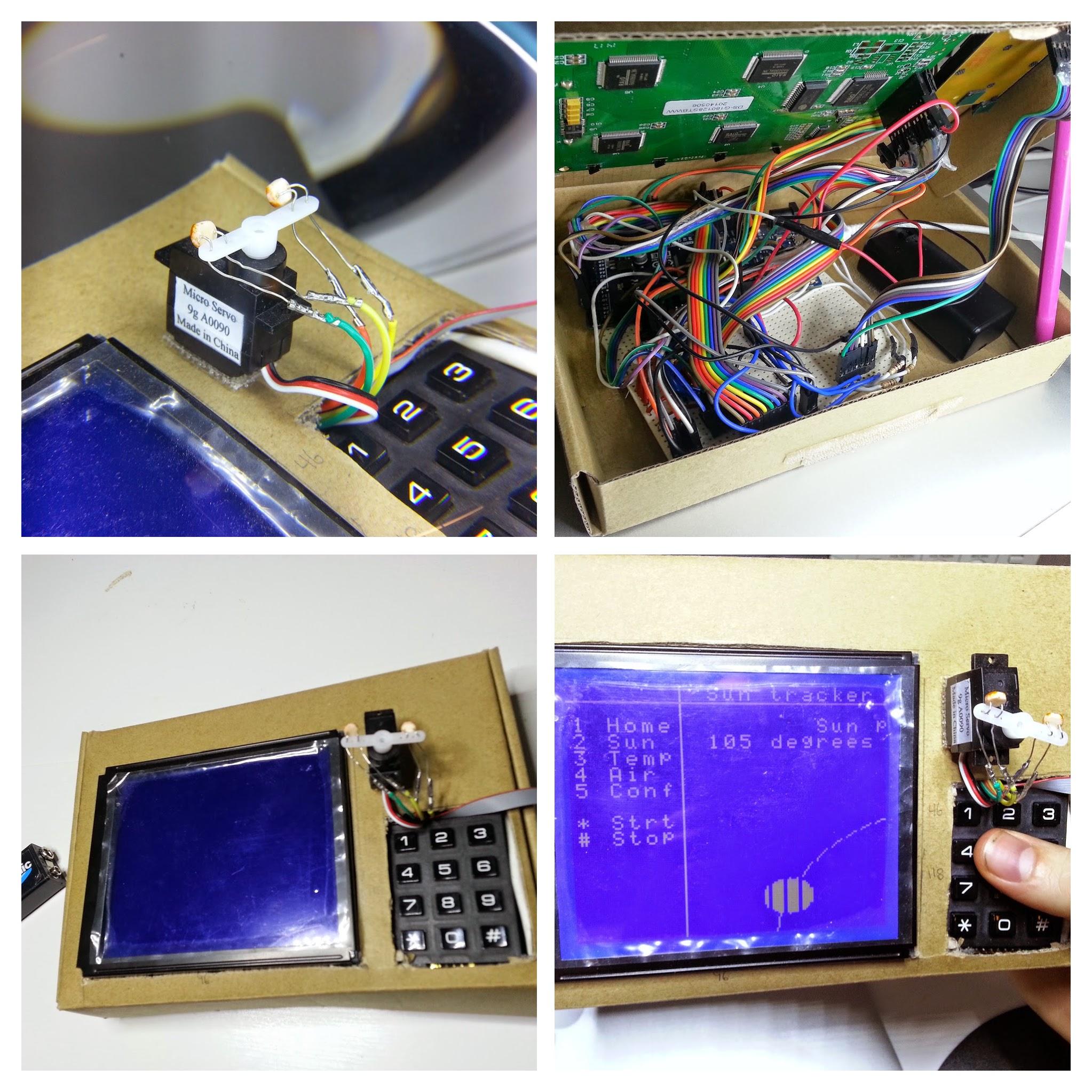
Technical documentation for Open Source

Weatherstation

by Daniel Dahlberg and Staffan Piledahl

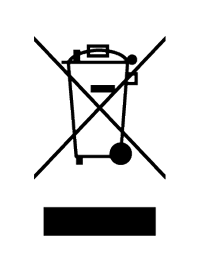


Table of contents

1. Description
2. User guide
   1. First startup
      1. Testing screen
      2. Setting the time and date
   2. The home screen and header
   3. The light follower screen
   4. The logged temperature- and air pressure-data screens
   5. The settings screen
3. Module list
   1. Temperature sensor
   2. Air pressure Sensor
   3. Memory and logging
   4. User interface
   5. Light follower
4. Tools and methodology
5. Test specification
   1. Test results
6. Description

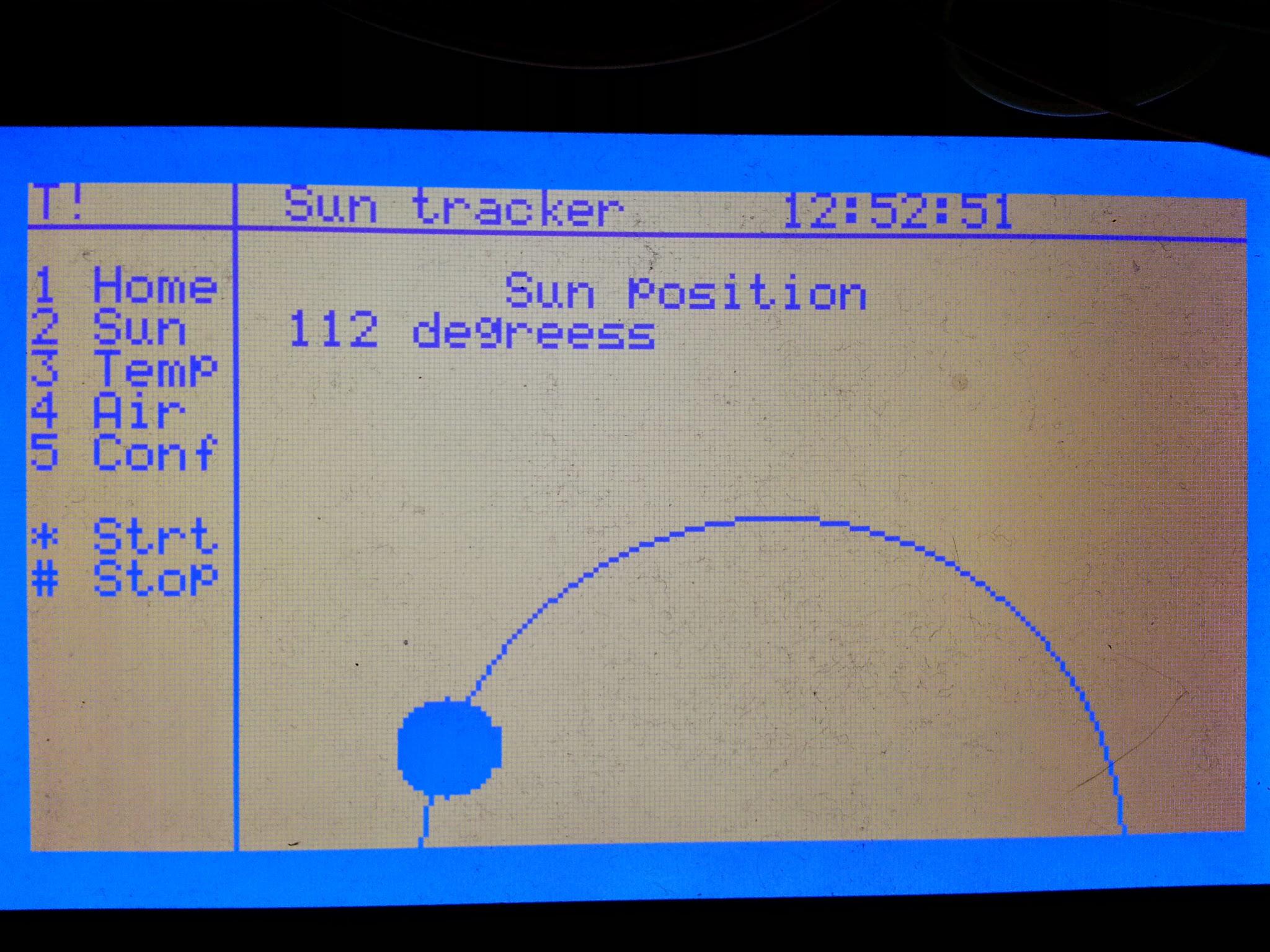
The OS Weatherstation features four different sensors one detachable servo motor and a big beautiful display that together with a familiar keypad creates a straightforward user interface.

