# Embedded Systems

## Sic Parvis Magna

**A thermometer and hygrometer project on Raspberry Pi 3 Model B**

Designed and developed by Silvio Christian Benanti

# Contents

# 1 Purpose

This project can be briefly summarized as a bare metal thermometer and hygrometer. It consists of a sensor that periodically sends temperature and humidity information, and these are displayed on the LCD monitor, there is also the possibility to interact with this instrument through a button to change the mode of use.
The project itself is the main part of the final exam of the Embedded Systems course taught by Professor Daniele Peri at the University of Palermo.
In this paper I am going to analyze in detail the hardware used and its operation, the approach I chose and several steps to pay attention to for proper operation and the software used.

# 2 Hardware

## 2.1 Raspberry Pi 3B

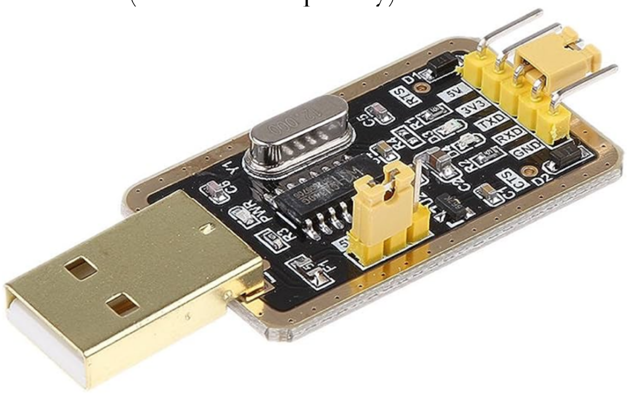
Raspberry Pi 3B is the core of the project. It is a single board computer (SBC) that is, a complete computer built on a single circuit board complete with microprocessor, memory, input/output and other features. Specifically, the features of Model 3B are as follows:

- Broadcom BCM2837 64bit 1.2GHz Quad Core CPU
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet
- GPIO extended to 40 pins
- USB ports 2
- 4-pin stereo output and composite video port
- HDMI® full-size
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- Micro SD port for loading the operating system and storing data
- Upgraded Micro USB switched power source up to 2.5A

## 2.2 I2C Uart Usb CH340 TTL

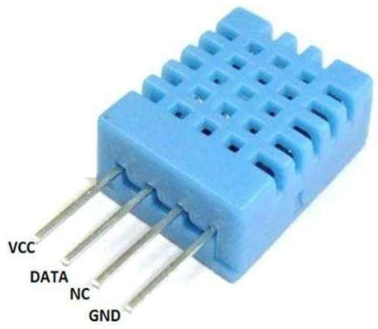
It's a small USB to TTL serial tool, using the CH340G chip. It's used to connect some serial device to PC via USB port. This is necessary because modern computers do not easily allow SBCs to use serial ports. The connection between this adapter and the Raspberry is made as follows:

- CH340 TXD PIN to GPIO 15 (UART RX of Raspberry)
- CH340 RXD PIN to GPIO 14 (UART TX of Raspberry)
- CH340 GND PIN to GND (PIN 6 of Raspberry)

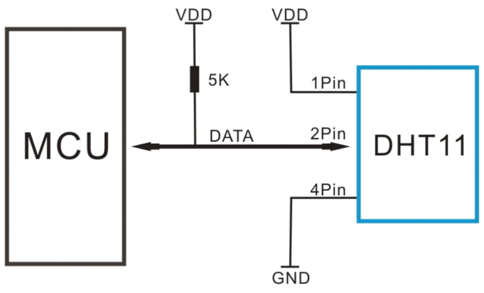


## 2.3 DHT-11 Humidity and Temperature Sensor

This small sensor can make temperature and humidity measurements with a calibrated digital output signal. It is not among the best on the market for accuracy and reliability, but it does its job well. The calibration coefficients are stored as program in the OTP memory, which are used by the sensor's internal signal detecting process. The single-wire serial interface makes system integration quick and easy. Its small size, low power consumption and up-to-20 meter signal transmission making it the best choice for various applications.

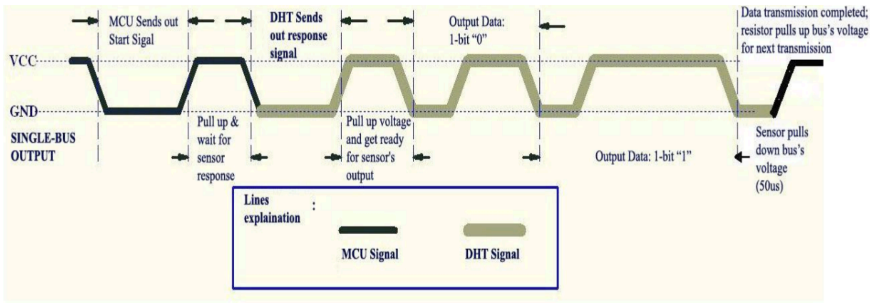


The sensor works with a 5V supply, a 5K resistor is recommended, but in my case I used a 10K resistor. A typical application schematic is as follows:



Communication is via single-bus format, in total communication takes about 4ms and includes 40bit. The transmitted data is divided into integer part and decimal part, in detail we have: 8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit check sum. For design purposes at the end only the respective integer parts of temperature and humidity will be used.

