

Iri Database

Yacob Ben Youb



Hardware:

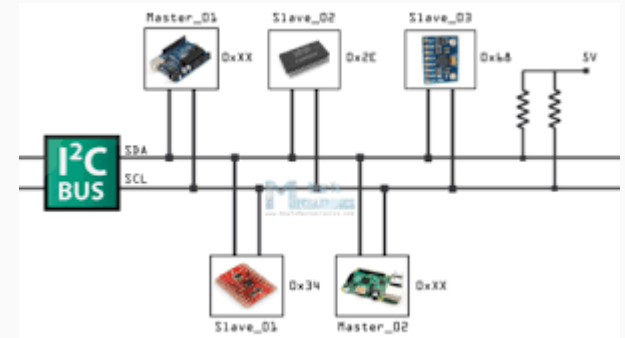
Controllers: Industruino

voor optimale beschikbaarheid zou een PLC systeem gebruikt moeten worden. Echter zullen een stel goed beschermde arduino's de taak ook zeer efficiënt volbrengen, en hoewel ze in men's oog "minder betrouwbaar" lijken, hebben deze controllers een zeer hoge beschikbaarheid. Tevens is het relatief simpel om een defecte sensor te vinden en te vervangen indien er iets fout gaat.



Protocol: I2C

Voor dit project werken we met sensoren welke [het I2C protocol](#) gebruiken. Dit is een seriele verbinding welke een groot aantal slaves op een bus accepteert, tot op grote afstand. Dit protocol wordt gebruikt omdat onze sensoren hier goede compatibiliteit mee hebben.



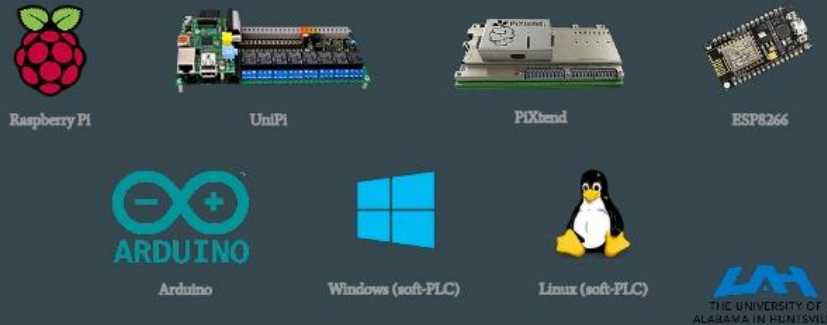
Operating System:

Voor het operating system op de server en clients moet een real time operating system worden geïnstalleerd om de sensoren in real time uit te lezen. Voor dit project gebruiken **Open PLC**. Dit is relatief nieuwe open-source operating system, welke meer realtime is dan de meeste PLC's, en zeer reliable is.

Eventueel is het mogelijk om een andere Linux distributie te gebruiken indien deze real-time compatible zoals bijvoorbeeld Ubuntu met het **Linux-Realtime package** geïnstalleerd.

OpenPLC - An Open Source Industrial Controller

Supported platforms



The image displays a collection of logos for various hardware and software platforms supported by OpenPLC. The logos are arranged in two rows. The top row includes the Raspberry Pi logo, a photograph of a UniPi board, a photograph of a PiXtend board, and the ESP8266 logo. The bottom row includes the Arduino logo, the Windows logo, the Linux logo (Tux penguin), and the logo for The University of Alabama in Huntsville.

Raspberry Pi

UniPi

PiXtend

ESP8266

Arduino

Windows (soft-PLC)

Linux (soft-PLC)

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Clients (Java):

Omdat de client in Java geschreven is, is het niet van groot belang welk operating systeem er gebruikt wordt. Java draait namelijk in zijn eigen virtuele omgeving en werkt op bijna elke OS precies hetzelfde.

In dit geval zullen we een normale PC met Linux gebruiken omdat het stabiel is dan Windows, en geen last heeft van onderbrekingen zoals Windows update.



Analog sensors (64 pieces 12 bit 100ms):

Texas Instruments TMP 101

Interface: I2C, SMBus, 2-Wire

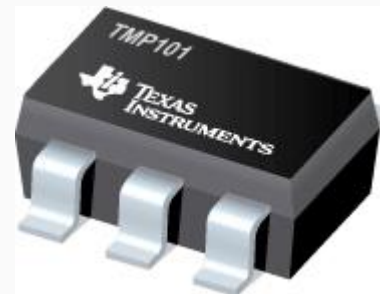
Local Sensor Accuracy (Max) (+/- C) : 2

Temp Resolution (Max) (bits) : 12

Operating Temperature Range (C) :-55 to 125

Max of 8 sensors per I2C bus in serie

Reden voor keus: Voldoet precies aan specs, nieuwere versie van TMP100 dus beter support en stability



Analog sensors (16 pieces 12 bit 10ms):

Texas Instruments AMC7812b

Interface: SPI. I2C

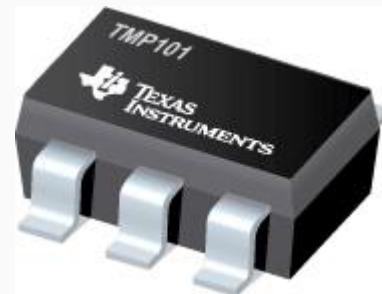
Local Sensor Accuracy (Max) (+/- C) : 2

Temp Resolution (Max) (bits) : 12

Operating Temperature Range (C) : -55 to 125

Reden voor keus: temperature range dan AMC7812, voldoet precies aan specs.