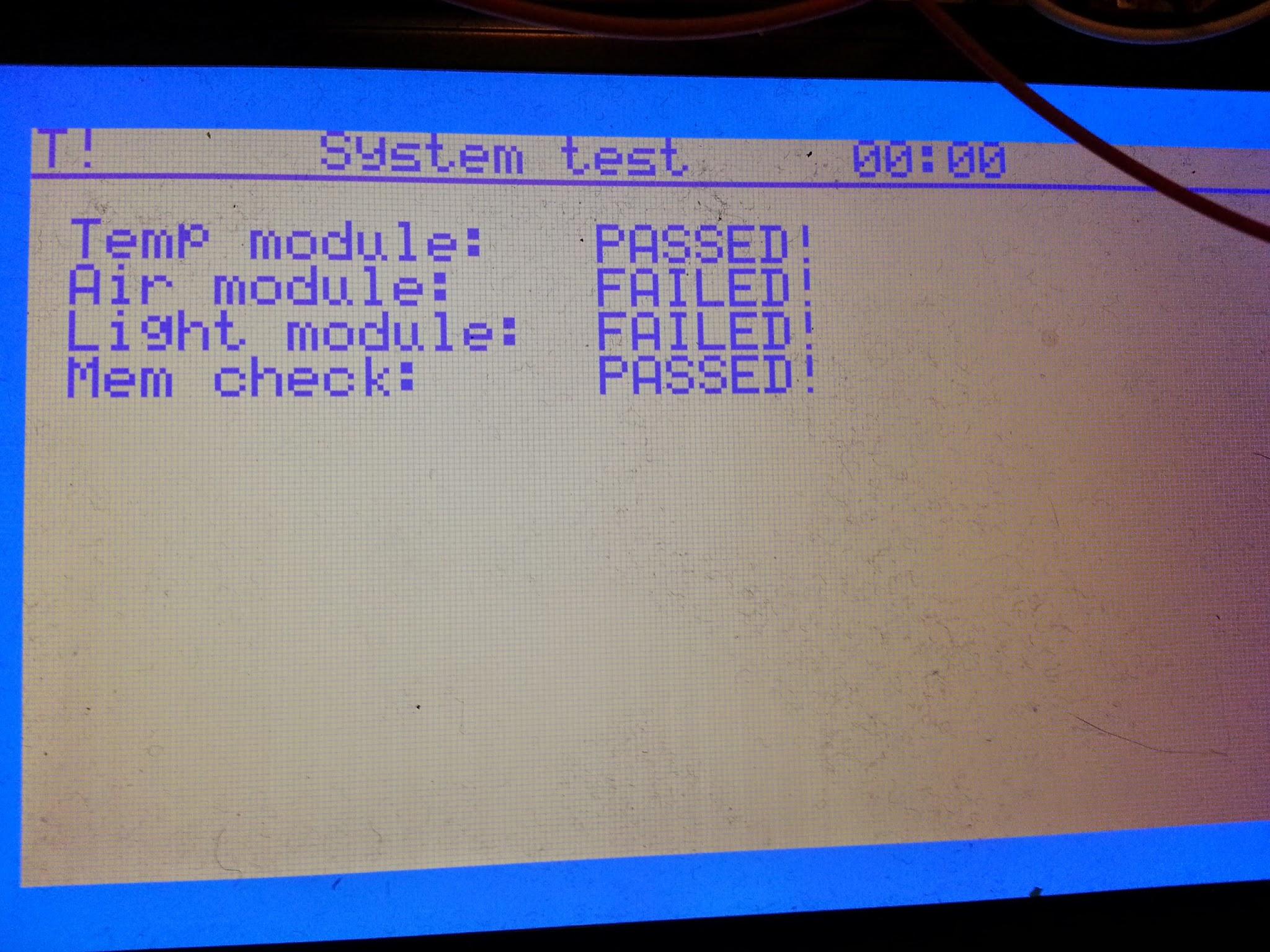
Being built on an arduino due platform together with tried-and-true peripheral hardware the weather station is robust and more information is readily accessible on the internet. Our goal was to create a platform that was ready to use from the box but easily extendable with more sensors or more custom code.

Applications: Keep track of weather changes in recent time, find the angle of a light source (like the sun) or best of all: add sensors or re-program the station into anything you want! Thanks to a well documented, easily decoupled code base with well known coding conventions the educational possibilities of the OS Weatherstation is endless.

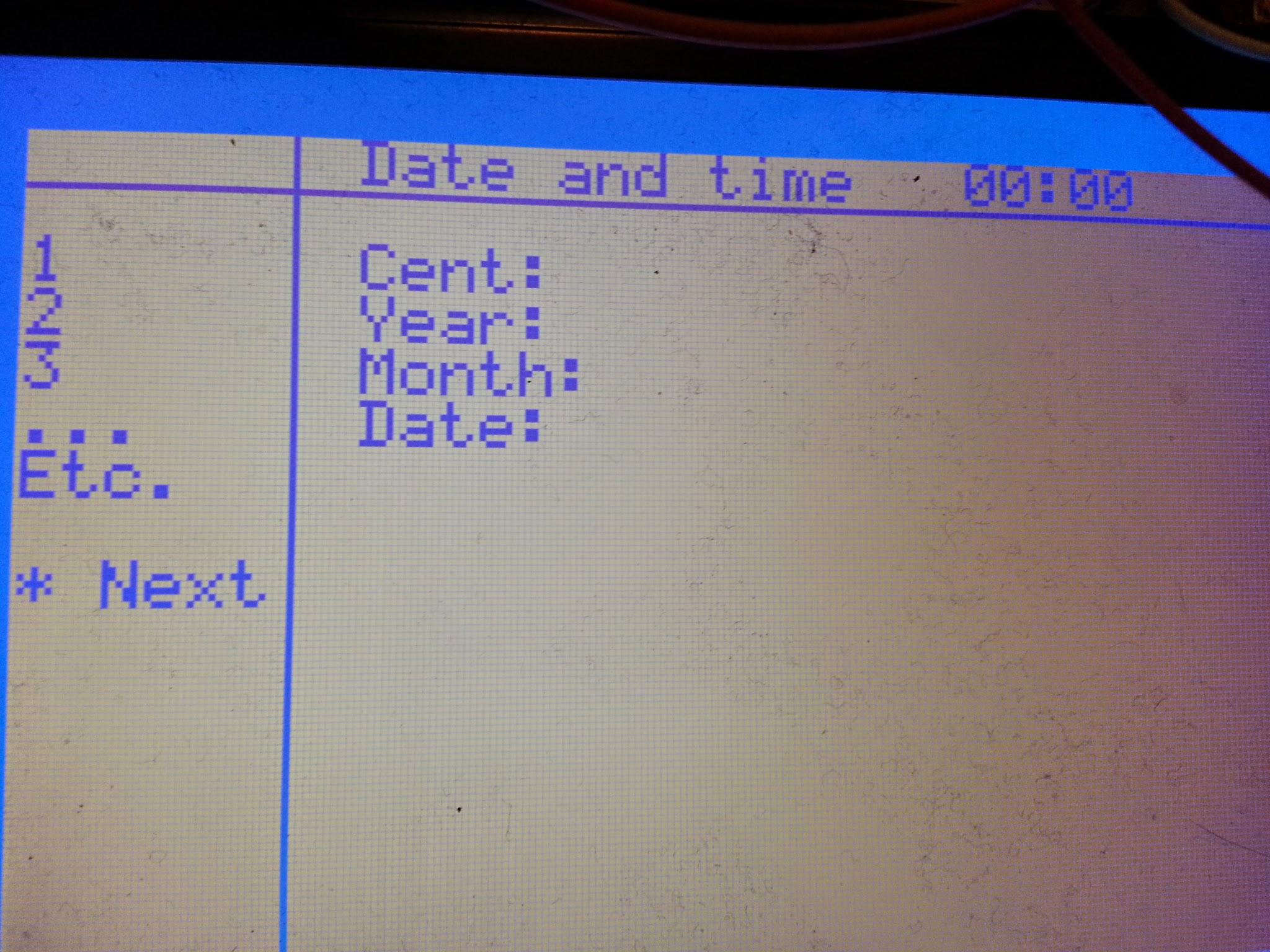
Take a look at <https://github.com/danned/projekt> on how to get started or help make the product better!