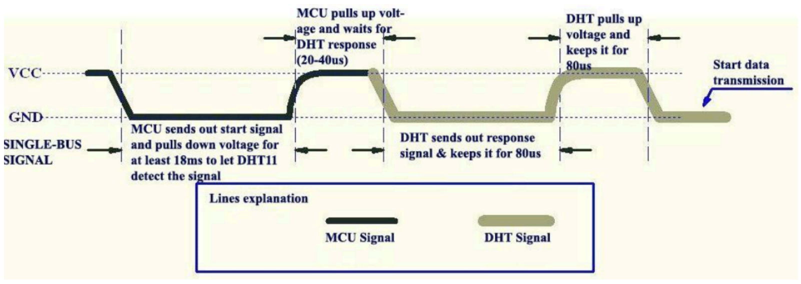
The overall communication process can be summarized as it follows: when MCU sends a start signal, DHT11 changes from the low-power-consumption mode to the running-mode, waiting for MCU completing the start signal. Once it is completed, DHT11 sends a response signal of 40-bit data that include the relative humidity and temperature information to MCU. Users can choose to collect (read) some data. Without the start signal from MCU, DHT11 will not give the response signal to MCU. Once data is collected, DHT11 will change to the low power-consumption mode until it receives a start signal from MCU again.


MCU Sends out Start Sigal
DHT Sends out response signal
Output Data: 1-bit "0"
Data transmission completed; resistor pulls up bus's voltage for next transmission
VCC
GND
SINGLE-BUS OUTPUT
Pull up & wait for sensor response
Pull up voltage and get ready for sensor's output
Output Data: 1-bit "1"
Sensor pulls down bus's voltage (50us)
Lines explaination :
MCU Signal
DHT Signal

We can divide this process into two phases:

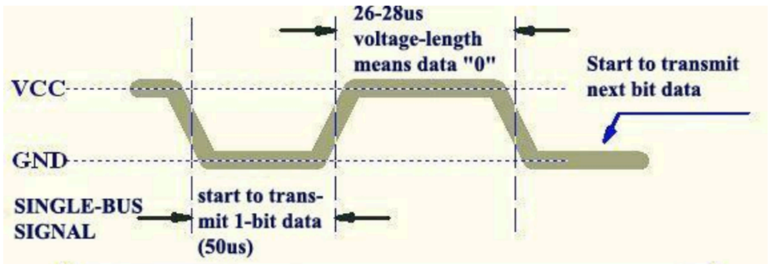
## MCU'S START SIGNAL

Data Single-bus free status is at high voltage level. When the communication between MCU and DHT11 begins, the program of MCU will set Data Single-bus voltage level from high to low and this process must take at least 18ms to ensure DHT's detection of MCU's signal, then MCU will pull up voltage and wait 20-40us for DHT's response.


MCU pulls up voltage and waits for DHT response (20-40us)
DHT pulls up voltage and keeps it for 80us
VCC
GND
Start data transmission
SINGLE-BUS SIGNAL
MCU sends out start signal and pulls down voltage for at least 18ms to let DHT11 detect the signal
DHT sends out response signal & keeps it for 80us
Lines explanation
MCU Signal
DHT Signal

## DHT-11 SENDS RESPONSE TO MCU

Once DHT detects the start signal, it will send out a low-voltage-level response signal, which lasts 80us. Then the program of DHT sets Data Single-bus voltage level from low to high and keeps it for 80us for DHT's preparation for sending data. When DATA Single-Bus is at the low voltage level, this means that DHT is sending the response signal. Once DHT sent out the response signal, it pulls up voltage and keeps it for 80us and prepares for data transmission. When DHT is sending data to MCU, every bit of data begins with the 50us low-voltage-level, and the length of the following high-voltage-level signal determines whether data bit is "0" or "1", 26-28us means 0 and 70us signal means 1.


26-28us voltage-length means data "0"
Start to transmit next bit data
VCC
GND
SINGLE-BUS SIGNAL
start to trans-mit 1-bit data (50us)

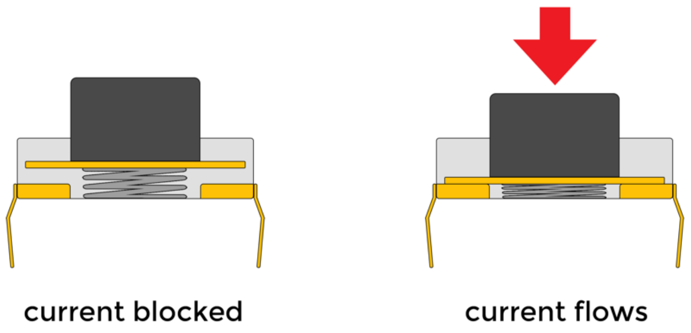
The sensor is connected to the raspberry pi through the breadboard in the following way:

- DHT-11 VCC PIN to 5v Power of Raspberry Pi (PIN 2)
- DHT-11 DATA PIN to GPIO 17 (PIN 11 of Raspberry Pi)
- DHT-11 GND PIN to Ground of Raspberry Pi (PIN 14)

## 2.4 Push Button B3F-1020

This button is a simple 4-pin tactical switch where the pins are connected 2 by 2. It is composed of a top cover, a plunger, which is the part that is pressed, the contact dome which is the part where the contact and closing of the circuit takes place (it is also the part that provides that audible feedback when the button is pressed) and finally the molded resin base.

