challenging when performed in everyday environments.

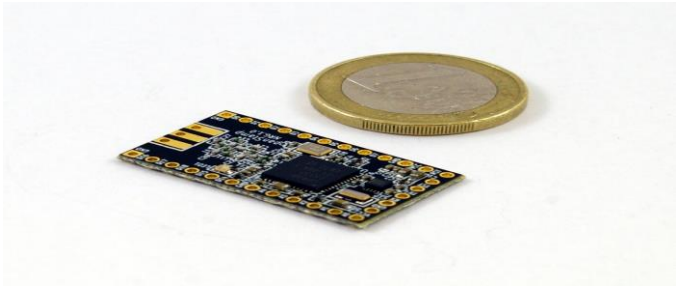


Figure 2.1 Panstamp NRG 2.0

3. Member Responsibilities

Name	Responsibilities
Animesh Mukherjee	• Studying CCradio communication protocol • Establish communication between panstamp and base station
Sharad Sirsat	• Sending temperature data to cloud and from Cloud to Android Application • Report writing
Giriraj Pawar	• Development of Android application • Database design

4. Functional Description

Core functionality of this system, the panstamp should be able to communicate between each other using CC1101 radio communication protocol. It is a protocol that allows the creation of a register-centred stack. From a practical point of view, we can say that the core of the stack is a table of register objects. This table consists of a mix of standard registers and custom ones. Moreover, the index of each register within the table matches the register ID. After getting data into serial monitor, it should be sent to cloud storage which in turn to Android application. Goal of the system design is to read the temperature data of SoC and then it should be able to send to cloud and further to android application for plotting of graph.

4.1 Functional Block Diagram

Following functional diagram has 3 panstamp, base station, cloud storage and Android application window. 3 Panstamp communicate to base station using CCRadio Communication, however, data to cloud storage and then to android application will be sent by running python script. We have used MIT app inventor to make an android application.

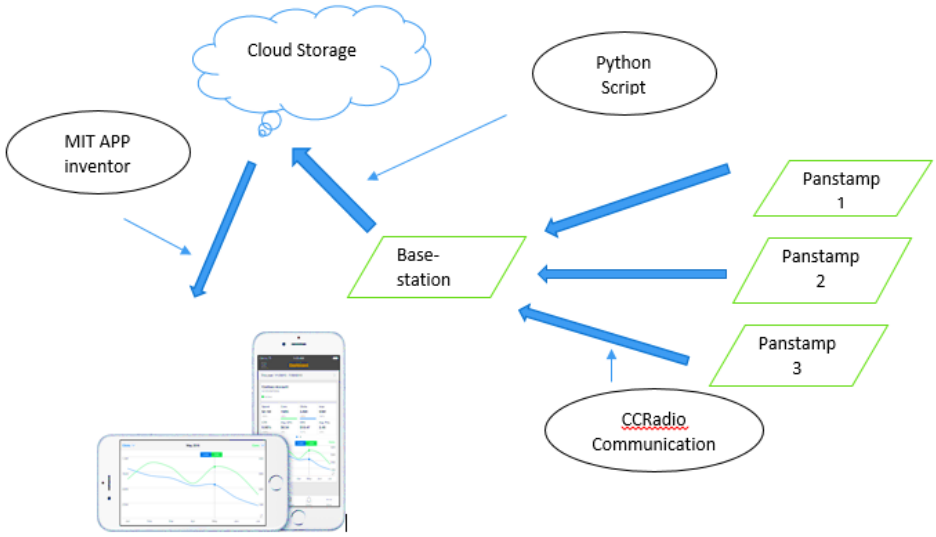


Figure 4.1 Functional block diagram of the proposed real time temperature measurement

4.2 Flowchart

Setup Phase:

As seen in the flowchart, Initially the sensor is initialized, and then pan-stamp RF Channel has been settled which updates the RF synchronization word and configuration registers. Then high-power transmission mode has been enabled. Further, CCPACKET length has been set.

Loop Phase:

In loop phase SOC temperature will be read by using I2C interface. Data field of CCPACKET has been updated which will give the information about packet destination address and Beacon address and IC Temperature.

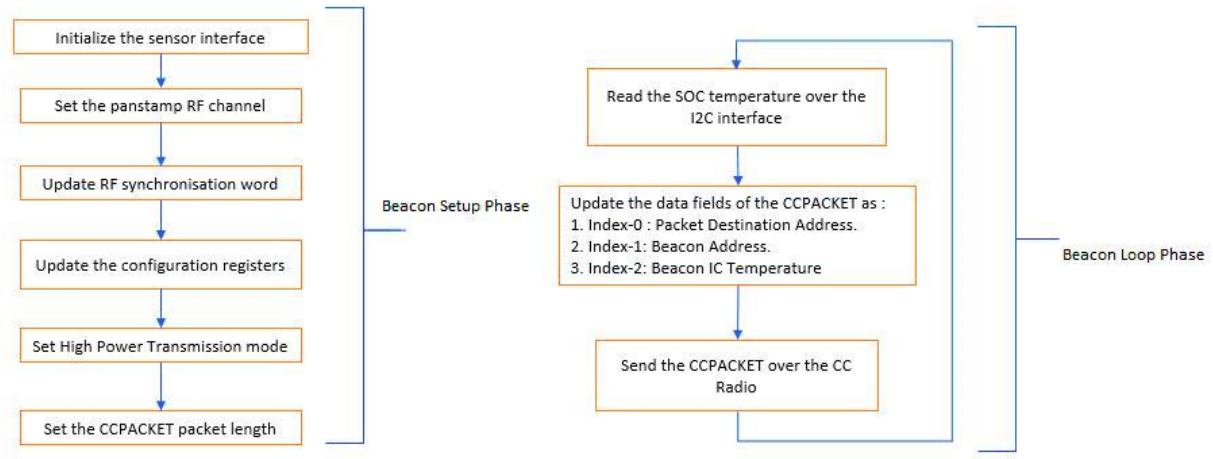


Figure 4.2 Pan-stamp transmitter (Beacon) flowchart

Modem Setup Phase:

In the receiver phase, serial COM port has been enabled, further device address will be set. Callback function needs to be attached to get an acknowledgement that packet has received.

Modem Loop Phase:

During loop phase, as packet received then pan-stamp will be turn on sleep mode which will switch off the modem reception. After that acknowledgment of packet data received will be sent which in turn off the pan-stamp sleep mode.

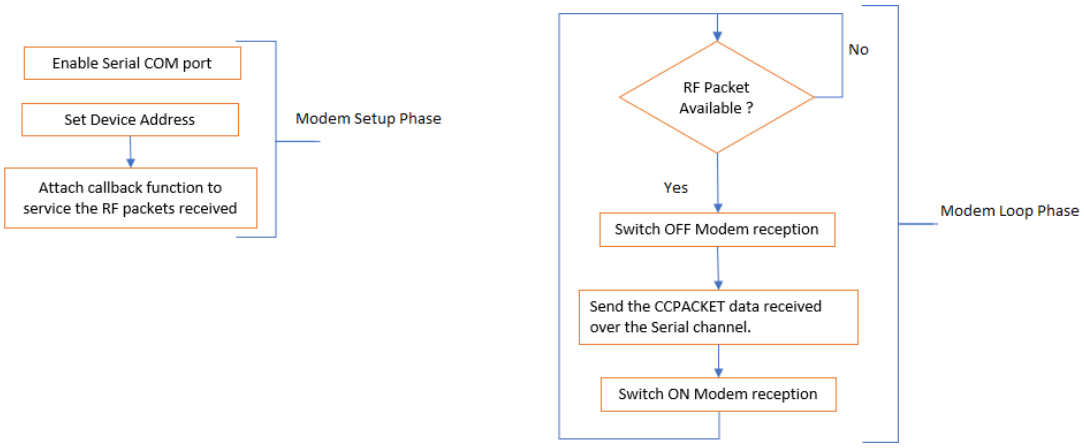


Figure 4.3 Pan-stamp Receiver (Modem) flowchart

5. Software

Software used in this project are Arduino IDE, MIT App inventor and Pycharm environment to program panStamp microcontroller.