* Daniel and Staffan



2. User guide

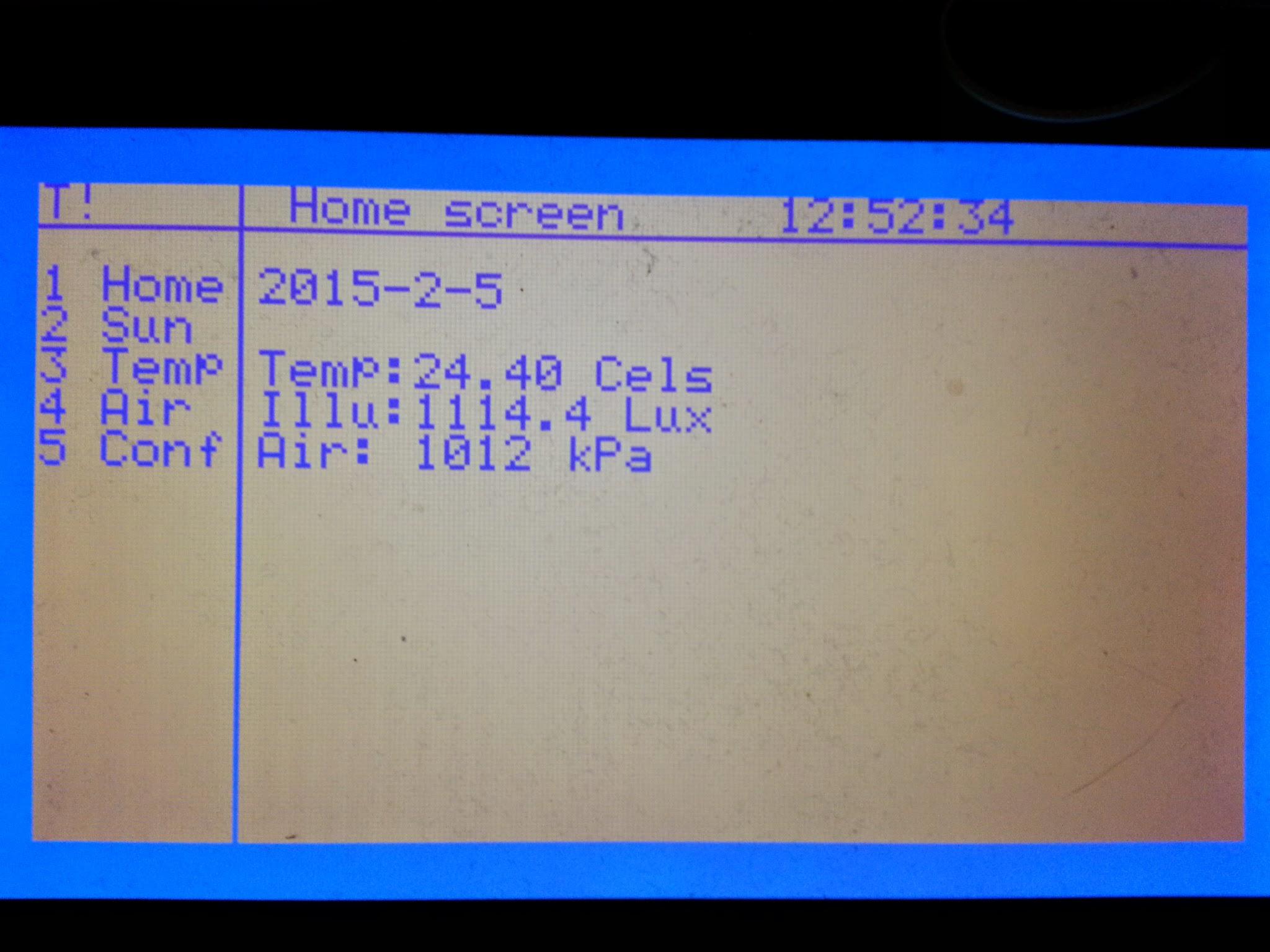
The first screen on startup that shows up is the component tests result. If a component does not meet its spec or is powerless, the test will fail. Press any key to continue



The second screen on startup is the date and time entry screen. Enter the values using the keypad. The first value is the century, the only acceptable values are 19 or 20.

The second value is the year, 00-99. (Use a leading 0, eg. 05 for “5”).

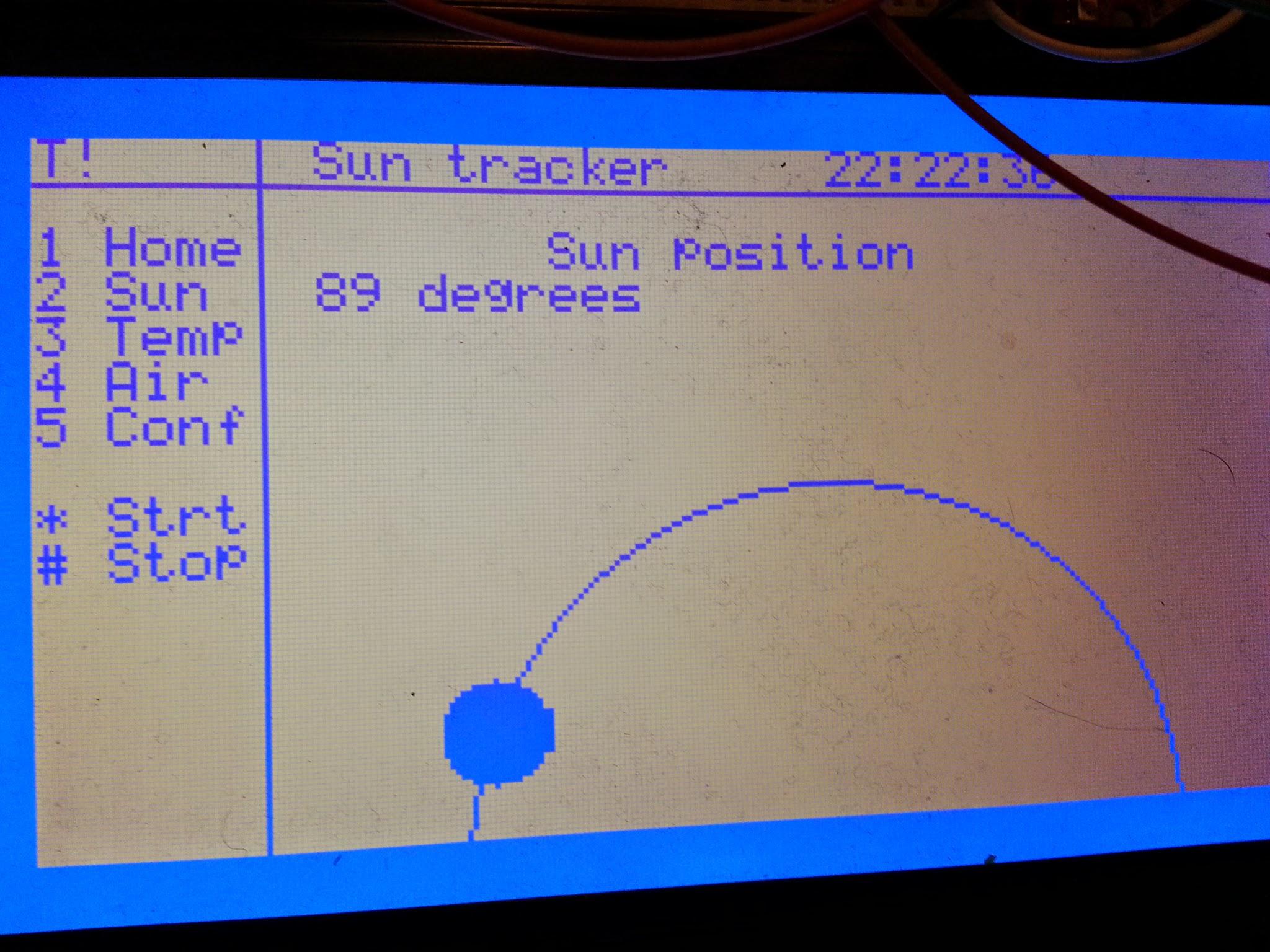
Next is month and date in the same manner. Any values that does not make sense will make the date invalid and the station must be restarted to enter a new date. After date has been entered a similar screen with time shows up.