When the button is in the starting state, there is no contact and no current flows, at the time of pressing and for the duration of pressing, current flows from one side to the other.



The button is connected to Raspberry Pi through the breadboard in this way:

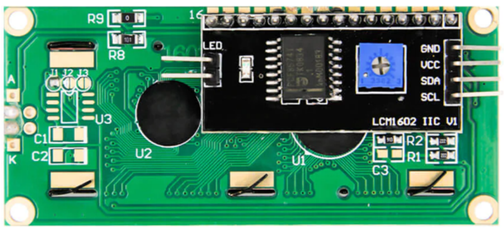
- One side of the button to GPIO 24 of Raspberry Pi (PIN 18)

## 2.5 LCD 1602

LCD1602, or 1602 character-type liquid crystal display, is a kind of dot matrix module to show letters, numbers, and characters and so on. There's a dot pitch between two characters and a space between lines, thus separating characters and lines. The model 1602 means it displays 2 lines of 16 characters. Generally, LCD1602 has parallel ports, that is, it would control several pins at the same time. LCD1602 can be categorized into eight-port and four-port connections.

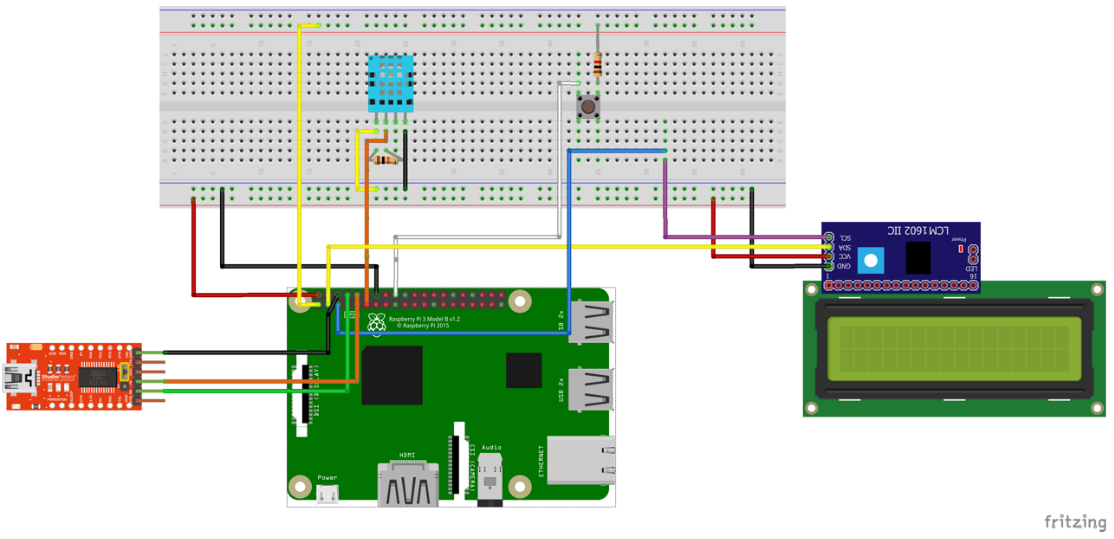
## 2.6 I2C Interface for LCD 1602

This module allows a great reduction in terms of pins used by this lcd, in fact basically this screen has 16 Pins to connect while with the i2c interface we are going to use only 4:



- VCC Pin to 5v Power of Raspberry Pi (PIN 2)
- SDA Pin to GPIO 2 (I2C SDA Pin 3 of Raspberry Pi)
- SCL Pin to GPIO 3 (I2C SCL Pin 5 of Raspberry Pi)
- GND Pin to Ground of Raspberry Pi (PIN 14)

## 2.7 Wiring Diagram



This diagram best represents the connections for proper program operation. In my case, I used a UART USB adapter model CH340, but since I could not find this specific component on the circuit design software, I took another generic one.

## 3 Environment and Program startup

### 3.1 pijFORTHos

PijFORTHos is a bare-metal operating system for Raspberry Pi, based on JonesForth-ARM.
The interpreter uses the RPi serial console (115200 baud, 8 data bits, no parity, 1 stop bit). If you have *pijFORTHos* on an SD card in the RPi, you can connect it to another machine (even another RPi) using a USB to serial cable. When the RPi is powered on (I provide power through the cable), a terminal program on the host machine allows access to the FORTH console.

### 3.2 Minicom and Picocom

Minicom is a text-based model control and terminal emulator program for Unix-like operating systems, it's used to setup a remote serial console. Picocom is a minimal dumb-terminal emulation program. It is, in principle, very much like minicom. It was designed to serve as a simple, manual, modem configuration, testing, and debugging tool. It has also served as a low-tech serial communications program to allow access to all types of devices that provide serial consoles.

To start the communication with Raspberry Pi, use this command:
**picocom --b 115200 /dev/tty.usbserial-1120 --imap delbs -s "ascii-xfr -sv -l100 -c10"**

- -bb 115200 defines the baud rate (the rate at which information is transferred in the communication channel)
- /dev/tty.usbserial-1120 specifies the UART port
- --imap delbs maps DELETE on backspace
- -s “ascii-xfr -sv -1100 -c10” specifies external program used to transmit files
- -sv specifies verbose mode (specifies that I want to display detailed processing information on screen)
- -l100 specifies the delay (100 milliseconds) after each line transmitted
- -c10 specifies the delay (10 milliseconds) between each character transmitted

### 3.3 Starting the project

After following the diagram for proper component connection, you can move on to booting the project (assuming you already have the kernel on the SD card along with the other boot files and have made the appropriate changes to the config.txt file).