5.1 Arduino IDE

The Arduino integrated development environment (IDE) enables the user to program the microcontroller in Arduino. It is written in JAVA and is available for Windows, Mac and Linux OS. It includes a text editor for writing code, a message window, a text editor, a toolbar with keys for mutual functions and a series of set menu. Programs written using Arduino Software (IDE) are called sketches. These sketches are written in the text editor and are saved with the file extension '. ino '[20]. Code structure in IDE is

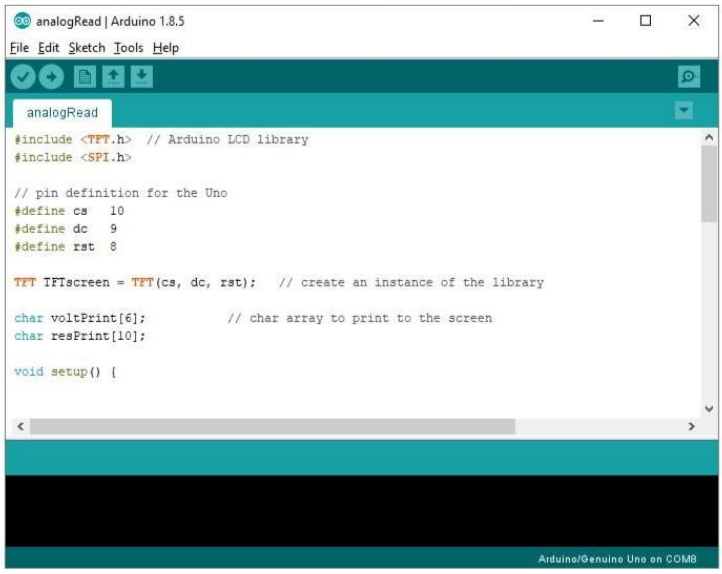


Figure 5.1 Android IDE interface

5.2 MIT app inventor for android application and Android SDK

MIT App Inventor [9] [10] [11] is an instinctive, visual programming environment that permits everybody to design useful applications for mobiles and laptops. App Inventor for Android is an open-source web application initially given by Google, and currently upheld by the Massachusetts Institute of Technology (MIT), which permits fresher's to computer aided programming to generate apps for the Android operating system (OS).

It offers a graphical border similar to Scratch and the Star Logo TNG user interface, which permits users to bead visual objects to makes an application that we can run on Android devices[10][11]. In making App Inventor, Google draw on important prior research in educational computing, as well as work which is done in Google on online development environments.

MIT App inventor 2 uses block-based coding method. It is developed by google and currently is maintained by MIT. It uses graphical interface to drag and drop visual objects to create application that can run on Android devices. MIT App inventor also provides cloud services like experimental Firebase DB component.