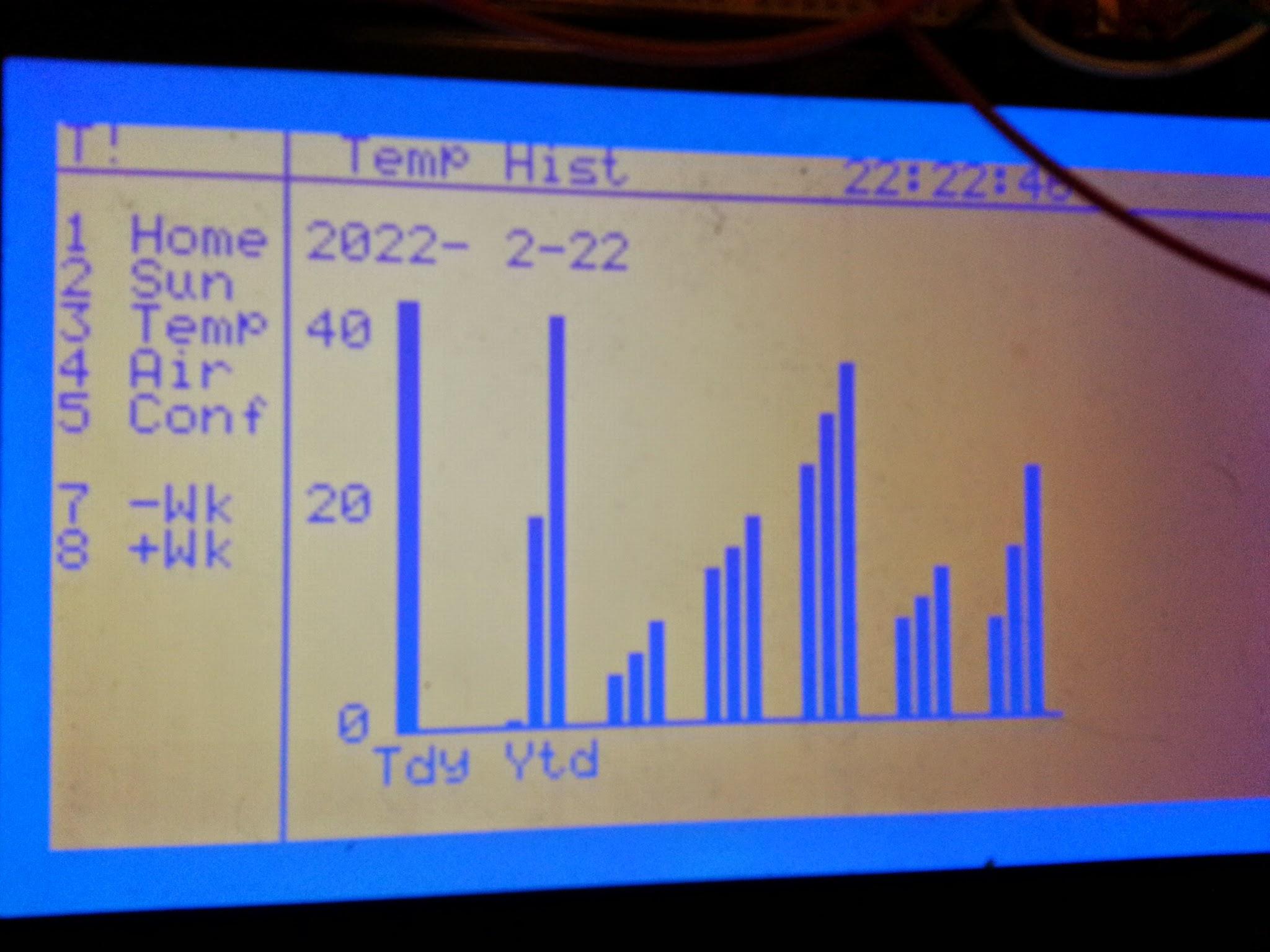


This is the screen where the current values for temperature and air pressure are shown, as well as todays date.

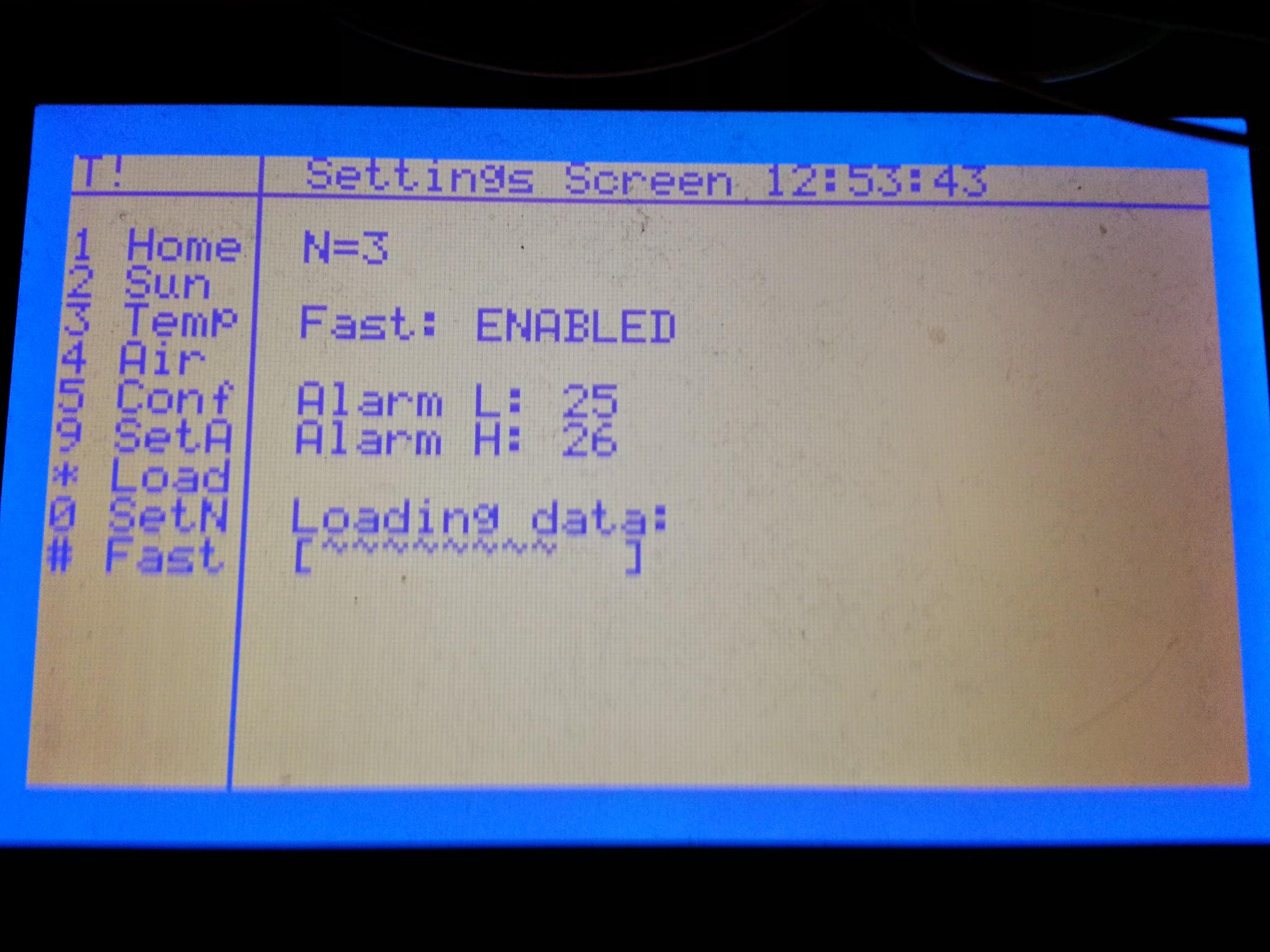
(A rough value for illumination is also displayed but should be taken *lightly*).

The header bar on top shows the title of the screen, the time and any warnings that may be active (T! for temperature warning and M! for memory overwriting warning). The sidebar shows the menu buttons 1-5 to change screens. This is also where contextual buttons appears.



This screen shows the angle of the motor and the start and stop buttons (\* and #). The image demonstrates how graphics is drawn on the screen.

This screen shows seven bar graphs at a time representing the most recent seven days with today and yesterday being indicated on the horizontal axis. The vertical axis is displaying the degrees. The first of the three individual bars is the minimum value for the day, the next is the average and the third is the maximum value that day. Browsing to previous weeks is done with the keys shown in the sidebar. The air pressure screen is similar but without browsing



In the setting screen using the buttons in the menu the averaging level N can be chosen from 1-10 (10 is star key). The temperature upper alarm limit and lower alarm limit can also be set in the same manner. But here the zero key is used as usual. Toggling of fast mode is done using the # key and loading preset data is done using the star key.