- First thing to do is to connect the USB UART adapter to the computer and run the previously written command: picocom --b 115200 /dev/tty.usbserial-1120 --imap delbs -s "ascii-xfr -sv -l100 -c10" where /dev/tty.usbserial-1120 is my serial port (change with yours).
- Connect the power supply to Raspberry Pi (I use the computer to power the Raspberry to avoid too high current voltages) and wait for pijFORTHos boot up.
- After that try typing on the keyboard to verify that the UART adapter is working properly

- Use CTRL+A and CTRL+S to load program files (load them manually or load only the file named “special.f”)
- If you have done everything correctly you will only need to use the SETUP words (to initialize i2c and lcd) and finally LAUNCH for the actual startup of the program.
- You can use the button to change modes between Celsius, Kelvin and Off.

## 4 Code structure and implementation choices

### 4.1 Code structure

The program code is divided into the following files:

- jonesforth.f defines standard jonesforth words for strings, number conversions and other…
- utils.f contains useful words for working with GPIO registers, easily derive these registers, select operation modes quickly, define time delays, and also define constants for gpio pins.
- i2c.f contains words to initialize and work with i2c through the BSC controller (broadcom serial control).
- lcd.f contains words to convert to ASCII characters and to send data and commands to the lcd.
- temphum.f provide functions to communicate with DHT-11 temperature and humidity sensor
- main.f is the main file of the program. It contains the main loop and the functions to change the mode of the system.