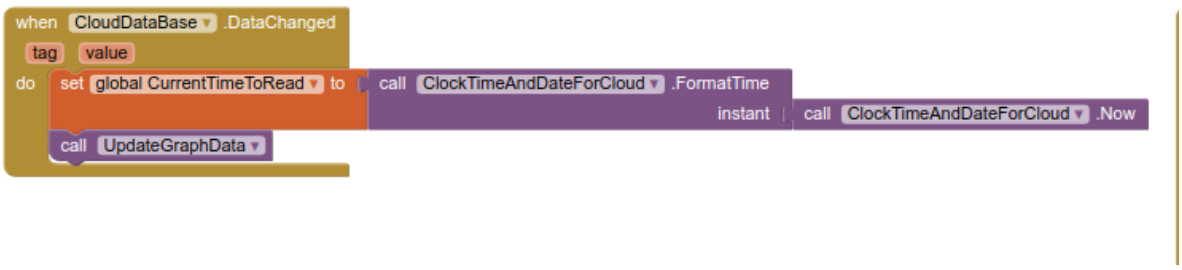


Figure 5.2 Data Changed Triggering

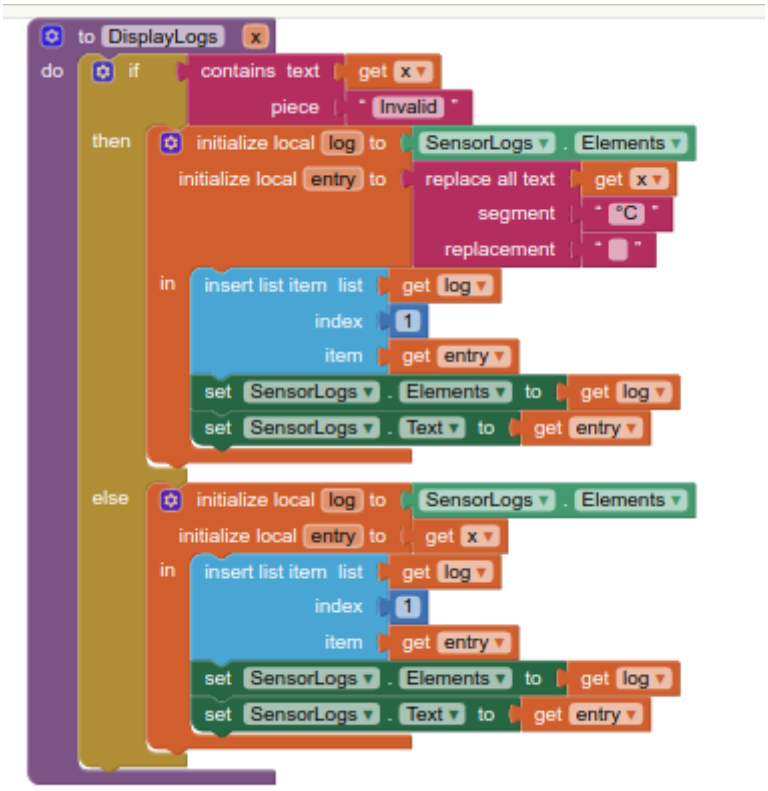


Figure 5.3 Display Logs

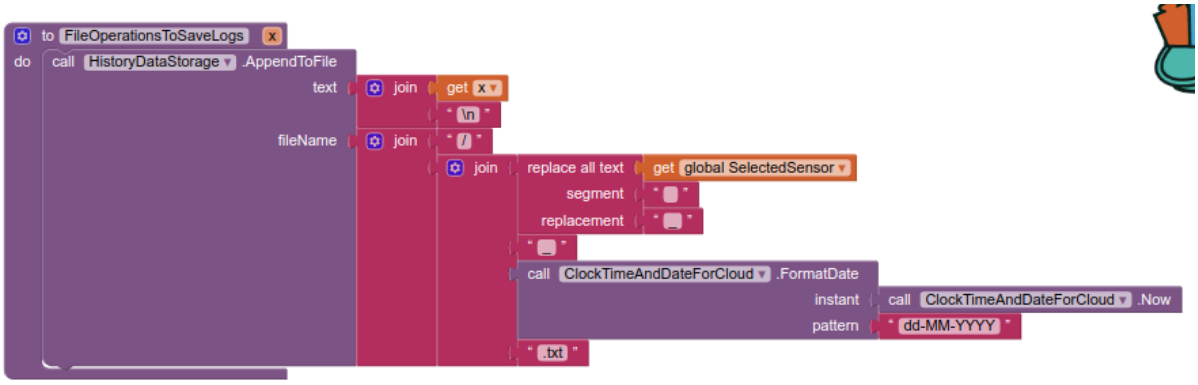


Figure 5.4 File Operation to save logs

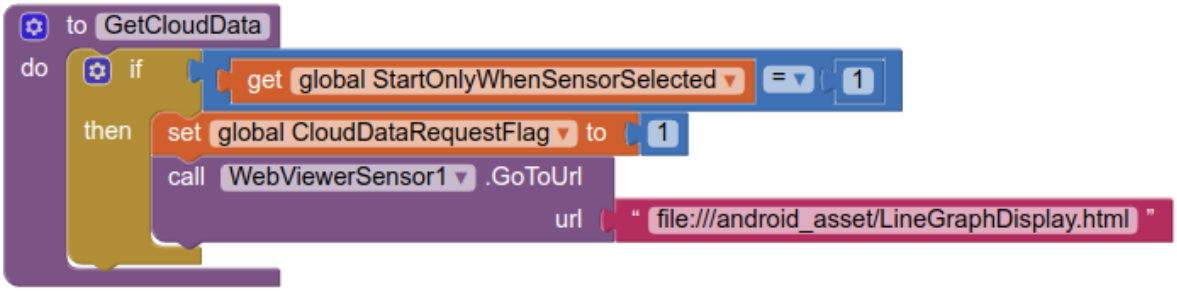


Figure 5.5 Get cloud data operations

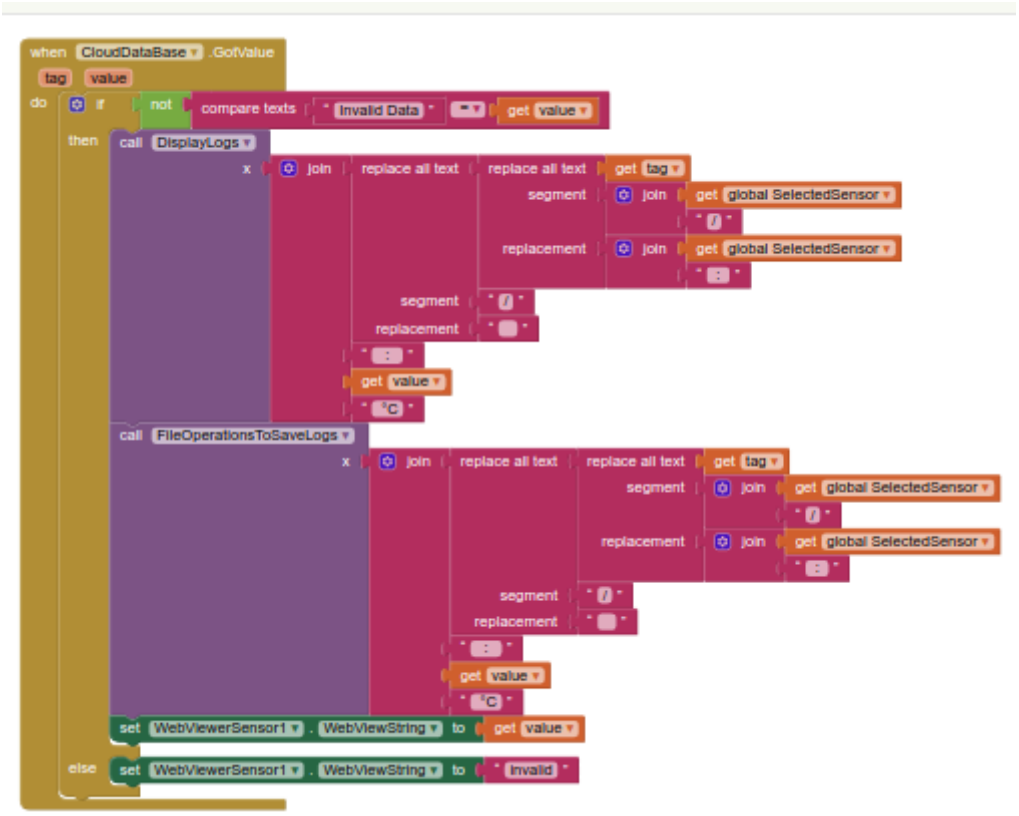


Figure 5.6 Data Value Reception

5.3 Firebase cloud service

Firebase provide a real-time list and backup as a competence.[12] The service gives application developers an Application Platform Interfaces that permits apps information to be synchronized transversely to clients and hoard on Firebase's cloud. Client reference library that allow incorporation with android, iOS, JavaScript, Java, Objective-C, Swift and Node.js applications. The database is also manageable through a REST Application Platform Interfaces and bindings for several JavaScript frame-works such as AngularJS, React, Ember.js and Backbone.js the REST API uses the Server-Sent Events protocol, which is an API for creating HTTP connections for receiving push notifications from a server. Developers using the real-time database can secure their data by using the company's server-side-enforced security rules. Firebase is the real-time cloud services provided by Google to read and write data in real time in the cloud. You can create nodes and update values to them just by using your Firebase project URL, in MIT app inventor we have just provided our Firebase project URL and Firebase Token to read and write data in the cloud

Firebase project URL: https://temperaturedatabase-47c41.firebaseio.com/

Firebase Token: AIzaSyCjaqDIxEpg53wLc32g-bNVS2lMIO7FiLQ

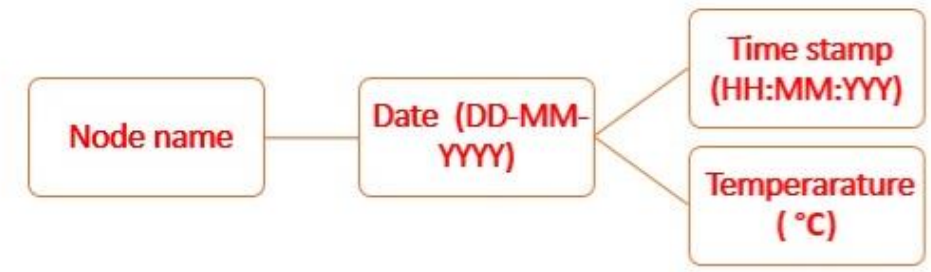


Figure 5.7 Data base model

Parsing of data sent to cloud storage has been done in following manner like a tree hierarchy. Firstly node will be created which explains from which sensor the data is coming from then we have date & timestamp, which gives details about the at what time the data was generated in the log files. Next is temperature, which is the exact value of the required temperature.

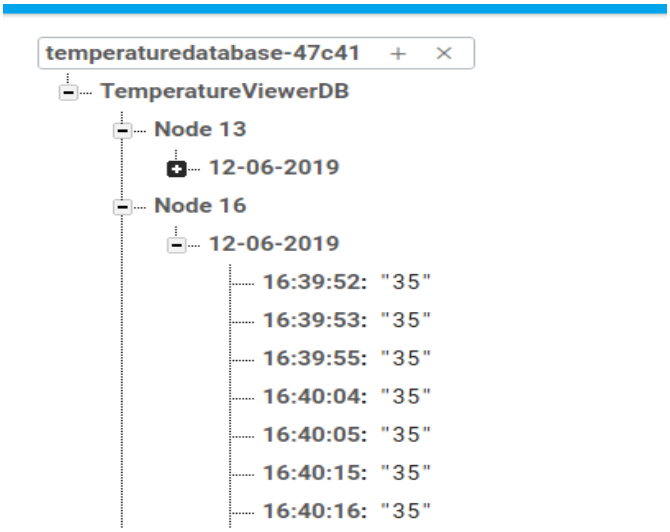


Figure 5.8 Firebase data model Interface

In the above figure we can see that likely hierarchy tree firstly there is node and then we have date and timestamp under that the actual data i.e. temperature is given.

5.4 Python3 Interpreter

Libraries used:

1) Pyre base library: It is a python3 wrapper library for the Firebase API.

2) Serial library: it is used to read the data from serial port

5.5 CC1101 Radio Communication Protocol

This is the API containing all basic core and radio functions. Panstamp is the main class and an object of this class, called panstamp, is automatically instantiated by the IDE. We need to point out that this API, the PANSTAMP class and the panstamp object do not

implement SWAP or any other wireless protocol. The following list shows the methods provided by this object.

6. Hardware

6.1 SI7021 Temperature Sensor

The Si7021 I2C Humidity and Temperature Sensor is a solid CMOS IC coordinating moistness and temperature sensor components, a simple to-computerized converter, signal handling, adjustment information, and an I2C Interface. The licensed utilization of industry-standard, low-K polymeric dielectrics for detecting mugginess empowers the development of low-control, solid CMOS Sensor ICs with low float and hysteresis, and amazing long-haul solidness. The stickiness and temperature sensors are processing plant aligned and the adjustment information is put away in the on-chip non-unpredictable memory. This guarantees the sensors are completely tradable, with no recalibration or programming changes required. The Si7021 is accessible in a 3x3 mm DFN bundle and is reflow weld capable. It very well may be utilized as an equipment and programming good drop-in update for existing RH/temperature sensors in 3x3 mm DFN-6 bundles, highlighting accuracy detecting over a more extensive territory and lower control utilization. The discretionary manufacturing plant introduced spread offers a position of safety, helpful methods for securing the sensor during get together (e.g., reflow patching) and for the duration of the life of the item, barring fluids (hydrophobic/oleo phobic) and particulates. The Si7021 offers an exact, low-control, manufacturing plant aligned computerized arrangement perfect for estimating dampness, dew-point, and temperature, in applications going from HVAC/R and resource following to mechanical and customer stages. Following is the temperature conversion formula for sensor SI7021:

$$Tempearature((°C) = \frac{175.72 * TempCode}{65536} - 46.85$$

Where: Temperature (°C) is the measured temperature value in °C Temp_Code is the 16-bit word returned by the Si7021

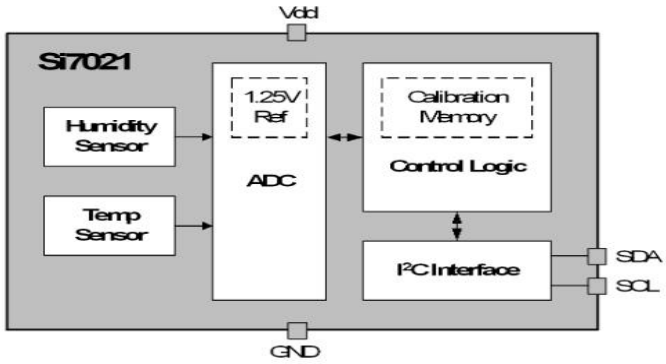


Figure 6.1 Functional Block Diagram of SI7021-A20

6.2 Panstamp & Panstick

PanStamp are a unit autonomous low-power wireless module programmable from the Arduino IDE and created for measurement and management modules. These modules communicate over the free 868-900-915 rate bands accessible worldwide and even years once power-driven from easy alkali batteries, counting on the duty cycle and transmission interval programmed. In addition, panStamps are completely wireless with direct access to the web and cloud storage. The corporate provides software package applications for configuring wireless service networks in turning any panstamp into an automation server. PanStamp conjointly provides custom-built Rasping pictures for Raspberry PI giving this fashionable pc platform a key role in any automation project as a wireless-IP entree with complete capabilities.