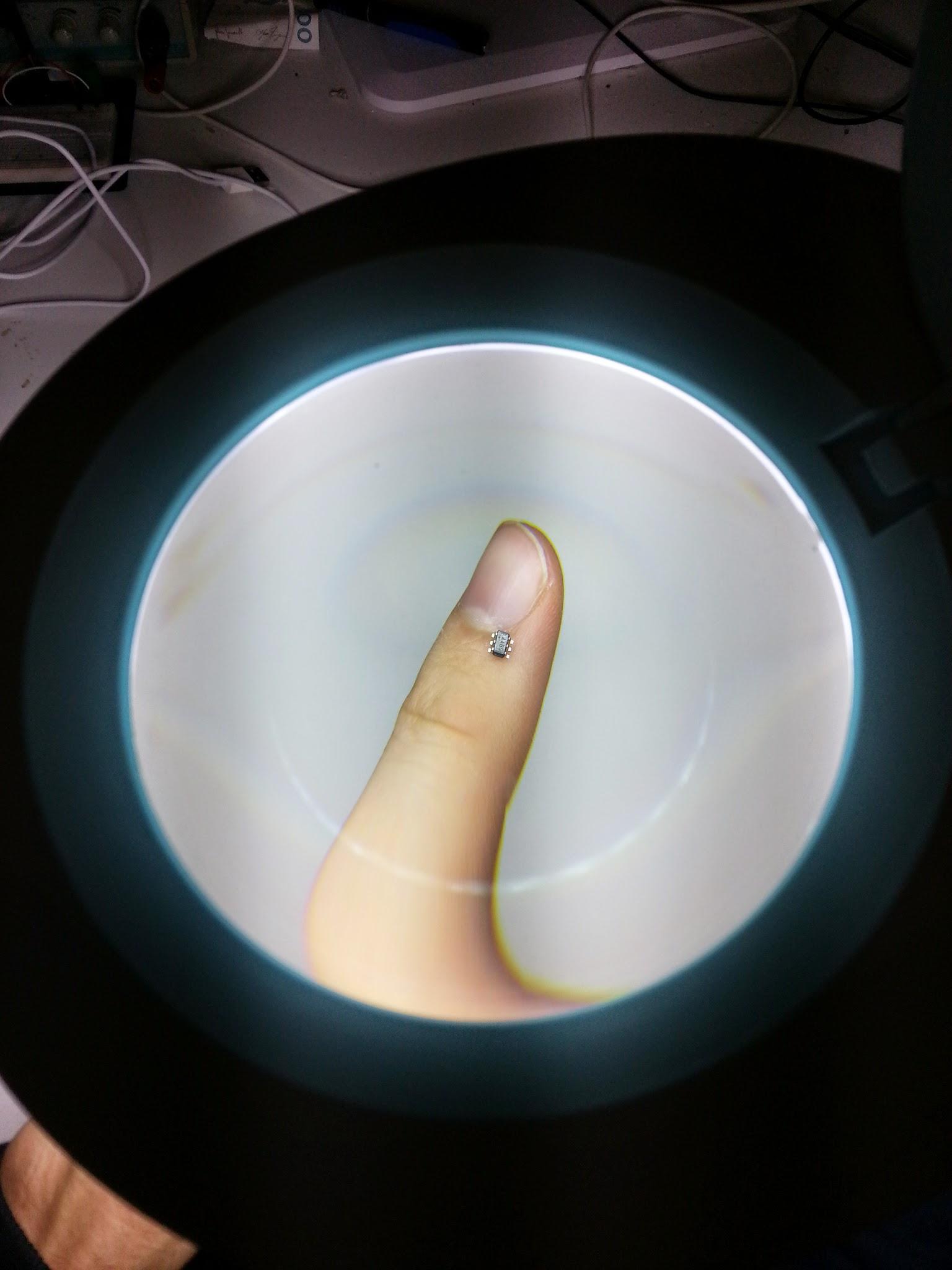
3. Module list

**Temperature sensor**

The temperature sensor is a digital IC called MAX6575L. Using asynchronous one-wire interface both reading and writing can be used over one IO pin. Note that this sensor differs radically from a typical NTC thermistor. This sensor requires a sequence of asynchronous pulses to get the temperature.

The chip requires a 15ms pulse before a reading can be done. (More info in sensor datasheet page 4 <http://datasheets.maximintegrated.com/en/ds/2024.pdf>). This pulse is generated using timer counter block 1 in waveform.

The length in time of the pulse response from the chip determines the temperature. This response is captured using timer/counter block 0 set to capture mode. More information on waveform and capture modes in MCU documentation:

<http://www.atmel.com/Images/doc11057.pdf> (TC section)

The code uses the flags: temperature.status.RESET\_READY and .READ\_READY to detect what stage of the following process is happening:

Sequence started using reset (required before every reading), when the flag comes up read() is called and finally when read flag comes up get() can be called to fetch a float with the degrees.

Functions to use:

void TEMP\_init(),

void TEMP\_reset(),

void TEMP\_read(),

float TEMP\_get()

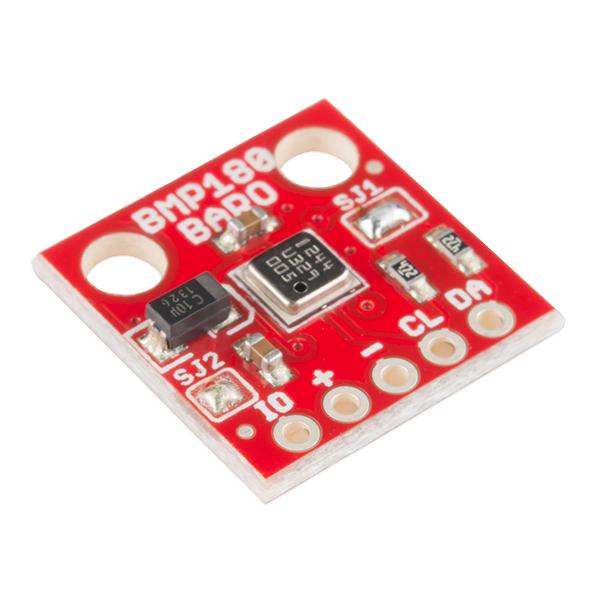
Blocks used: TC0,TC1, PIOB

Note: Works safely down to one reading every 20ms. can be read once every 17ms with testing setup but not recommended.

Usage examples in temp\_sensor.h <https://github.com/danned/projekt/blob/master/peripherals/temp_sensor.h>

**Air pressure sensor**

The air pressure sensor is a BOSCH BMP180 on a sparkfun breakout board. It is a digital sensor communicating over two-wire interface (I2C). The implementation made directly support air pressure reading but the hardware is configurable for other ways of operation. Full I2C RW exists which makes extending this module easy. See BMP 180 device manual for other functionality: <https://ae-bst.resource.bosch.com/media/products/dokumente/bmp180/BST-BMP180-DS000-09.pdf>

The two wires mentioned is one data wire and one clock wire. To communicate over this I2C bus the “address” of the device in question is needed as well as any internal register addresses on the device (found in datasheet). The weather station setup features only one I2C device so the MCU is hardcoded as master and the BMP180 as slave (control is never released). 

The TWI block in the MCU is tailored to this protocol. Using for example a certain bit in a TWI block register, the direction of transfer can be set; receive or transmit. And two registers holds data coming from the slave or going to the slave. These are called holding registers.

The pressure value is delivered in three bytes placed in three different registers, hence three reads are required.

Functions to use:

void AIRSENS\_init( );

int AIRSENS\_getPres();

Blocks used: PIOA, TWI0

Note: Library includes I2C drivers tailored for this device.

Usage examples in air\_sensor.h <https://github.com/danned/projekt/blob/master/peripherals/air_sensor.h>

**Memory and logging**

This part consists of a time module (Real time clock, RTC) and a memory module. The real time clock is a tailored block in the MCU to keep track of date and time with seconds being the atomic unit. The module also generates interrupts on seconds, minutes or days depending on which mode the station is in. These interrupts is the entry point for logging data. More info on the RTC can be found in Atmels datasheet for the MCU (mentioned above) around page 256.

Every minute is recorded using averaging with a user configurable amount of readings throughout that minute (1-10), this gives a smoother value. The RTC could have managed this in normal mode but in fast mode it requires more frequent interrupts. More on these interrupts can be found in the Atmel datasheet under “Systick”.

The memory module is based around a singly linked list for the temperature logging and an array for the air pressure logging. Both readings store the minimum, maximum and average value for a day as well as what level of averaging has been done. The module will raise a flag when the memory is full (MEM\_FULL) and overwriting of old entries has begun. A flag for memory errors also exists (MEM\_ERROR). The flags and data structures can be reached from the mem structure.

To simplify testing and demonstration a random set of temperatures and air pressure values can be loaded into the station from the settings screen. If the station is in its testing mode (fast mode) the function will fill the memory. If the station is in normal mode, one week of data will be loaded.

Functions to use:

void MEM\_init();

int MEM\_save(float new\_temp\_f, unsigned new\_pres\_u32);

int MEM\_remove();

void MEM\_fill();