### 4.2 Utils.f

In this file I have collected the main functions that are "useful" and reused in other files. From defining constants to easily operate with GPIO registers to defining functions to set and reset these registers. I also included here functions regarding timers and delays because being a few lines I did not want to create a separate file.

```
HEX

3F000000 CONSTANT PERI_BASE    \ Base address of peripherals
3F200000 CONSTANT GPIO_BASE    \ Base address of GPIO peripheral
GPIO_BASE CONSTANT GPFSEL       \ GPIO Function Select
GPIO_BASE 1C + CONSTANT GPSET   \ GPIO Pin Output Set
GPIO_BASE 28 + CONSTANT GPCLR   \ GPIO Pin Output Clear
GPIO_BASE 34 + CONSTANT GPLEV   \ GPIO Pin Level
3000      CONSTANT TIMER_OFFSET \ Offset for the System Timer
PERI_BASE TIMER_OFFSET + 04 + CONSTANT TIMER \ Creates constant for the System Timer
Counter Lower bits.

VARIABLE LAST_TIME \ Creates variabile to store last time read.

00          CONSTANT LOW \ Creates constant for LOW bit value.
```

```
01          CONSTANT HIGH \ Creates constant for HIGH bit value.

00          CONSTANT INPUT \ Creates constant for input function.

01          CONSTANT OUTPUT \ Creates constant for output function.

04          CONSTANT ALT0   \ Creates constant for alternate function 0.

\ Multiplies a number (hex) by 4 (shifts left by 2) to refer to word offsets of GPIO
registers
: >WORD ( n -- word )
2 LSHIFT ;

\ Returns GPFSEL register address for a given GPIO pin (hex)
: GPFSEL_ADDR ( pin -- addr )
0A /
>WORD GPIO_BASE + ;

\ Returns GPSET register address for a given GPIO pin.
: GPSET_ADDR ( gpio_pin_number -- gpio_pin_address )
    20 /                                 \ GPSET register number
    >WORD GPSET + ;

\ Returns GPCLR register address for a given GPIO pin.
: GPCLR_ADDR ( gpio_pin_number -- gpio_pin_address )
    20 /                                 \ GPCLR register number
    >WORD GPCLR + ;

\ Returns GPLEV register address for a given GPIO pin.
: GPLEV_ADDR ( gpio_pin_number -- gpio_pin_address )
    20 /                                 \ GPLEV register number (word of 32 bits)
    >WORD GPLEV + ;                      \ Gets the address

\ Returns a 32 bit word with just one bit high in the proper positon for a GPIO pin.
: BIT>WORD ( gpio_pin_number -- bit_word )
    20 MOD                               \ The result is mod 32 (0-31)
    01 SWAP LSHIFT ;                    \ Swap and shit 01 by the result of MOD
obtaining all 0 except 1 position

\ Returns 0 or 1 depending on the value of the bit in a given position of a given 32 bit
word.
: WORD>BIT ( bit_word bit_number -- bit_value )
    RSHIFT 01 AND ;

\ Copies the top of the return stack wihout affecting it
: R@ ( -- top_of_return_stack )
    R> R> TUCK >R >R ;

\ Creates mask for a given GPIO pin.
: MASK ( gpio_pin_number -- mask )
```

```
    0A MOD                                    \ Offset (base 10) for gpio_pin_number in
GPFSEL contents
    DUP 2* +                                  \ Multiplies by 3 to get the real offset
    07 SWAP LSHIFT INVERT ;                   \ Returns the mask (INVERT inverts bits
value)

\ Sets a GPIO output high for a given GPIO pin.
: SET_HIGH ( gpio_pin_number -- )
    DUP BIT>WORD                              \ Gets the right bit position to update the
contents of GPSET
    SWAP GPSET_ADDR ! ;                       \ Updates the content of GPSET

\ Sets a GPIO output low for a given GPIO pin.
: SET_LOW ( gpio_pin_number -- )
    DUP BIT>WORD                              \ Gets the right bit position to update the
contents of GPCLR
    SWAP GPCLR_ADDR ! ;                       \ Updates the contents of GPCLR


\ Returns 0 or 1 depending on the level (low or high) of a specific GPIO pin when set as
input.
: READ ( gpio_pin_number -- )
    DUP GPLEV_ADDR @                               \ Gets the contents of GPLEV register
    SWAP WORD>BIT ;

\Clears specified bits of a given word and a pattern.
: BIC ( word pattern -- word_clear_bits )
    INVERT AND ;

\ Returns a configuration for a GPFSEL contents update given a functionality in 0-7 and
a GPIO pin number.
\IT ONLY RETURNS, DOESNT APPLY THE CONFIGURATION
\000 = GPIO Pin is an input
\001 = GPIO Pin is an output
\100 = GPIO Pin takes alternate function 0
\101 = GPIO Pin takes alternate function 1
\110 = GPIO Pin takes alternate function 2
\111 = GPIO Pin takes alternate function 3
\011 = GPIO Pin takes alternate function 4
\010 = GPIO Pin takes alternate function 5 ;
: CONFIGURATION ( functionality_number gpio_pin_number -- configuration_for_GPFSEL )
    0A MOD                                    \ Offset (base 10) for gpio_pin_number in
GPFSEL contents
    DUP 2* +                                  \ Multiplies by 3 to get the real offset
    LSHIFT ;                                  \ Return the content to update pin's
functionality



\ Configures a specific functionality for a GPFSEL given its number and the
functionality in 0-7.
\001 = GPIO Pin is an output
\100 = GPIO Pin takes alternate function 0
```

```
\101 = GPIO Pin takes alternate function 1
\110 = GPIO Pin takes alternate function 2
\111 = GPIO Pin takes alternate function 3
\011 = GPIO Pin takes alternate function 4
\010 = GPIO Pin takes alternate function 5 ;
: CONFIGURE ( gpio_pin_number functionality_number -- )
    SWAP DUP GPFSEL_ADDR >R                     \ Gets GPFSEL register address and stores
in the return stack
    DUP MASK                                    \ Gets the mask for gpio_pin_number
    R@ @ AND                                    \ Cleans up the GPFSEL register contents
for gpio_pin_number fetching the value of address stored in return stack
    -ROT CONFIGURATION OR                       \ Updates the contents to set up the
functionality
    R> ! ;                                      \ Stores the new value in the GPFSEL
register address

\ Starts the timer by storing the time read in LAST_TIME.
\ Usage: TIMER START
: START ( timer_address -- )
    @ LAST_TIME ! ;                    \ Stores the time read

\ Stops the timer by subtracting the time stored in LAST_TIME to the current time.
\ Usage: TIMER STOP
: STOP ( timer_address -- time_in_us )
    @ LAST_TIME @ - ;                  \ Gets the time passed

\ Delays by a certain amount of time.
: DELAY ( delay_amount_in_us -- )
    TIMER START
    BEGIN
        DUP
        TIMER STOP                     \ Gets the time passed
        <                              \ Checks if the time passed is less than the delay
amount
    UNTIL DROP ;
```