In this project, panstamp functionality used for communicating and sending the data to the cloud storage. In the panstamp we do have inbuilt SI7021 sensor which senses the temperature of internal MCU. Following are the functional specification of the panstamp:

- ARM Cortex-M4 processor
- LoRa/Lora WAN, NB-IoT, LTE-M or GPRS connectivity
- Power supply: 12VDC / 24VDC
- Binary inputs: 8 x opt isolated inputs capable to handle 12/24VDC signals
- Analog inputs: 2 x 4-20 mA inputs
- Binary outputs: 2 x open collector outputs ready to drive external relays
- Internal temperature sensor
- Other communication ports: RS485 Modbus RTU/ASCII
- Programmable from the Arduino IDE via USB
- Support for external displays

Then we have used 1.5V batteries for giving a power supply to the panstamp. Firebase database has been used for storing the data and further the data is been sent to android application for plotting a real time graph.
PanStamps has on-board microcontroller so that they will be run autonomously with non-external processor. They are 2 completely variant models of panStamp, every employing a totally different microcontroller pan-stamp NRG modules share constant footprint associated similar pinout because the AVR model however use a MSP430 core rather than an Atmega MCU. These modules were specially designed to be programmed from Arduino one.6. Link to the panStamp NRG page

PanStamps AVR and NRG will communicate with one another since they use constant radio core -- the popular CC1101 radio created by TX Instruments. This implies that any node exploitation this radio will be part of any existing network of panStamps. Since all panStamps resources (libraries, protocol, software system tools hardware styles and sample applications) square measure open supply, the stack may be ported to any MCU platform or SOC.

Unlike alternative wireless modules, panStamps were born to interoperate with styles from alternative developers and additionally with nodes created by alternative makers. In contrast to alternative protocols, SWAP, the obtainable protocol for panStamp, is easy, compact and powerful. It may be enforced even in tiny microcontrollers with very little Flash and RAM.

PanStamp is an entire low-power wireless IoT platform. the total answer is being employed in many various comes, together with building automation, energy watching, waste management, mining, transport and art. The platform may be extended with new styles, compatibility with third-party applications and communication technologies. Custom Python scripts may be hosted by computers connected to the low-power wireless network and knowledge may be pushed to remote servers and services via panStamp's software system entrance.

PanStamps may be used over distances of many meters in open areas, or over kilometres once exploitation panStamp's Long Distance carrier boards. This makes panStamp the right platform either for straightforward home automation comes or for giant good town deployments.

The magic behind panStamp is that users will develop new devices and deploy wireless networks terribly quickly. Albeit it's a low-priced answer, panStamp isn't an easy hardware platform. The subsequent chapters can try and justify why panStamp is exclusive in providing low-power consumptions, flexibility, ability, openness and simple use at a fraction of the value of alternative business alternatives.

Figure 6.2 Panstamp

Panstick may be a special USB-UART entree designed for panstamp a pair of. It provides the mandatory footprint to incorporate a panStamp (NRG or AVR) and may even be used while not a wireless module, by stacking or plugging below or onto alternative carrier boards. This entree is used as either a serial engineer and as a serial electronic equipment for wireless networks.

This version will program either panStamp AVR a pair of or NRG a pair of. By setting the on-board selector to the acceptable position, you may be ready to place the panstick into one in every of these modes:

- Left position (PR label): program and run applications on panStamp NRG a pair of. This position will not work once victimisation the module serially from a serial terminal or as an electronic equipment, because of the special use of the RESET line (without a series capacitor).
- Right position: program and run applications on AVR modules. This position also can be accustomed run NRG applications together with serial applications (USB).

Apart from victimisation panstick to program panStamp, it is used as a serial interface to speak with wireless networks from a serial terminal SWAPdmt, lagarto-swap, fhem, Open or your custom software system. So as to try and do this, you have got to program your on-board panStamp with our electronic equipment sample application and place the manual put on the proper position (opposite to "PR"). You'll examine our serial protocol and AT commands here.

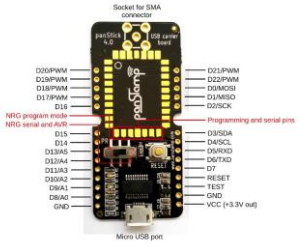


Figure 6.3 Panstick

7. Realization

The entire project can be classified into 3 steps: Reading temperature data from sensor, sending data to cloud storage, Receiving Data from cloud storage to android application. Simply all these steps compromise of the communication of panstamp and plotting of temperature graph on android application. Generally, each program has different task. For that case every program is set of instruction that are written to vary the state of hardware and check the temperature reading at that instance

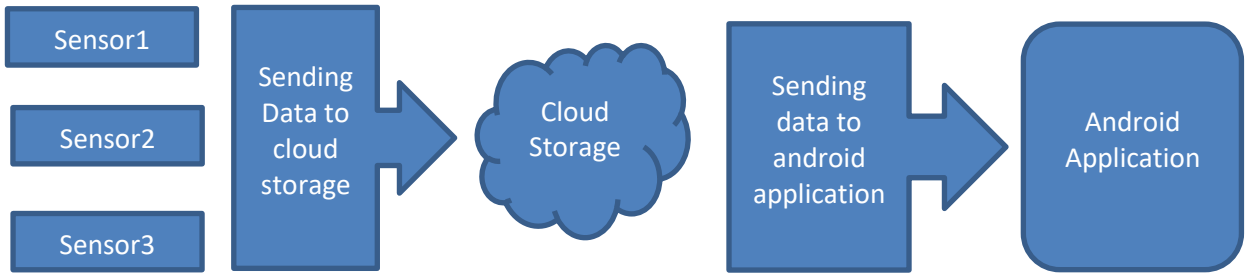


Figure 7.1 Steps involved in realization of system

7.1 System design

Three panstamp are installed and are communicating with the modem (Panstamp which is attached with panstick) the modem acts as receiver, as it only receives the temperature date that these 3 panStamps are sending to modem. Modem or base station will always act as listener here. The 3 panstamp and Modem are flashed using Arduino IDE for providing the internal temperature of SOC. After flashing and providing power supply to 3 panstamp using batteries. The communication between modem (base station) and 3 panstamp. Whatever the temperature reading these 3 panStamps are sending will be stored in cloud "firebase". When we get a data on cloud storage, we will run the python script for sending data from cloud to android application. Whenever the android application gets the temperature reading data it will plot the graph according to it.

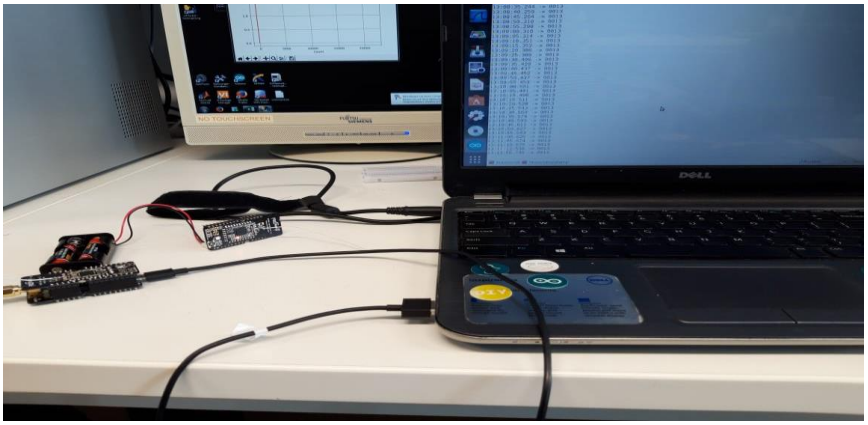


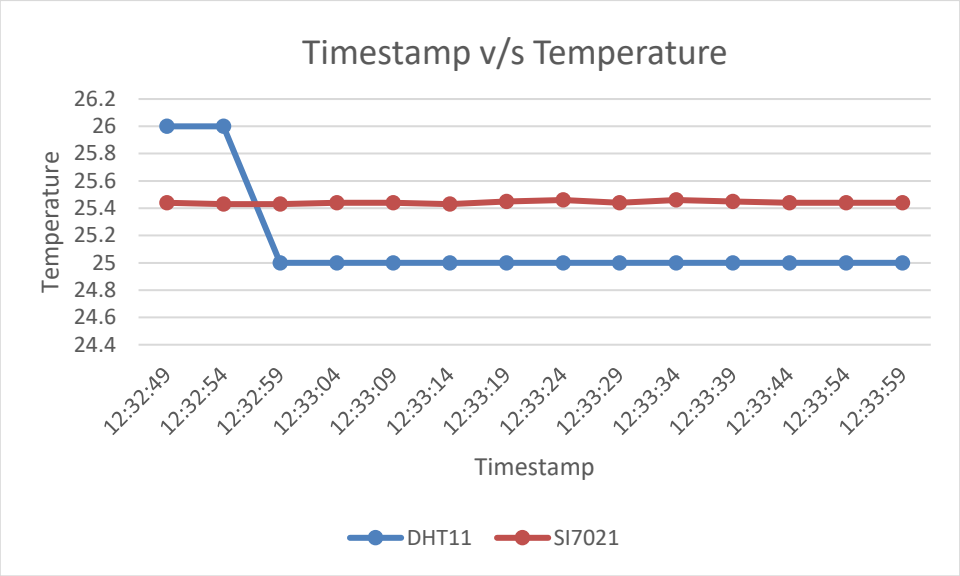
Figure 7.1 System Setup

7.2 Temperature sensor reading

SI7021 Sensor Reading with Arduino:

