Preface

This document describes the hardware, firmware and software specifications of the AquTemp project. Effort was made for this document to contain all the necessary information without errors in a form that can help the novice user as well as the experienced one. The language used is as simple as possible in order for everyone to understand the functions of the project. All information presented here provided “as is”. No responsibility/liability is held true if any error or damage occur by the usage of the data presented in this project. You may use all info for personal use only – as always at your own risk!

Contact info, bugs report and other info

If any bugs or errors found, please report them to nikos@tng.gr (alternative email is nbovos@gmail.com) in order to be corrected as soon as possible.
For any questions feel free to send me an email, I will reply as soon as possible.
Please note, the construction and usage of data of this project is at your own risk.
Donations via PayPal to keep coding to nikos@tng.gr would be very much appreciated :-)

This document was written using Apache Open Office™ 4.1.3 on my DELL D630 Laptop, running Microsoft Windows 7 Home Premium© and exported to PDF via the PDF export function.

Usually I'm told that i over-analyze things, I'm afraid I might do it also here, so I apologize in advance for that...

Open Office first build by Sun microsystems & the community (now acquired by Apache Software Foundation)
Apache Open Office is a trademark of Apache Software Foundation
Microsoft, Windows, Windows 7 Home Premium are registered trademarks of Microsoft Corporation

What is AquTemp ?

AquTemp is an acronym for “Aquarium Temperature Guardian”. It is build around an 8-bit MCU and is used to monitor my aquarium's water temperature (which has volume of 120 liters with dimensions 100cm X 30cm X 40cm) and automate the cooling of the water, in order to make my life and fishes life easier. The MCU selected for this project is ATtiny2313*, primarily because I have a bunch of them available and has exactly the I/O pins needed for this project.

Basic specifications of the project

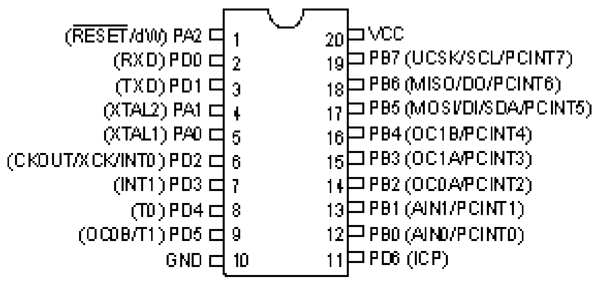
1) Temperature measurement by using DS18B20* digital temperature sensor.
2) Programmable temperatures for start and stop of fans.
3) Cooling the water temperature by using two 12V DC fans
4) Display of current temperature using two 7-Segment displays.
5) Input from user by using 3 push-buttons (Set, Next, Prev)
6) Can be programmed with the use of a Personal Computer (PC) using the dedicated Aquarium Temperature Guardian software* (compatible with Microsoft Windows© Operating Systems)
7) Can be programmed by the three user buttons also (Set, Next, Prev)
8) With the PC-Software can log aquarium's temperature to file.
9) MCU clock frequency = 4MHz internal clock.
10) Communication to PC at 19200bps, 8-data bits, 1-stop bit and Even-parity.

This project was really a sub-project of a “bigger brother” (the Aquarium Guardian), which started before that one, using ATmega162 and additionally had programmable lightning and feeding of fishes, real time calendar with leap years support, 2x16 LCD display and some other features, but never finished, since most of the equipment in the aquarium is programmable (lights, auto-feeder etc), so there was really no real point in creating such a big project (especially in assembly) for practically only the coolers function (heater is automatic too), so I decided to stop it, make and use this one and all other equipment as is, because summer has arrived and water temperature is rising...

Atmel and AVR are registered trademarks of Microchip Technology
ATtiny2313 is a 8bit AVR micro-controller
DS18B20 is a programmable 9bit to 12-bit temperature sensor by Maxim Integrated
AquTemp software uses loyalty free icons:IconArchive (http://www.iconarchive.com/), IconFinder (https://www.iconfinder.com) , icons created by myself.

The AquTemp micro-controller. The AVR ATtiny2313

Aquarium Temperature Guardian is built with Microchip's **ATtiny2313**. The micro-controller used is the one in PDIP package presented below (picture from datasheet):



This MCU has the AVR 8-bit RISC architecture
120 instructions, most executed in 1 clock cycle
32 general purpose registers x 8 bit each
Fully static operation
2 kBytes of In-System programmable flash
128 bytes EPROM memory
128 bytes internal SRAM memory
One 8-bit Timer/Counter
one 16-bit Timer/Counter, one Programmable serial USART and up to 20 MIPS throughput at 20MHz clock and many other features, check on Microchip's web site this microcontroller on the web page: https://www.microchip.com/wwwproducts/en/ATtiny2313

The ATtiny2313 MCU has 18 Input/Output programmable pins. It was selected mainly because the number of I/O pins needed are 17 (since $\overline{\text{RESET}}$ pin will not be used as an I/O pin) and many of these MCU still exist from past purchases/projects.