## 4.3 I2c.f

This file contains the definition of constants to work with the Broadcom Serial Controller 1 registers, functions to set/reset some registers, and send data to the lcd screen via i2c.

```
\Load file after utils.f
HEX

\ BSC1 (Broadcom serial control 1) is the reference master, masters 2 and 7 aren't
accessible for the user.
804000 CONSTANT BSC1       \ Base address of BSC1 register

\This offsets are relative to the base address of BSC1 register.
\See BCM2711 ARM Peripherals for more details.
```

```
BSC1 PERI_BASE +             CONSTANT CTRL
BSC1 PERI_BASE + 04 +        CONSTANT STATUS
BSC1 PERI_BASE + 08 +        CONSTANT DLEN
BSC1 PERI_BASE + 0C +        CONSTANT SLAVE
BSC1 PERI_BASE + 10 +        CONSTANT FIFO
BSC1 PERI_BASE + 14 +        CONSTANT DIV
BSC1 PERI_BASE + 18 +        CONSTANT DEL
BSC1 PERI_BASE + 1C +        CONSTANT CLKT

\Set slave address
: SET_SLAVE ( addr -- )
  SLAVE ! ;

\Set amount of data bytes to be transmitted
: SET_DLEN ( n -- )
  DLEN ! ;

\Set FIFO data in 8 bits at time to transmit on the bus
: APPEND_FIFO ( n -- )
  FIFO ! ;

\Reset CTRL register without changing reserved bits.
\Reserved bits are bits 31:16, 14:11, 6, and 3:1 (see BCM2711 ARM Peripherals);
: RESET_CTRL ( -- )
  CTRL @ 87B1 BIC CTRL ! ;

\Reset STATUS register without changing reserved bits.
\Reserved bits are 31:10 (see BCM2711 ARM Peripherals);
\Use 302 because other bits are read-only flags. 9 is CLKT, 8 is ERR, 1 is DONE.
\Set bits at 1 because they are cleared by writing 1 to them.
: RESET_STATUS ( -- )
  STATUS @ 302 OR STATUS ! ;

\Clear FIFO register without changing reserved bits.
\Sets bits 5:4 of CTRL register to X1 or 1X to clear before the new frame starts.
: CLEAR_FIFO ( -- )
  CTRL @ 10 OR CTRL ! ;

\Set the CTRL register to start the transmission.
\Need to set all bits to 0 except 15 (I2CEN) to enable the BSC controller;
\and 7 (TA) to start the transmission.
\Interrupts are disabled.
: TRANSFER ( -- )
  CTRL @ 8080 OR CTRL ! ;

\Transfers the data through the I2C bus.
\The data is sent in 8 bits at time.
: >I2C
  RESET_STATUS
  RESET_CTRL
  CLEAR_FIFO
  01 SET_DLEN
  APPEND_FIFO
  TRANSFER ;

\Set up I2C bus and slave address.
\Use GPIO pin 2 for Serial Data (SDA) and pin 3 for Serial Clock (SCL). This functions
\are defined on respective ALT0 function of the pins.
\My LCD address, according to a I2C scan, is 0x27 so I set it as slave address.

: INIT_I2C
  02 ALT0 CONFIGURE
  03 ALT0 CONFIGURE
  27 SET_SLAVE ;

\Our data transfer structure is: D7 D6 D5 D4 BL EN RW RS, to send a byte we need to
divide it
\ in two parts (nibbles) and send them in two different frames followed by a combination
of
\ BL EN RW RS bits. The first frame is the most significant nibble and the second frame
is
\ the least significant nibble.
\If a byte is a part of a command (RS = 0) then RW is 0 and EN is 1. And the transfer is
obtained
\by sending: HIGH 1 1 0 0 -> HIGH 1 0 0 0 -> LOW 1 1 0 0 -> LOW 1 0 0 0;
\If a byte is a part of a data (RS = 1) then RW is 0 and EN is 1. And the transfer is
obtained
\by sending: HIGH 1 1 0 1 -> HIGH 1 0 0 1 -> LOW 1 1 0 1 -> LOW 1 0 0 1;
\RW is always 0 because we are writing to the LCD display.

\Define a word that returns settings parts to be sent based on a truth value that
indicates
\if the byte is a part of a command or a data.
: SETTINGS ( value -- setting_1 setting_2 )
  IF
    0C 08   \Command
  ELSE
    0D 09   \Data
  THEN ;

\Given setting_1 and setting_2 returns a nibble linked to the settings.
: COMPOSE ( setting_1 setting_2 nibble  -- nibble_setting_2 nibble_setting_1 )
  04 LSHIFT DUP \setting_1 setting_2 nibble_to_byte nibble_to_byte
  ROT OR        \setting_1 nibble_to_byte nibble_to_byte_setting_2 (OR between setting_2
and nibble_to_byte)
  -ROT OR ;     \nibble_to_byte_setting_2 nibble_to_byte_setting_1 (OR between setting_1
and nibble_to_byte)

\Returns lower nibble of a byte (LSB nibble).
: >NIBBLE ( byte -- l_nibble )
  0F AND ;

\Divides a byte in two nibbles.
: BYTE>NIBBLES ( byte -- l_nibble h_nibble )
```

```
  DUP >NIBBLE \Returns l_nibble
  SWAP 04 RSHIFT >NIBBLE ; \Returns h_nibble

\Sends a nibble to LCD display with the settings (expressed as a truth value).
: SEND_NIBBLE ( nibble value -- )
  SETTINGS ROT \setting_1 setting_2 nibble
  COMPOSE      \gets the two bites to send (nibbles aggregated with settings)
  >I2C 1000 DELAY
  >I2C 1000 DELAY ;

\Transmit input to LCD given instruction or data.
: >LCD ( value -- )
  DUP 08 WORD>BIT >R \Stores command/data bit in R stack
  BYTE>NIBBLES R@    \Creates two nibbles from the byte
  SEND_NIBBLE R>     \Sends the first nibble (h_nibble)
  SEND_NIBBLE ;       \Sends the second nibble (l_nibble)
```