Timestamp	Sensor	Temp(°C)
12:32:49	SI7021	25.44
12:32:49	DHT11	26.00
12:32:54	SI7021	25.43
12:32:54	DHT11	26.00
12:32:59	SI7021	25.43
12:32:59	DHT11	25.00
12:33:04	SI7021	25.44
12:33:04	DHT11	25.00
12:33:09	SI7021	25.44
12:33:09	DHT11	25.00
12:33:14	SI7021	25.43
12:33:14	DHT11	25.00
12:33:19	SI7021	25.45
12:33:19	DHT11	25.00
12:33:24	SI7021	25.46
12:33:29	DHT11	25.00
12:33:29	SI7021	25.44
12:33:34	DHT11	25.00
12:33:34	SI7021	25.46
12:33:39	DHT11	25.00
12:33:39	SI7021	25.45
12:33:44	DHT11	25.00
12:33:49	SI7021	25.44
12:33:54	DHT11	25.00
12:33:59	SI7021	25.44
12:33:59	DHT11	25.00
12:34:04	SI7021	25.44

Graph:



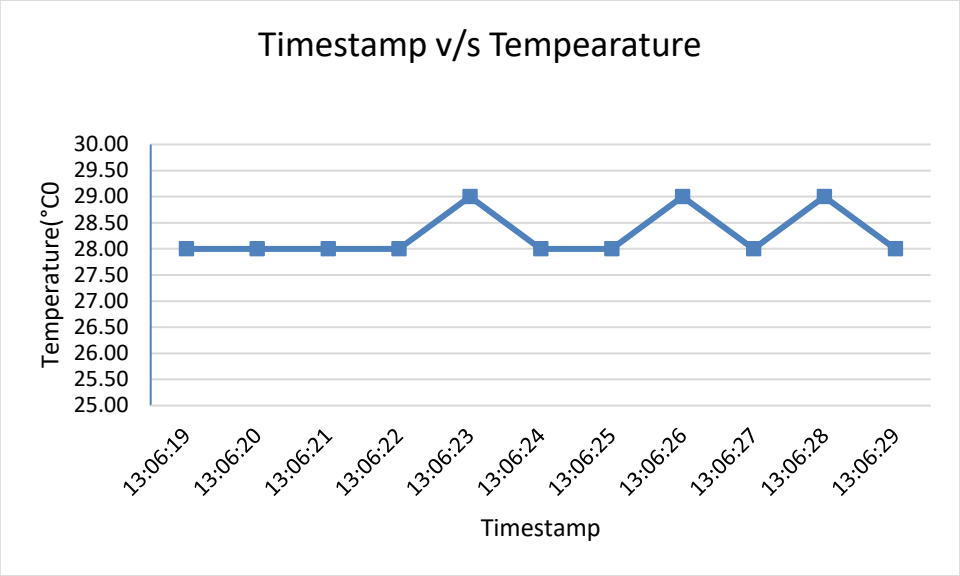
Time v/s Temperature plot for SI7021 & DHT with Arduino

Considering 3 scenarios for taking reading:

Panstamp kept under Normal Room Temperature:

Node	Date	Time	Temperature
13	02-07-2019	13:06:19	28 °C
13	02-07-2019	13:06:20	29 °C
13	02-07-2019	13:06:21	28°C
13	02-07-2019	13:06:22	28°C
13	02-07-2019	13:06:23	29°C
13	02-07-2019	13:06:24	28°C
13	02-07-2019	13:06:25	28°C
13	02-07-2019	13:06:26	29 °C
13	02-07-2019	13:06:27	28 °C
13	02-07-2019	13:06:28	29 °C
13	02-07-2019	13:06:29	28°C
13	02-07-2019	13:06:30	28 °C

Graph:

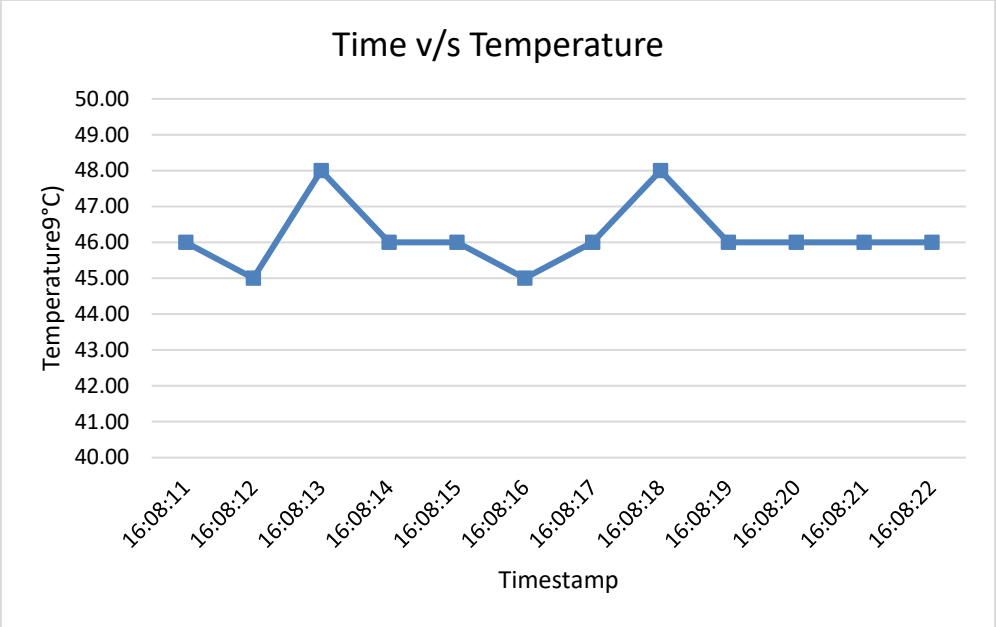


Time v/s Temperature plot for sensor kept under normal temperature

Panstamp Kept under High temperature:

Node	Date	Time	Temperature
13	02-07-2019	13:08:21	46°C
13	02-07-2019	13:08:22	45°C
13	02-07-2019	13:08:23	48°C
13	02-07-2019	13:08:24	46°C
13	02-07-2019	13:08:25	46°C
13	02-07-2019	13:08:26	45°C
13	02-07-2019	13:08:27	46°C
13	02-07-2019	13:08:28	48°C
13	02-07-2019	13:08:29	48°C
13	02-07-2019	13:08:30	46°C
13	02-07-2019	13:08:31	46°C
13	02-07-2019	13:08:32	46°C

Graph:

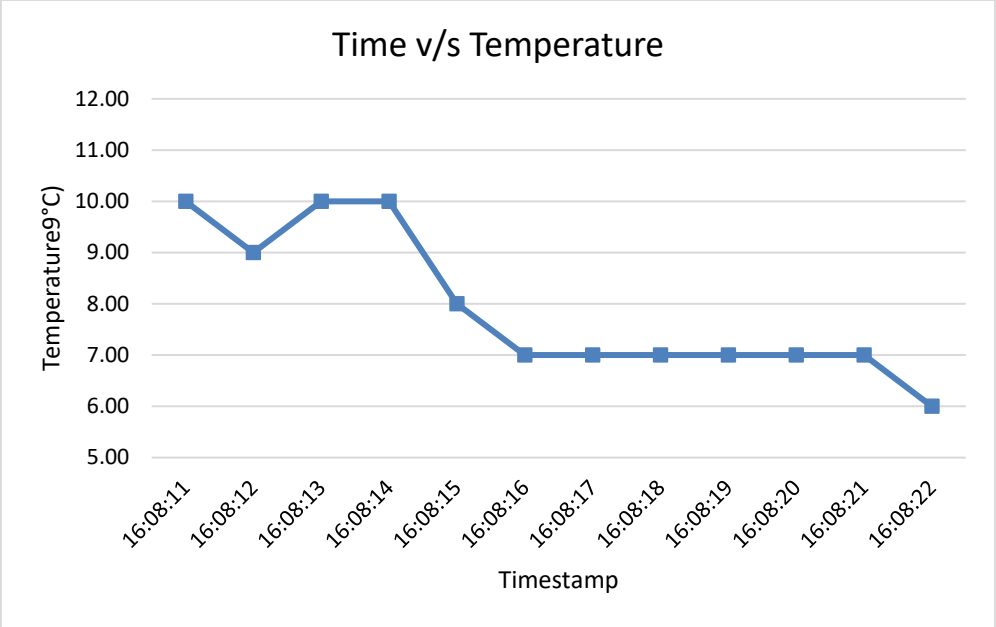


Time v/s Temperature plot for sensor kept under high temperature

Panstamp Kept under Freezing temperature:

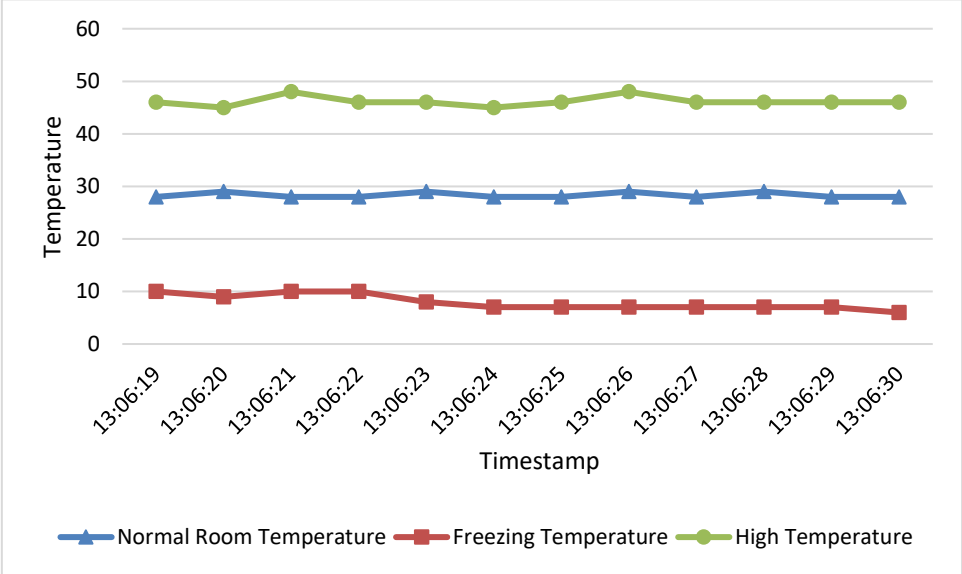
Node	Date	Time	Temperature
13	02-07-2019	16:08:11	10°C
13	02-07-2019	16:08:12	9°C
13	02-07-2019	16:08:13	10°C
13	02-07-2019	16:08:14	10°C
13	02-07-2019	16:08:15	8°C
13	02-07-2019	16:08:16	7°C
13	02-07-2019	16:08:17	7°C
13	02-07-2019	16:08:18	7°C
13	02-07-2019	16:08:19	7°C
13	02-07-2019	16:08:20	7°C
13	02-07-2019	16:08:21	7°C
13	02-07-2019	16:08:22	6°C

Graph:



Time v/s Temperature Graph for sensor kept under freezing temperature

Comparison of all 3 Scenarios:



Time v/s Temperature Graph for all 3 scenarios

7.2 Android application

When Firebase database is updated with new temperature value then ".DataChanged" event will be triggered, later in".GotValue" code block "DisplayLogs", "FileOperationsToSaveLogs" functions are called and latest value of temperature will be plotted by "WebViewer" module. Please refer below steps to use an Android application.

1) Select the node from node list

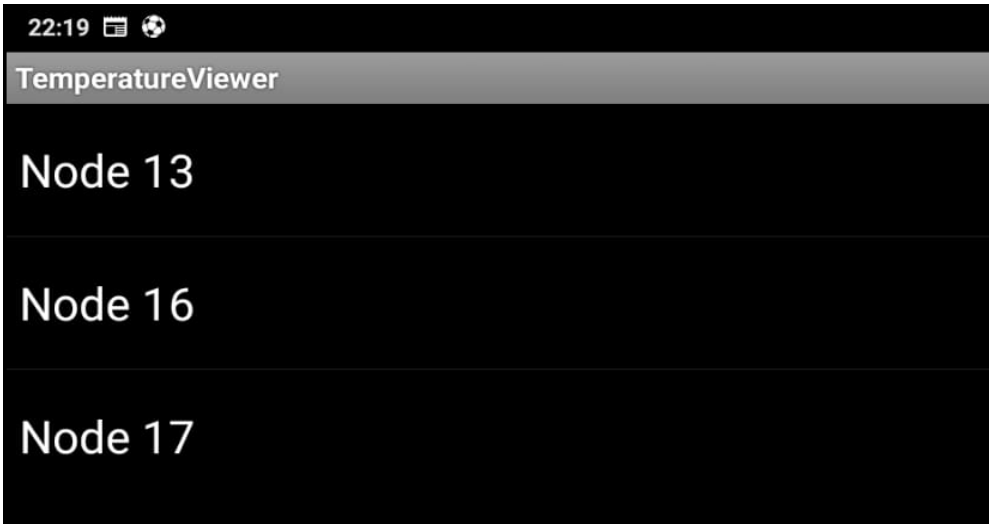


Figure 7.3 Node list interface

2) Click on Start Button to start the real time temperature plot

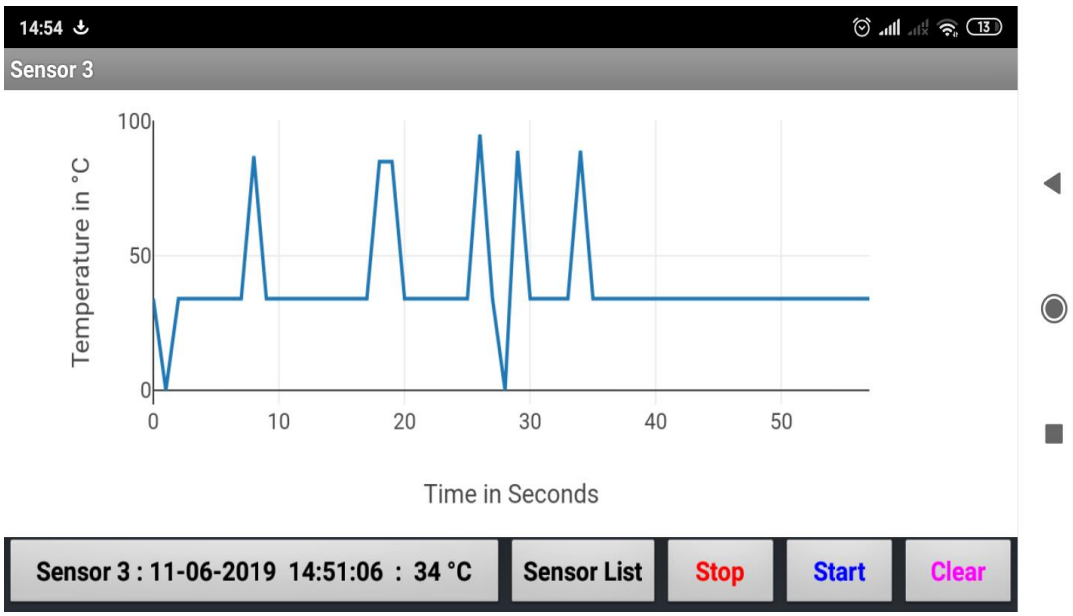


Figure 7.4 Graph Interface

3) Click on the logs tab to view the temperature logs with time stamps

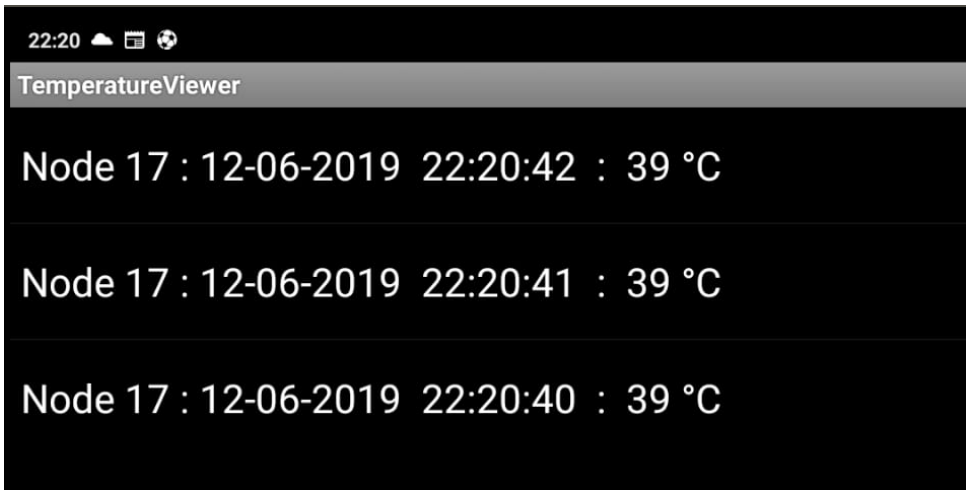


Figure 7.5 Data Log interface

4) All the logs are stored in the android phone root folder '/'.