<http://www.ti.com/product/AMC7812>



Binary sensors (384 pieces):

Texas Instruments 74CBTLV1G125DBVRQ1

Number of Bits: 1 bit

On Resistance - Max: 25 Ohms

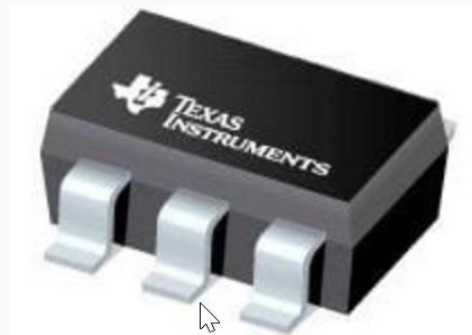
Propagation Delay - Max: 250 ps

Minimum Operating Temperature:

- 40 C

Maximum Operating Temperature:

+ 125 C



Reden voor keus: Temperature range past goed bij de specs. Werkt met hoge weerstand op afstand.

<https://eu.mouser.com/ProductDetail/Texas-Instruments/74CBTLV1G125DBVRQ1?qs=sGAEpiMZZMvjbjwkTuU2advoth7oducQXHuyfROn25Y%3d>

Software:

Iri Database:

The Iri Database is written in C. It utilizes Structs to keep track of sensors in an object oriented way.

After receiving the sensor values they are converted to a JSON format in an object oriented way and then sent to the server using Mongoose. The Client will then be able to read the JSON from there.

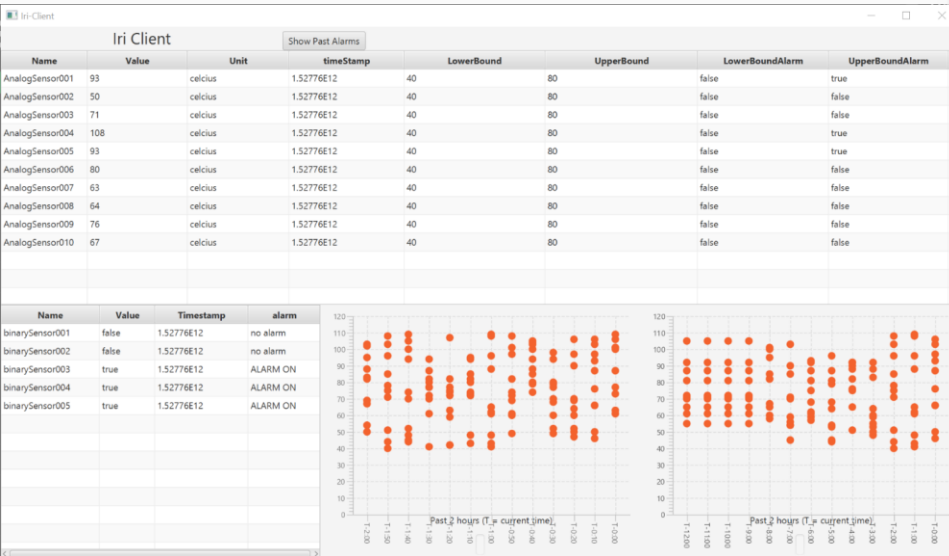
To change the boundaries on the client, a special arraylist is made to keep track of changed sensor values, which is applied at the start of every method.

The screenshot displays a Visual Studio IDE with several open files. The primary focus is on a C source file named `main.c`, which defines two structs: `binarySensor` and `analogSensor`. The `binarySensor` struct includes fields for `name` (40 bytes), `value` (boolean), `epochTime` (double), and `alarm` (40 bytes). The `analogSensor` struct includes `name` (40 bytes), `value` (int), `unit` (15 bytes), `epochTime` (double), `lowerBound` (int), `higherBound` (int), `lowerBoundAlarm` (boolean), and `higherBoundAlarm` (boolean). A static array `changedSensorList` of size 1000 is also declared.

Overlaid on the code is a window titled `analogSensor` showing a JSON representation of an analog sensor's data. The JSON object contains the following fields: `sensorValue` (68), `unit` ("celcius"), `timestamp` (1.527759e+12), `lowerBound` (40), `upperBound` (80), `lowerBoundAlarm` (false), and `upperBoundAlarm` (false). Below this, two more JSON objects are shown for `AnalogSensor009` and `AnalogSensor018`, each with their respective values and bounds.