## 4.4 Temphum.f

This file contains everything needed to communicate with the dht-11 sensor and the use of the button. A complete exchange of information takes 4ms and the sensor respects precise timing. With the SET-DHT11 function I start the sensor preparation by sending the signal from the MCU and setting the sensor input at the end. The DHT-START function checks that the MCU receives the 80us signal (high and low) and proceeds with reading 40 bits.

```
HEX

\Load after i2c.f file

\This file provide functions to communicate with DHT-11 temperature and humidity
\sensor.

\Define the GPIO pin to which the DHT11 is connected
11 CONSTANT DHT11
18 CONSTANT BUTTON

\Define variables to store the temperature and humidity
VARIABLE DATAS
VARIABLE TEMP_INT
VARIABLE HUMIDITY_INT
VARIABLE TEMP_KELVIN

\Defines function to prepare DHT-11 to send data
: SET-DHT11 ( -- )
    DHT11 01 CONFIGURE          \Set the pin to output
    DHT11 SET_LOW               \Set the pin to low
    4650 DELAY                  \Wait 18ms (expressed in us to use the DELAY function)
    DHT11 SET_HIGH              \Set the pin to high
    DHT11 00 CONFIGURE ;        \Set the pin to input
```

```
\Set the GPIO pin of button to input mode
: SET-BUTTON ( -- )
    BUTTON 00 CONFIGURE ;

\Defines a function that waits as long as sensor sends a low signal
\It's used to check the 80us low signal sent by the sensor to signal the start of the
data
: WAIT-LOW ( -- )
    BEGIN
        DHT11 READ
        LOW =
    WHILE
    REPEAT ;

\Defines a function that waits as long as sensor sends a high signal
\It's used to check the 80us high signal sent by the sensor to signal the start of the
data
: WAIT-HIGH ( -- )
    BEGIN
        DHT11 READ
        HIGH =
    WHILE
    REPEAT ;

\Gets data from DHT11
: DHT-START ( -- )
    WAIT-LOW                                \Wait for the sensor to send a low signal
    WAIT-HIGH                               \Wait for the sensor to send a high signal
    1F BEGIN                                \Until we have read 5 bytes
        DATAS DUP @ 1 LSHIFT SWAP !         \Shift the bits to the left and store the value
in DATAS
        WAIT-LOW TIMER @                    \Wait for the sensor to send a low signal and
read the timer
        WAIT-HIGH TIMER @                   \Wait for the sensor to send a high signal and
read the timer
        SWAP - 32 >                         \Subtract the two timers and check if the result
is greater than 32 (50us between two bits)
        IF                                  \If the result is greater than 32 (50us passed
between two bits)
            DATAS DUP @ 1 OR SWAP !         \Add a 1 to the DATAS variable
        THEN
            1 - DUP 0 >                      \Decrement the counter and check if it is
greater than 0
        WHILE
        REPEAT
        DROP ;

\Memorize the integer part of humidity
: READ-HUMIDITY-INT ( -- )
    DATAS @
    18 RSHIFT               \Shift the bits to the right to get the integer part of
humidity
    HUMIDITY_INT ! ;

\Memorize the integer part of temperature
: READ-TEMP-INT ( -- )
    DATAS @
    8 RSHIFT FF AND        \Shift the bits to the right to get the integer part of
temperature
    DUP DUP 13 >           \Check if the value is greater than 19
    SWAP 32 <              \Check if the value is lower than 50
    AND  .S                 \Check if the value is between 19 and 50
    IF
        TEMP_INT !         \If the value is between 19 and 50, store it in TEMP_INT
    ELSE
        DROP
    THEN ;

\Reads data from DHT11
: READ-DATA ( -- )
    READ-HUMIDITY-INT
    READ-TEMP-INT ;

\Converts the temperature from Celsius to Kelvin
: CELSIUS-TO-KELVIN ( -- )
    TEMP_INT @ 111 + TEMP_KELVIN ! ;

\Prints the humidity
: PRINT-HUMIDITY ( -- )
    DECIMAL
    ." Humidity : " HUMIDITY_INT ?  ." % " CR
    HEX ;

\Prints the temperature
: PRINT-TEMPERATURE ( -- )
    DECIMAL
    ." Temperature : " TEMP_INT ? ." ° C" CR CR
    HEX ;

\Prints temperature in Kelvin scale
: PRINT-TEMPERATURE-KELVIN ( -- )
    DECIMAL
    ." Temperature : " TEMP_KELVIN ? ." ° K" CR CR
    HEX ;

\Prints all
: PRINT-DATA ( -- )
    PRINT-HUMIDITY
    PRINT-TEMPERATURE
    PRINT-TEMPERATURE-KELVIN ;

\Make a measurement and print the values
```

```
: MEASURE ( -- )
    0 DATAS !
    0 TEMP_KELVIN !
    SET-DHT11
    DHT-START
    READ-DATA
    CELSIUS-TO-KELVIN
    PRINT-DATA ;
```