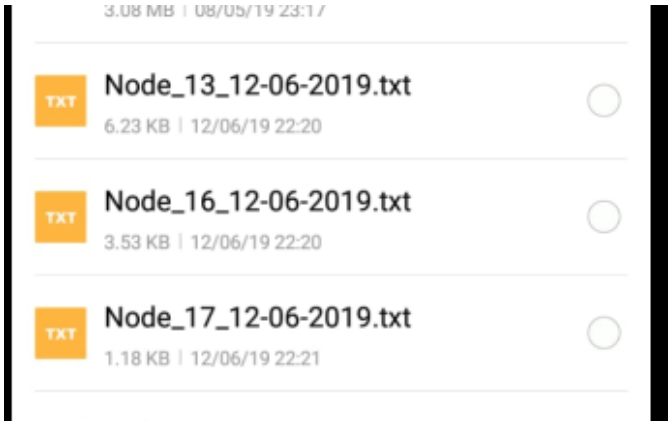


Figure 7.6 Node data textile storage

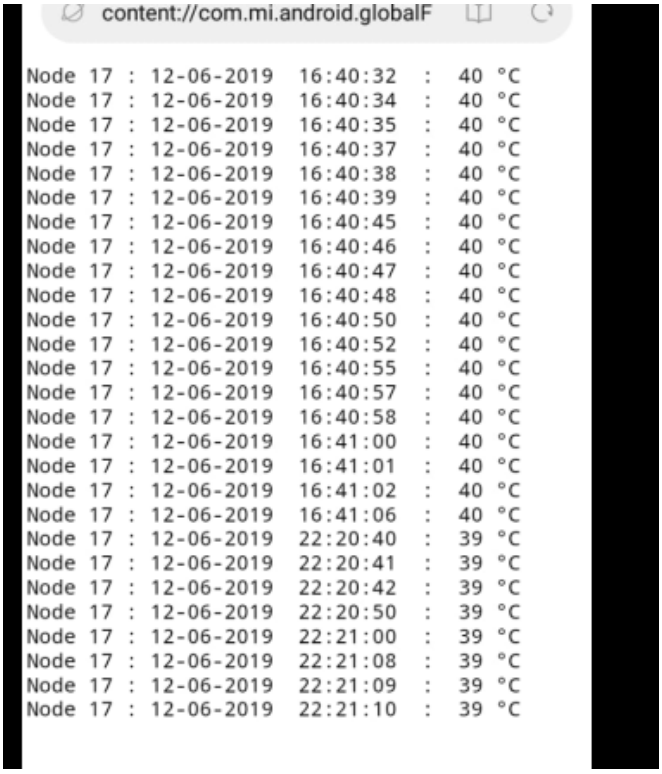


Figure 7.7 Node-data Interface

7.4 Discussion and Problems incurred

1. SWAP Protocol was not any more supported and maintained.

Initially starting with SWAP communication protocol for communicating between panstamp but as we went deeper into understanding the SWAP protocol, we got to know that we were

unable to establish the communication between panstamp and base station. Studying further and after few more research communicated with panstamp manufacturer and got to know that SWAP protocol was not any more supported and maintained. So as a solution to this CC1101 Radio communication protocol can be used and then were able to establish the communication between panstamp and base station.

2. Reading data from serial port and uploading to the cloud.

As serial UART has been used for reading the temperature reading from the sensor and were able to read the data serially from the sensor on serial monitor. Then, after reading data on serial monitor, need it to send it to cloud storage. But as the serial port was busy in reading the data and to send the data which can be read serially through serial port and need a serial port. As both functions couldn't be done simultaneously, we were facing an issue in sending data to cloud storage. For resolving this issue, did some research and found, didn't need serial monitor the data and able to read the temperature data directly using python script. So, solution was to make made a python script.

3. Synchronization of Real-time data.

Going further, as we established the communication between panstamp and base station we tested with different temperature reading and that too in different environment condition. We were successful in getting all the temperature correctly, but now we were facing an issue with data synchronization i.e. data which we were getting from temperature sensor and plotting it on graph. Between these there were no synchronization. For resolving this issue we made some changes in our python script and issue got resolved.

4. Use of plotly.js JavaScript library to plot a real time graph

As we were aiming to plot the graph for every 5 Seconds, but the sensor sensitivity wasn't too much and also, we were using serial port, so we were getting data serially. So, data wasn't getting plotted for every 5 seconds, it was getting plotted for random values and seconds. So, to resolve this issue we decided to take reading for every second and made some changes in the python script. The issue was got resolved.

5. Temperature formula conversion ambiguity.

Finally, when we were able to plot graph, and everything was going properly we were facing an issue with the temperature reading. We were getting some undesirable temperature reading. To resolve this, we went through the data sheet of SI7021, Adafruit library, Sparkfun libraries still we were facing the issue. After putting lot of efforts, we asked manufacturer regarding this and what we observed that we were missing the SI7021 sensor from the panstamp so to fix this we used the panstamp which has inbuilt SI7021 sensor and issue got resolved.

8. Conclusion

With the effort put in this project, so have enough reason to conclude, the use of panStamps for measuring real time temperature is precise or sensitive to a given change in SOC temperature while communicating with other panstamp or based on behavior of SOC in different environment. The use of CC1101 radio communication is efficient as Simple wireless abstract protocol was not any more supported and maintained. Also, the storing data on cloud storage is also important if we wanted to study the behavior of SOC under different

circumstances. As we are using android application to monitor the graph, it's easy to study and observe the behavior of SOC for different condition. There is still room for improvement in this for example, more accurate data can be obtained if the sensitivity of sensor is increased. Also, with plotting a graph of temperature we can plot a graph of RSSI value i.e. strength of signal that the base station is receiving. Improvement in the Android application user interface can be done.

9. Description of Files

Folder Structure: Group 27A

9.1 Project Documents

Project_Documentation.docx	Project Documentation
Final_Presentation.pptx	Final Presentation Slides

9.2 Project Files

Modem.ino	Source Code for Arduino IDE
Beacon_12.ino	Source Code for Arduino IDE
Beacon_13.ino	Source Code for Arduino IDE
Beacon_14.ino	Source Code for Arduino IDE
Test.py	Python Script for pycharm IDE

10. References

[1] Si7021 temperature sensor, Available: https://www.silabs.com/documents/public/data-sheets/Si7021-A20.pdf [Accessed May 12, 2019]

[2] Panstamp nodes, Available: http://www.ply.me.uk[Accessed May 12, 2019]

[3]McRoberts Michael, Beginning Arduino, Apress, 2010, Available: https://www.apress.com/gp/book/9781430250166 [Accessed May 12, 2019]

[4]Temperature and humidity measurement with Si7021, https://github.com/panStamp/panstamp/wiki/Temperature-and-humidity-measurement-with-SI7021-I2C-sensor [Accessed May 14, 2019]

[5] PanStamp-NRG3.-Technical-details, Available: https://github.com /[Accessed May 14, 2019]

[6] Panstamp forum, Available: http://www.panstamp.org/forum/ [Accessed May 15, 2019]

[7] Data-to-firebase-database-using-esp8266, Available: https://circuitdigest.com [Accessed May 19, 2019]

[8]https://www.google.com/search?q=si7021+temperature+formula&oq=si7021+temperature+formula&aqs=chrome..69i57j69i60.216j0j7&sourceid=chrome&ie [Accessed May 21, 2019]

[9] MIT App inventor ,Available: http://ai2.appinventor.mit.edu [Accessed May 21, 2019]

[10] Ai2.appinventor- components, Available: http://ai2.appinventor.mit.edu[Accessed May 23, 2019]

[11] MIT app inventor, Available:https://groups.google.com/forum/#!forum/mitappinventortest [Accessed May 25, 2019]

[12] Firebase tutorial, Available: https://rominirani.com/firebase-iot-tutorial-46203a92f869 [Accessed May 28, 2019]

[13] Plot of Graph, Available: https://plot.ly/javascript/getting-started/#start-plotting [Accessed May 28, 2019]

[14] H. Preston-Thomas, "The International Temperature Scale of 1990 (ITS-90)," Metrologia 27, 3–10 (1990). ,Available: https://iopscience.iop.org/article/10.1088/0026-1394/27/1/002/pdf [Accessed May 10, 2019]

[15] L. A. Guildner and W. Thomas, "The Measurement of SOC Temperature," Temperature. Its Measurement and Control in Science and Industry (American Institute of Physics, New York, 1982), Vol. 5, pp. 9–19.

[16] M. Biebl, G. Brandl, and R. T. Howe, "Young's modulus of insitu phosphorus-doped polysilicon," in Proceedings of the 8thInternational Conference on Solid-State Sensors and Actuators, and Eurosensors IX, pp. 80–83, Stockholm, Sweden, Jun 1995.

[17] G. L. Pearson,W. T. Read Jr., andW.L.Feldmann, "Deformation and fracture of small silicon crystals," Acta Metallurgica, vol. 5, no. 4, pp. 181–191, 1957.

[18] E. A. de Vasconcelos, S. A. Khan, W. Y. Zhang, H. Uchida, and T. Katsube, "Highly sensitive thermistors based on high-purity polycrystalline cubic silicon carbide," Sensors and Actuators A:Physical, vol. 83, no. 1–3, pp. 167–171, 2000.