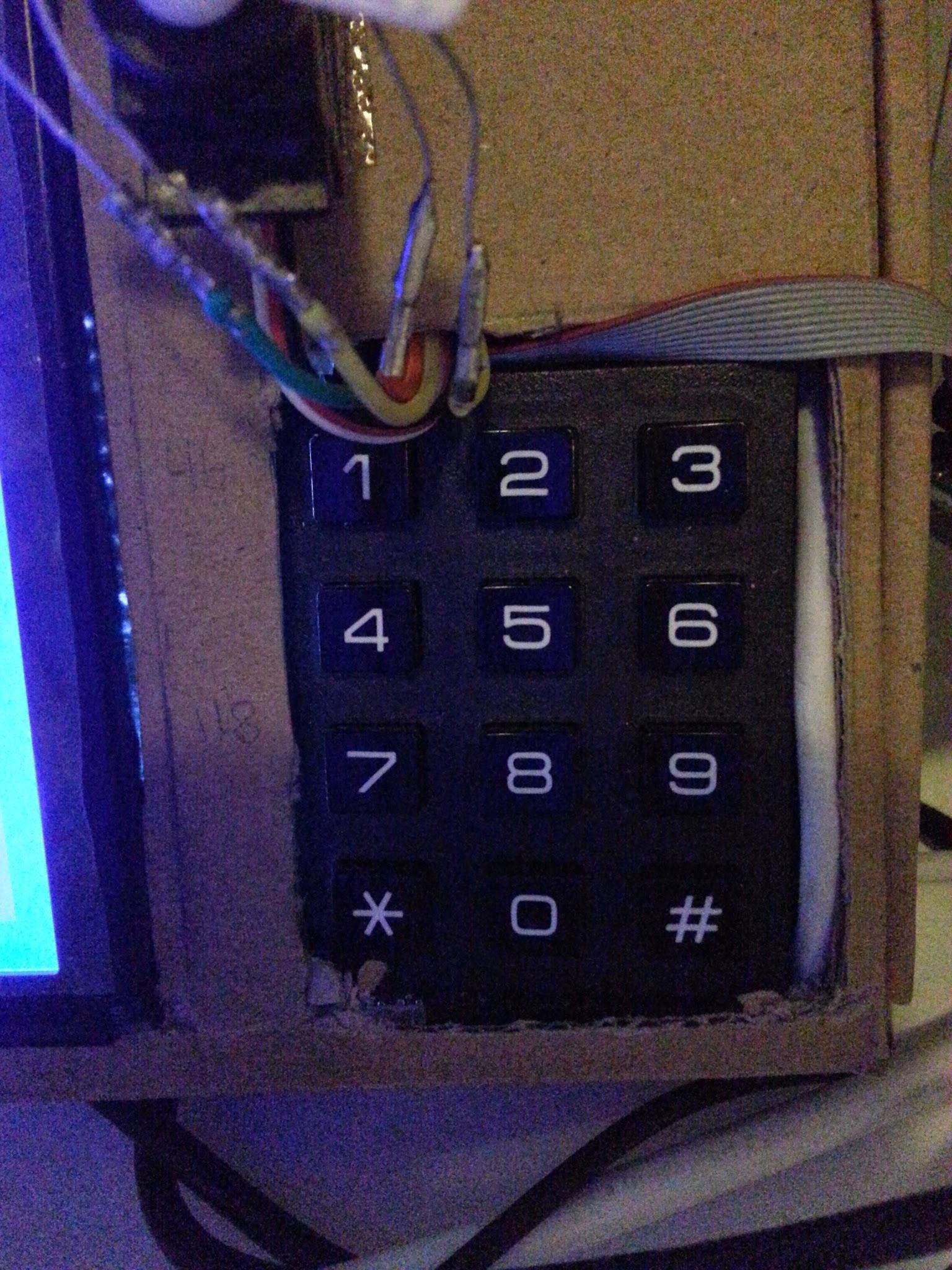
Blocks used: RTC, Systick

Note: No heap space will be available after the memory has been filled up.

Usage examples in mem\_test.c - <https://github.com/danned/projekt/blob/master/mem_test.c>

**User interface**

Note: The UI consists of two hardware modules with drivers and a software controller module to handle the user requests. The hardware is the display and the keypad which are each connected to the MCU via octal transceiver buses to allow for both multiplexing and direction control. This is handled internally and should only be of concern when using keypad or display in advanced ways or when changing hardware. More information about the buses used: <http://www.nxp.com/documents/data_sheet/74LVC_LVCH245A.pdf>

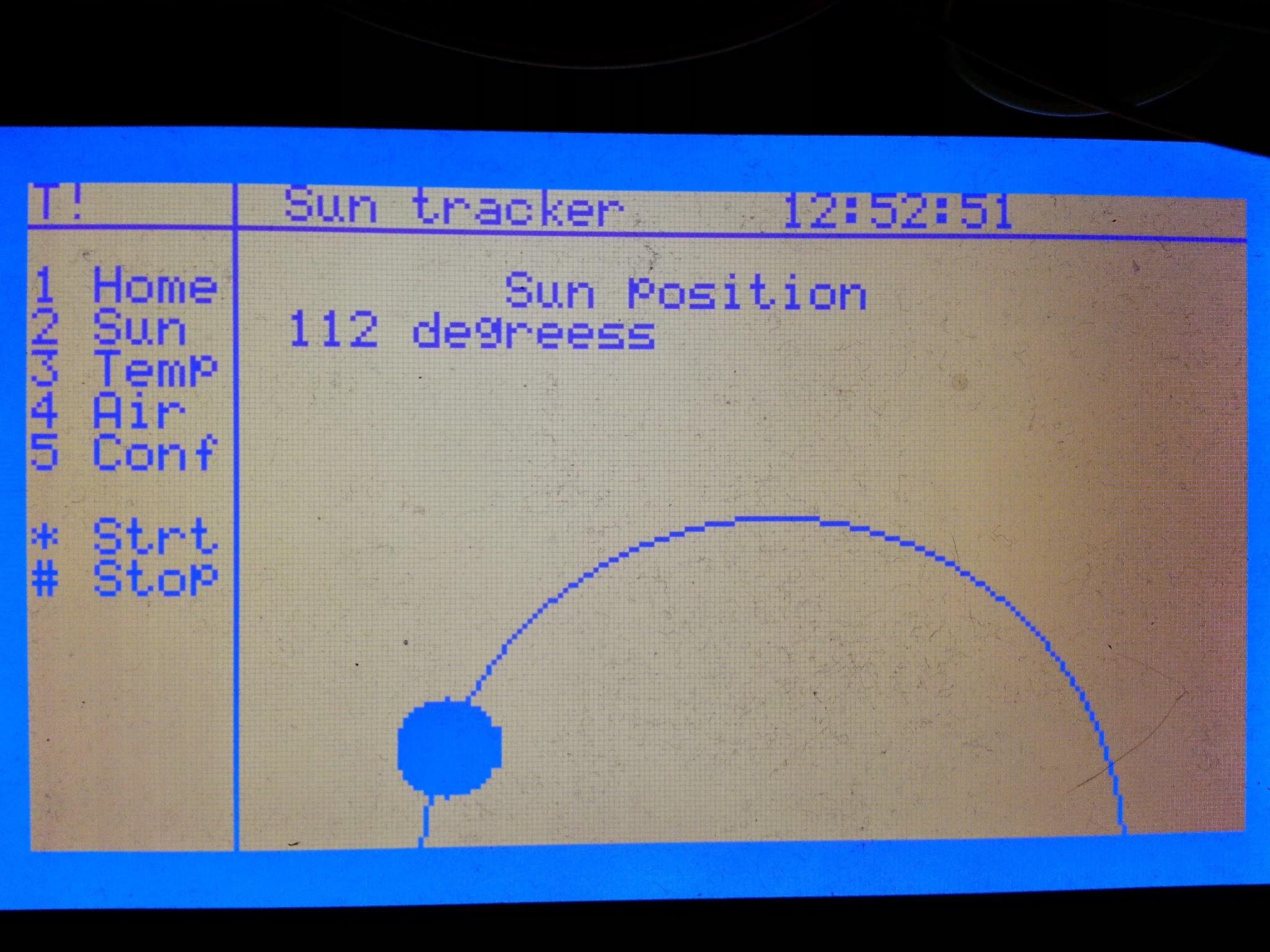


The keypad uses a matrix of four rows (inputs to MCU) and three columns (high outputs from MCU). By polling the rows while setting one of the columns to zero a button press on that column will result in the the row going low. Using Keypad\_Read() will handle this under the hood.

The display is a RA6963 Dot Matrix LCD Controller. It uses a set of control lines and a set of data lines. The data lines are connected via the transceiver mentioned above. (The first two data lines are used to get status messages from the display hence not only multiplexing but direction control is useful).

The display works by reading one or two data bytes followed by a command byte, for example a byte that means the letter a and a byte that means write and increment cursor. This underlying way of operation is abstracted away in the write, draw and clear methods mentioned below. But note that the screen implementation in the device originally does not make use of the screens vast internal memory for saving different screens, but rewrites to one place in memory every time. If screen-intense applications are needed then a change in the code will be necessary to load in screens asynchronously and then just changing the internal memory pointer of the display chip instead of rewriting the screen from MCU in real time.