### 4.5 Lcd.f

Lcd.f contains basic commands for the lcd screen (initialization, cleaning, pointer moves) and functions for printing strings, decimal numbers, and generally temperature and humidity measurements.

```
HEX

\Load this file after temphum.f

\This files provide words to work with the LCD display.

\Returns ASCII code of 0.
: '0' ( -- 0_ascii )
  [ CHAR 0 ] LITERAL ;

\Set up the LCD to 4 bits mode and turn on/off cursor and cursor position.
: INIT_LCD ( -- )
    102 >LCD        \ 4 bits mode
    10C >LCD ;       \ Turn on/off cursor and cursor position

\Clear the LCD screen.
: CLEAR_LCD ( -- )
    101 >LCD ;         \ Clear LCD screen

\Move cursor to first cell of first line. RH stands for Return Home.
: RH_LINE1 ( -- )
    102 >LCD ;         \ Move cursor to first cell of first line

\Move cursor to first cell of second line. RH stands for Return Home.
: RH_LINE2 ( -- )
    1C0 >LCD ;          \ Move cursor to first cell of second line

\Print strings on the LCD.
: PRINT-STRING ( address length -- )
    OVER + SWAP     \address_last_char+1 address_first_char
    BEGIN           \While there are chars to print
        DUP C@ >LCD \Print char at address_first_char location
        1+          \add 1 to address_first_char
        2DUP =      \are we done? String is terminated?
    UNTIL 2DROP ;   \If not, repeat.
```

```
DECIMAL \Convert the base to decimal base.

: DEC>ASCII ( 1_digit_dec -- 1_digit_ascii )
    '0' + ; \Add 0_ascii symbol to get specific digit ascii symbol

\Print a 2 digits decimal number on the LCD.
: PRINT-DEC ( 2_digits_dec -- )
    DUP 10 / DEC>ASCII >LCD \Print first digit
    10 MOD DEC>ASCII >LCD   \Print second digit
    HEX ;

\Print a 3 digits decimal number on the LCD.
: PRINT-DEC3 ( 3_digits_dec -- )
    DUP 100 / DEC>ASCII >LCD \Print first digit
    DUP 10 / 10 MOD DEC>ASCII >LCD \Print second digit
    10 MOD DEC>ASCII >LCD   \Print third digit
    ;

\Print the humidity value on the LCD.
: HUMID>LCD ( -- )
    RH_LINE1
    S" Humidity: " PRINT-STRING
    HUMIDITY_INT @ PRINT-DEC
    S" % " PRINT-STRING ;

\Prints the temperature in Celsius to LCD.
: CELS>LCD ( -- )
    RH_LINE2
    S" Temp: " PRINT-STRING
    TEMP_INT @ PRINT-DEC
    223 >LCD \Print degree symbol
    S" C" PRINT-STRING ;

\Prints the temperature in Kelvin to LCD.
: KELV>LCD ( -- )
    RH_LINE2
    S" Temp in K: " PRINT-STRING
    TEMP_KELVIN @ PRINT-DEC3
    223 >LCD \Print degree symbol
    S" K" PRINT-STRING ;

\Prints the two measures using Celsius for temperature.
: DATA-CELSIUS ( -- )
    CLEAR_LCD
    HUMID>LCD
    CELS>LCD ;

\Prints the two measures using Kelvin for temperature.
: DATA-KELVIN ( -- )
    CLEAR_LCD
    HUMID>LCD
    KELV>LCD ;
```

## 4.6 Main.f

You need this file to start the program since it contains the system SETUP and LAUNCH functions.

```
HEX

\Load after temphum.f

\This file is the main file of the program. It contains the main loop and the
\functions to change the mode of the system.

\ Creates constant for 2 seconds delay.
1E8480 CONSTANT 2SECONDS

\The systems can switch between CLEAR (0), CELSIUS (1), KELVIN (2).
VARIABLE MODE

\Change mode of the system.
: CHANGE-MODE ( -- )
    MODE @ 1 + 3 MOD MODE ! ;

\ Welcome quote to start the program.
: QUOTE ( -- )
    CLEAR_LCD
    RH_LINE1
    S" Sic Parvis Magna " PRINT-STRING
    RH_LINE2
    2SECONDS DELAY
    CLEAR_LCD
    RH_LINE1
    S" Waiting for "        PRINT-STRING
    2SECONDS DELAY
    RH_LINE2
    S" the Launch... "       PRINT-STRING ;

\Setup the system to work.
: SETUP ( -- )
    INIT_I2C        \ Initializes the I2C interface
    INIT_LCD        \ Initializes the LCD
    SET-BUTTON
    QUOTE ;

: WORK-CELSIUS ( -- )
    MEASURE        \ Measures the temperature and humidity
    DATA-CELSIUS ; \ Prints the temperature in Celsius and humidity in %

: WORK-KELVIN ( -- )
    MEASURE         \ Measures the temperature and humidity
    DATA-KELVIN ;   \ Prints the temperature in Kelvin and humidity in %

: LAUNCH ( -- )
    BEGIN
```

```
        2SECONDS DELAY BUTTON READ  \ Waits for 2 seconds and reads the button
        IF                          \ If the button is pressed
            CHANGE-MODE             \ Changes the mode
        THEN
        MODE @ 1 =                  \ If the mode is Celsius
        IF                          \ Then
            WORK-CELSIUS            \ Work in Celsius
        ELSE                        \ Else if
        MODE @ 2 =
        IF
            WORK-KELVIN             \ Work in Kelvin
        ELSE
            CLEAR_LCD
        THEN
        THEN
    AGAIN ;
```

## 5 Conclusions

This simple project took about 6 weeks to complete, and we thank Professor Peri for providing essential tools for its development. The project itself is intended to be the conclusion of a course of study that has always centered on the importance of embedded systems regarding the resources they require and the extensive functionality they offer every day.
I am sure that this program, excellent as it is, can find other optimizations in the future, it will be up to me or those who wish to contribute on Github, for that matter, as the project itself proves: Sic Parvis Magna (so, from small things, great things).

## 6 Bibliography

- slides by professor peri (teaching materials A.Y. 2022/23)
- Datasheet BCM2711 Raspberry Pi
- https://pinout.xyz/
- Datasheet DHT-11 Humidity and Temperature Sensor
- Datasheet Tacticle Switch B3F-1020
- https://www.electronicwings.com/8051/lcd16x2-interfacing-in-4-bit-mode-with-8051
- Starting Forth, Leo Brodie (PDF Version)
- https://github.com/organix/pijFORTHos