[19] J. B. Casady, W. C. Dillard, R. W. Johnson, and U. Rao, "A hybrid 6H-SiC temperature sensor operational from 25◦C to 500◦C," IEEE Transactions on Components, Packaging, and Manufacturing Technology: Part A, vol. 19, no. 3, pp. 416–422 .

11. Annex

11.1 Source Code

FinalDelivery.zip

Core Function for CC1101 Radio Communication

These functions (methods) are obtainable from the panstamp object. They are primarily related to the power management and initialization of the module.

Init

void init(uint8_t freq) //deprecated in API 2.0

Description:

It will initialize board and peripherals. This method is mechanically called by Arduino so you we don't have to do it unless we want to set a centre frequency different than 868 MHz. This method is deprecated as of the new API 2.0 and Carrier Frequency should only be changed in "panstamp.h" or use panstamp.radio.setCarrierFreq(CFREQ_915);

Parameters

freq : carrier frequency (CFREQ_868, CFREQ_915, CFREQ_433, CFREQ_918). Default frequency is CFREQ_868 if the value is omitted.

Example

panstamp.init(CFREQ_915); // Init panStamp at 915 MHz as the center frequency

---

reset

void reset(void)

Description

It will be used for restarting panStamp board.

Parameters

None

Example

panstamp.reset(); // Restart module

---

rxOn

void rxOn(void)

Description

It enables RF receiver. Receiver is enabled by default. This method is infrequently used since radio.sendData() and radio.receiveData() already handle this option by themselves.

Parameters

None

Example

panstamp.rxOn(); // RF reception enabled

---

rxOff

void rxOff(void)

Description

It disables RF receiver. This avoids the MCU which is interrupted on each wireless packet received. This method is infrequently used since radio.sendData() and radio.receiveData() already handle this option by themselves.

Parameters

None

Example

panstamp.rxOff(); // RF reception disabled

---

sleep

void sleep(void)

Description

This function allows panstamp to enter sleep mode forever. Only a pin interrupt can wake-up the module and no RF packet can wake the module. A fix should be on the way involving Wake-on-Radio state. This mode is the one with the lowest current consumption (1 uA). After waking-up, the program continues with the following instruction after sleep (). This instruction can be typically called from loop (). Calling this function or any sleep related functions will (re)enable all interrupts.

Parameters

None

Example

panstamp.sleep(); // Enter sleep mode

---

sleepSec

void sleepSec(uint16_t time)

Description

It allows panstamp to enter sleep mode for a given period of time. This mode is the one with the lowest current consumption (under 1 uA). After this period has elapsed, the module will automatically wake-up and continue with the following instruction. This instruction can be typically called from loop ().

Parameters

Time: Sleep time in seconds

Example

panstamp.sleepSec(5); // Sleep 5 seconds

---

wakeUp

void wakeUp(void)

Description

Wake-up from the sleep mode. This function needs to be called from the ISR (Interrupt Service Routine) having interrupted the module from its sleeping state.

Example

```
void myISR()
{
  panstamp.wakeUp();
}
```

---

attachInterrupt

inline void attachInterrupt(void (*funct)(CCPACKET*))

Description

It specifies a named Interrupt Service Routine (ISR) to call when a wireless packet is received. Replaces any previous function that was attached to the interrupt. This instruction allows declaring custom call back functions where to parse incoming packets from. Interrupts can originate from RF, Timer or pins (P1.x and P2.x only). See Low power mode for additional important notes.

Parameters

funct : callback function with the following prototype: void myISR(CCPACKET *packet), where packetis the wireless packet received.

Example

```
void myISR(CCPACKET *packet)
{
  // Do whatever we need with packet->data[], packet->length, packet->crc, packet->rssi and packet->lqi`
}
void setup()
{
  ...
  panstamp.attachInterrupt(myISR);  // Declare custom ISR
  ...
}
```

---

delayClockCycles

void delayClockCycles(uint16_t n)

Description

Creates a delay based on the MCU's clock cycle for fine precision timings. The source of the clock cycle is MCLK.

Parameters

n : 2-byte clock cycles to wait (2-byte hex number/65535 decimal number)

Example

panstamp.core.delayClockCycles(0xFFFF); // 65535 clock cycles x f_MCLK = delay in sec

---

getTemp

int getTemp(void)

Description

This function returns the MCU/Ambient temperature (temp) in ºC. It's not the most accurate/stable way to measure ambient temperature, but it's there.

Parameters

None

Example

unsigned int temp = panstamp.core.getTemp();

Serial.print("Temperature in degrees Celsius: ");

Serial.println(temp);

Example (more accurate)

int DegC = ((analogRead(A10) - 673) * 423) / 1024; //Celsius

int DegF = ((analogRead(A10) - 630) * 761) / 1024; //Fahrenheit

Serial.print("Internal MCU Temperature: ");

Serial.print(DegC); Serial.println("*C"); //Replace with desired scale

---

Radio Functions

These functions (methods) are available from the radio object, owned by the panstamp object (panstamp.radio). This radio API manages everything related to RF communications (radio configurations, transmissions, receptions).

---

setChannel

void setChannel(uint8_t chnl)

Description

Configure RF channel. Typically from 0 to 9. Default value = 0.

Parameters

chnl : channel number (0..9)

Example

panstamp.radio.setChannel(0); // Use channel 0

---

setCarrierFreq

void setCarrierFreq(uint8_t freq)

Description

Configure carrier frequency. Typically from CFREQ_433, CFREQ_868, CFREQ_915 and CFREQ_918. Default value = CFREQ_868.

Parameters

freq : frequency (CFREQ_433...CFREQ_918)

Example

panstamp.radio.setCarrierFreq(CFREQ_433); // Use 433 Mhz frequency

---

setSyncWord

void setSyncWord(uint8_t *sync)

void setSyncWord(uint8_t syncH, uint8_t syncL)

Description

Configure RF synchronization word. This word can also be taken as a kind of network ID. Default value = 0xB547.

Parameters

sync : 2-byte synchronization word (2-byte array)

or

syncH : synchronization word. High byte

syncL : synchronization word. Low byte

Example

panstamp.radio.setSyncWord(0x12, 0x23); // Sync word = 0x1223

---

setDevAddress

void setDevAddress(uint8_t addr)

Description

Set physical 1-byte device address for RF communications. If address check is enabled in the device then only packets addressed to our device or broadcasted (destination address = 0) will be received.

Parameters

addr : 1-byte device address (1..255). Address 0 is reserved for broadcasts.

Example

panstamp.radio.setDevAddress(21); // Device address = 21

---

disableAddressCheck

void disableAddressCheck(void)

Description

Disable address check. This enabled the radio to receive any packet transmitted in the same RF channel and with the same synchronization word regardless of the destination address of the packet.

Parameters

None

Example

panstamp.radio.disableAddressCheck(); // Disable address check

---

enableAddressCheck

void enableAddressCheck(void)

Description

Enable address check. This disables the radio from receiving any packet not directly addressed to it or not broadcasted.

Parameters

None

Example

panstamp.radio.enableAddressCheck(); // Enable address check

---

sendData

bool sendData(CCPACKET packet)

Description

Transmit packet. Return true if transmission is completed. Return false otherwise.

Parameters

packet : CCPACKET to be transmitted

Example

```
CCPACKET packet;  // CCPACKET pbject

packet.length = 10;  // packet length = 10 bytes

// Build packet payload (00 01 02 03 04 05 06 07 08 09).
// The first byte (0x00) is the destination address
// 0x00 means that the packet will be broadcasted
for(i=0 ; i<packet.length ; i++)
  packet.data[i] = i;

// Transmit packet
panstamp.radio.sendData(packet);
```

---

receiveData

uint8_t receiveData([CCPACKET *packet)

Description

Read wireless packet (CCPACKET) from Rx FIFO. This method has to be called every time the MCU is interrupted by a packet reception. The stack already uses this function internally so, instead of calling it from your application, attach a custom function following the Example below.

Parameters

packet : Pointer to the packet received

Example

```
/**
 * This function is called whenever a wireless packet is received
 */
void rfPacketReceived(CCPACKET *packet)
{
  if (packet->length > 1)
  {
    // packet.data[0];   // Our source address
    // packet.data[1];   // First data byte
    // packet.data[2];   // Second data byte
```

```
   // ...
   // packet.data[60];  // Maximum data byte per packet
  }
}

/**
 * Arduino's setup function
 */
void setup()
{
  // Init RF IC
  panstamp.init(CFREQ_868);
  panstamp.radio.setChannel(RFCHANNEL);
  panstamp.radio.setSyncWord(SYNCWORD1, SYNCWORD0);
  panstamp.radio.setDevAddress(SOURCE_ADDR);
  panstamp.radio.setCCregs();

  // Let's disable address check for the current project so that our device
  // will receive packets even not addressed to it.
  panstamp.radio.disableAddressCheck();

  // Declare RF callback function
  panstamp.setPacketRxCallback(rfPacketReceived);
}
```