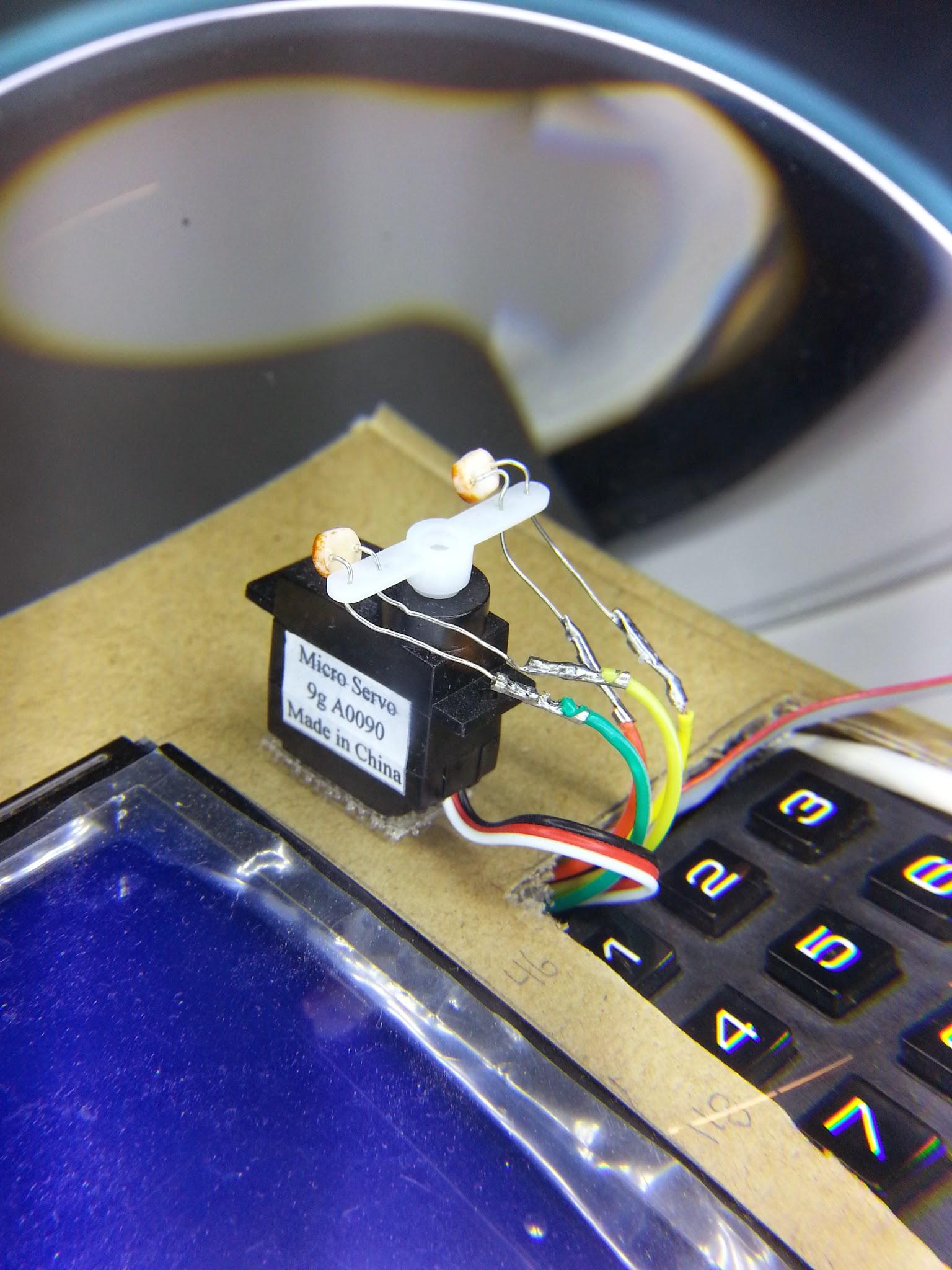
To learn more consult the datasheet: <http://www.mitsutech.com/RA6963_DS_v13_Eng.pdf>



For general purpose use of the display the built in functions should suffice.

See the usage examples in display.h - <https://github.com/danned/projekt/blob/master/peripherals/display.h>

The controller provides application specific switch case and state structure as well as holding the memory full and temperature warning flags. It is the center of the UI web reading the keypad and writing the screen. It is divided into loading screens on buttons 1-5 and handling contextual button presses within a screen using higher numbers, as a general rule. The exception being when numbers need to be entered, then obviously the entire keypad is made available. By using a module to handle most of the interaction the decoupling of drivers and application specific code makes it easier to extend or reuse the hardware.



Functions to use:

char CTRL\_userInput(char pressed);

void KEYPAD\_init(void);

int KEYPAD\_read(void);

void DISPLAY\_init(void);

void DISPLAY\_write(char \*text, char x, char y);

void DISPLAY\_clearGraphics();

void DISPLAY\_clearText(void);

void DISPLAY\_drawPixel(int x, int y);

Blocks used: PIOC, PIOD

**Light follower mode**

This component consists of two hardware modules with drivers and one light follower module in software. Entering into light follower mode via the mentioned controller enables two photoresistors mounted on a servomotor. Using the difference in their read value the servo is adjusted accordingly to follow the strongest light source in real-time.

The photoresistors are connected as a voltage divider together with a fixed value resistor producing voltages between 0V and 3.3V. This analog value is the read by an analog-to-digital-converter or ADC. The MCU has built in ADCs and a block specific for this, making life simple. A reading is initiated and an interrupt is generated when the reading is ready. The code uses the following flags:

INACTIVE, READ\_REQ, READ\_DONE and READING. They are located on lightsens.state.

The servo motor is connected by one wire to the MCU over which a pulse width modulated signal is sent. This means a periodic signal (here 20ms period) gets a high period of 1.5 - 2.3 ms out of the whole period. This tells the motor to what angle it should tur, a 1.5ms high period turning the motor all the way to one side and vice versa.

Functions to use:

int LightFollow();

- This function is a catch all for starting the mode, below are some internal functions listed.

void LIGHTSENS\_init(void);

void LIGHTSENS\_startMeas(void);

float LIGHTSENS\_getDiff(void);

void SERVO\_init( void );

int SERVO\_getPos(void);

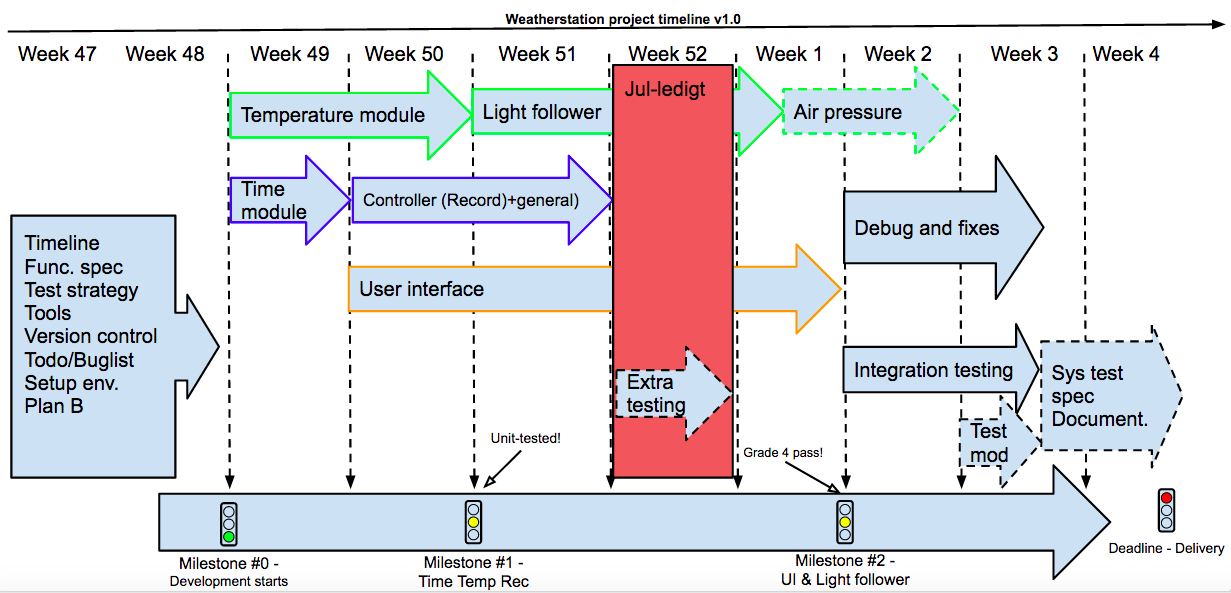
void SERVO\_setPos(int val);

Blocks used: ADC (CH1,2), PWM(CH2)

Note: The Light following mode is blocking everything but the interrupts.

4. Tools and methodology

**Project plan and overview**

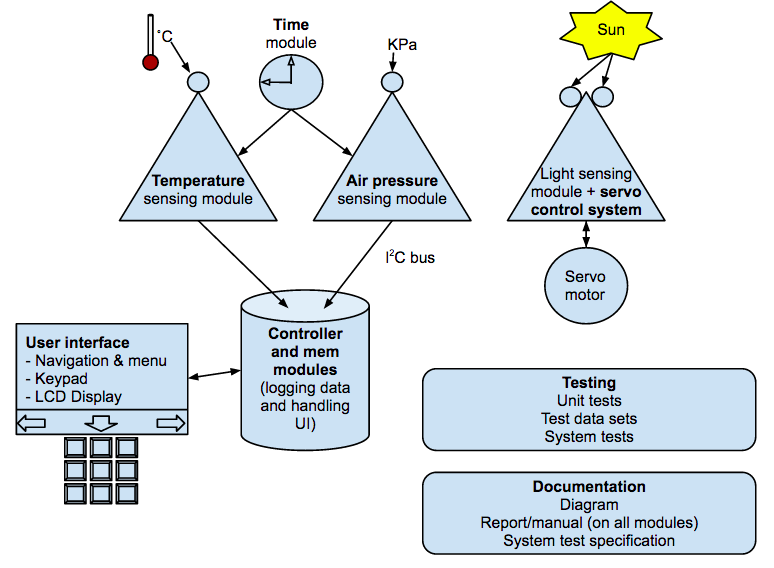


**Programming tools**

J-link EDU debugger

IAR Tool Bench for compiling and debugging.