The IDE's interface includes a Solution Explorer on the left, a Status Bar at the bottom showing 0 errors and 0 warnings, and a Diagnostic Tools window on the right displaying a diagnostic session summary.

Java Client:

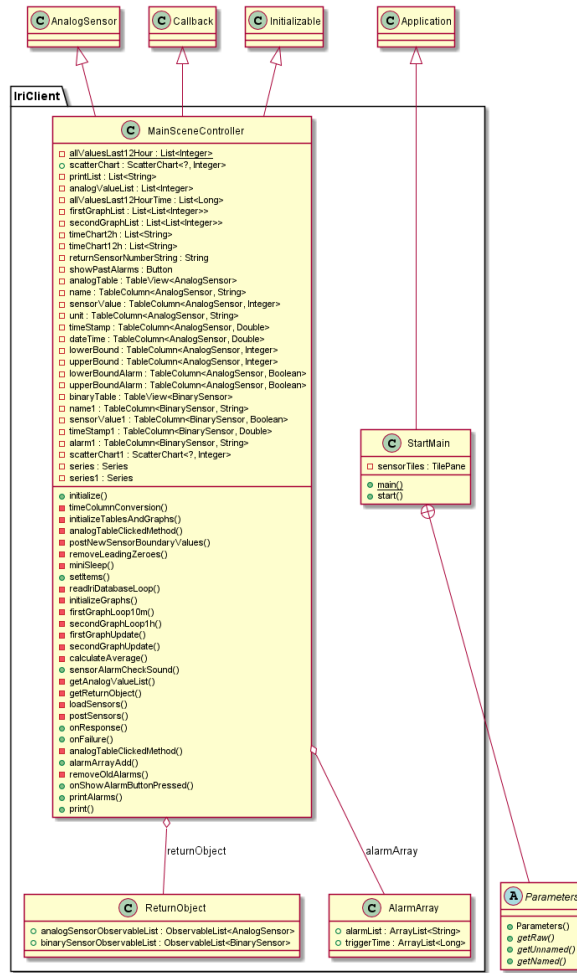


The Java client reads the values from the C database using Retrofit, which are then visualized using JavaFX. There are two tableviews, along with two graphs to see the data over time. One of these graphs holds values for each 10 minutes over a period of 2 hours, the other one for each hour over a period of 12 hours.

The database also sounds an alarm whenever a sensor exceeds its allowed maximum value. Furthermore it can show you all triggered sensors from the past 12 hours, and print those.

UML:

IriClient's Class Diagram



timezone
Struct

Fields

- tz_dsttime
- tz_minuteswest

analogSensor
Struct

Fields

- epochTime
- higherBound
- higherBoundAl...
- lowerBound
- lowerBoundAla...
- name
- unit
- value

binarySensor
Struct

Fields

- alarm
- epochTime
- name
- value

Einde

Visual Studio Code interface showing the development of the Iri-server. The main editor displays the `analogSensor.c` file, which defines the `analogSensor` structure and its initialization. The structure includes fields for `name`, `value`, `unit`, `epochTime`, `alarm`, `lowerBound`, `upperBound`, `lowerBoundAlarm`, and `upperBoundAlarm`. The `main` function initializes a list of sensors and starts a loop to read sensor data.

```
1 #include <stdio.h>
2 #include <stdbool.h>
3
4 struct binarySensor
5 {
6     char name[40];
7     bool value;
8     double epochTime;
9     char alarm[40];
10 };
11
12 struct analogSensor
13 {
14     char name[40];
15     int value;
16     char unit[15];
17     double epochTime;
18     int lowerBound;
19     int higherBound;
20     bool lowerBoundAlarm;
21     bool higherBoundAlarm;
22 };
23
24 static int changedSensorList[1000];
```

The `analogSensor` structure is defined with the following fields:

- `name`: char[40]
- `value`: int
- `unit`: char[15]
- `epochTime`: double
- `alarm`: char[40]
- `lowerBound`: int
- `higherBound`: int
- `lowerBoundAlarm`: bool
- `higherBoundAlarm`: bool

The `main` function initializes a list of sensors and starts a loop to read sensor data.

```
1 int main()
2 {
3     // Initialize sensors
4     // ...
5     // Start loop to read sensor data
6     // ...
7 }
```

The `analogSensor` structure is defined with the following fields:

- `name`: char[40]
- `value`: int
- `unit`: char[15]
- `epochTime`: double
- `alarm`: char[40]
- `lowerBound`: int
- `higherBound`: int
- `lowerBoundAlarm`: bool
- `higherBoundAlarm`: bool

The `main` function initializes a list of sensors and starts a loop to read sensor data.

```
1 int main()
2 {
3     // Initialize sensors
4     // ...
5     // Start loop to read sensor data
6     // ...
7 }
```