Notepad++ and Sublime text 3 for editing

Git for version control ( <https://github.com/danned/projekt> ) with Github client, git CLI and Source Tree.

**Conventions**

* Flag usage minimizes time in interrupts.
* Structs holds flags and variables in each module. More OO like.
* Trying to follow the same naming conventions for consistency.
* Decoupling as much as possible and keeping dependencies low.
* Extensive documentation in header files and minimal documentation in c file for increased readability.
* Pair-programming the drivers and dividing the application code into front-end and back-end. Optimal for broad team understanding but still creating efficient specialists.

Test specification

Temperature range: 0 °C to +45°C

Temperature accuracy: ±1 °C

Air pressure range: 90 kpa to 120 kpa

Pressure accuracy: ±1 hpa

Operating temperature: -1°C to +50 °C

Storage temperature: -10 °C to +60 °C

Calendar: 1900–2099

Time mode: 24h

Backup battery: 9V

Sampling (both sensors): Once per minute (60 times per minute in fast mode)

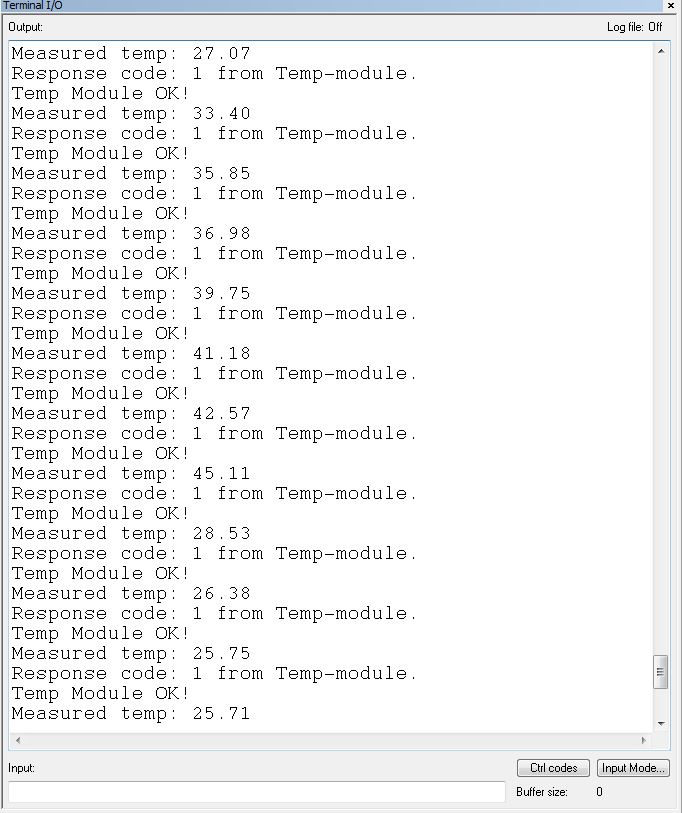
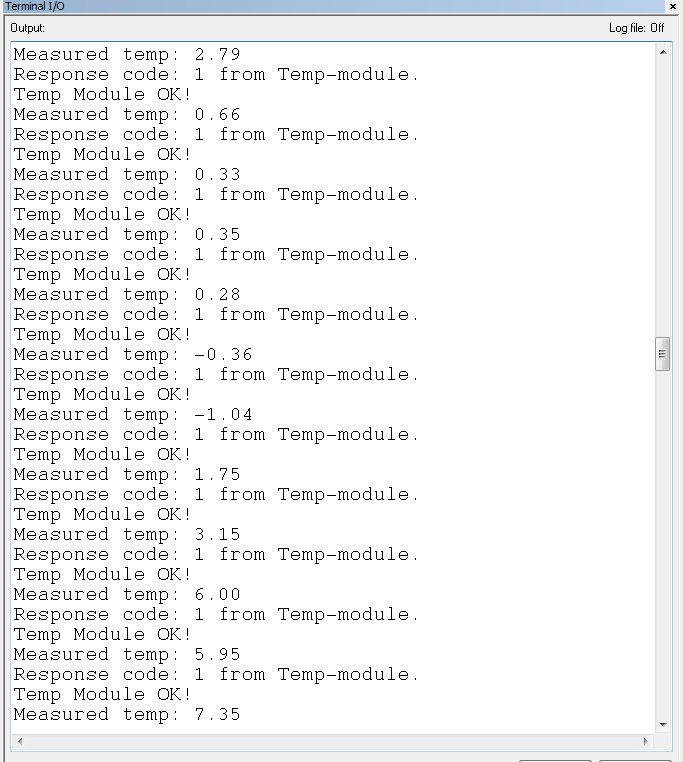
Sensor averaging: 1-10 times per minute (user configurable)

Temperature log: >300 days history (when full, overwrites oldest entry)

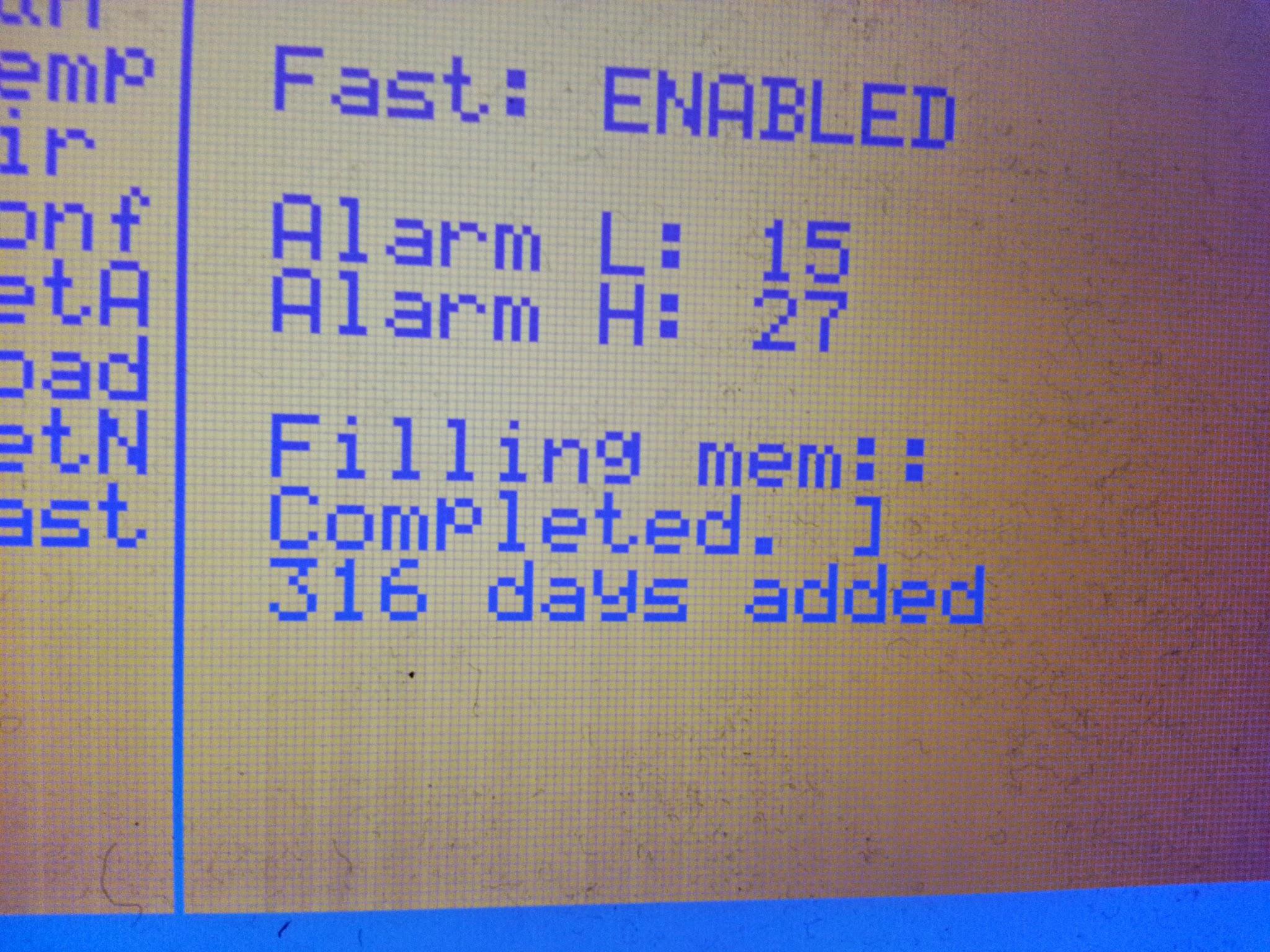
Air pressure log: 7 days history (when full, overwrites oldest entry)

Temperature sensor range and accuracy tests verified using infrared thermometer.

The temperature component test tests all parts initialization and reading. It separates timeout errors (device unplugged ~100ms) from value out of range (outside -5°C through +50°C).



Memory tests verified by inspection and continued normal operation of device.



The memory component test fills the entire memory then clears it. Reporting how many entries (equivalent to days) it had.