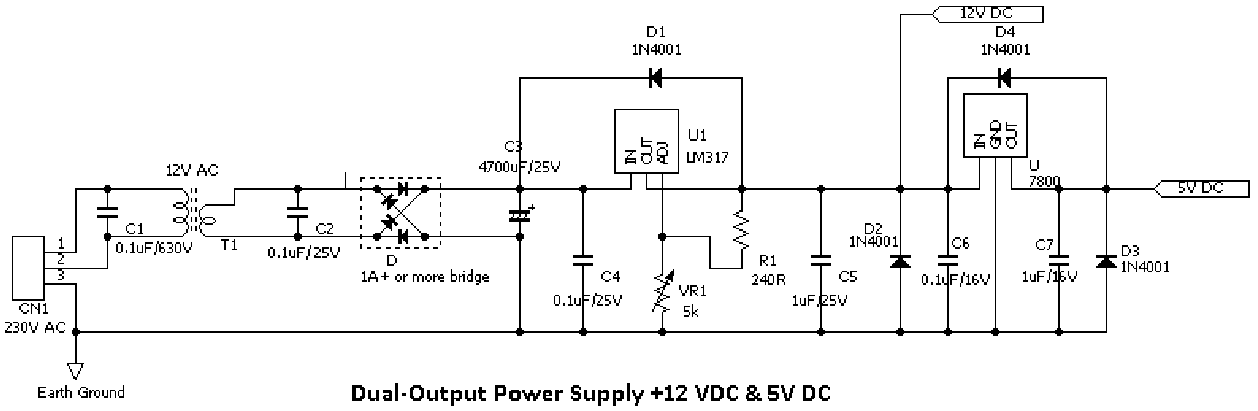
ATtiny2313 Port-Map

PORT A	Function	Direction	PORT B	Function	Direction
PA0	Decs digit driver	OUT	PB0	7-Segment dot	OUT
PA1	Units digit driver	OUT	PB1	7-Segment g	OUT
PA2	Used as $\overline{\text{RESET}}$	IN	PB2	7-Segment f	OUT
Usefull Notes a) Port A is a 3-bit Port (usually PA2 = $\overline{\text{RESET}}$) b) Port B is an 8-bit port c) Port D is an 7-bit port			PB3	7-Segment e	OUT
			PB4	7-Segment d	OUT
			PB5	7-Segment c	OUT
			PB6	7-Segment b	OUT
			PB7	7-Segment a	OUT
			🟥	🟥	🟥
			PORT D	**Function**	**Direction**
			PD0	USART RxD	IN
			PD1	USART TxD	OUT
			PD2	PREV button	IN
			PD3	NEXT button	IN
			PD4	SET button	IN
			PD5	Fans driver	OUT
			PD6	1-Wire Bus	Mixed IN / OUT

As seen, no free I/O pins left, but no update might ever be necessary for this kind of project :-)

Power requirements of project

The main power supply needed for the project is **5V DC** used by most of the peripherals (MCU, displays, temperature sensor), but because 5V fans were not available at the time of the project construction, but many **+12V DC** fans from older projects and computer systems existed, a dual power supply providing +5V DC and +12V DC was used (actually "borrowed" from previous project of mine) and +12V fans were selected for this project instead. The schematic of the power supply (made with Bsch3V 0.83.1 Software (c) 1997-2016 by H. Okada(Suigyodo) as any other presented in this document):



Dual-Output Power Supply +12 VDC & 5V DC
by Nikos Bovos - Design November 2018

It's just the standard 3-terminal regulator power supply using LM317 and a standard LM7805 or equivalent for the +5V power line. The transformer used in this project is a simple single 12V AC / 400mA power transformer, with the current capability being more than enough for all project's operations (actually it was my smallest power transformer available, so I didn't buy a new one for this project).

By using the equation of LM317 to calculate the VR_1 value $V_{OUT} = V_{REF} * (1 + \frac{R_2}{R_1})$ the value of VR1 is calculated as **VR_1 = 2064 Ω** and so a 5kΩ potentiometer was used. If a similar circuit is to be used for the +5V DC line as well, then the calculated resistor value will be Rx = 720 Ω (and thus using a 1kΩ potentiometer will do the job).

Alternatively, the dual power supply can be constructed with 7812 and 7805 voltage regulators, without having the LM317. Both of these circuits are functioning the same, the choice is a matter of taste (and availability of components)...

<u>Operation of project and implementation of MCU firmware</u>

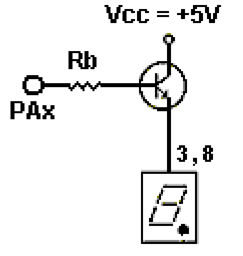
As already mentioned, two Seven Segment displays are used (specifically Kingbright's SA-52 common Anode displays) to show current water temperature and various messages to the user. While the device is operating, the current water temperature in Celsius degrees is displayed. Two temperature limits (Low and High) have been set in the firmware, but can also be altered by the user using 3 push buttons. These limits determine the operation of the two DC fans. When water temperature reaches high limit, fans are started. The low temperature limit determines when the DC fans will stop to operate. Driver-transistor of dc fans must provide the required power (in this project two 12V fans are used with total operating current of approximately 280mA, and so a BC639 BJT transistor used (max. current 1A) with h_{FE} = 100 and by that the base driving resistor need to be Ic/h_{FE} = Ib= 3mA, so R_B=(5V-0,65V)/3mA=1,45kΩ (1,2kΩ selected)

Temperature limits must be set according to the type of fishes that live in the aquarium. For my setup with four Carassius Auratus Auratus (goldfish as known around) temperature limits set for **Low = 22^{O}C and High = 26^{O}C**. Decimal places are really out of logic especially for this type of fish in the aquarium and project, even though the temperature sensor – a Maxim's DS18B20 have resolution from 9 up to 12-bits (0,5 degrees down to 0,0625 degrees Celsius).

And as always...My lovely quote :-)

<u>A must-say note!</u> *Although no code accompany this document, the guidelines and information provided here are more than enough for anyone with "typical knowledge" to build this project and adjust it for his/her own needs. Because I spent too much of my – not so free - time to construct, write, build, test, update etc I'm awfully irritated when I'm asked to provide the full listing for "copy & paste" from others. I'm sorry to say that, but buying 2 boards, connect them with 2 cables and writing two lines of code to your computer does not help you in any way to learn how and more important why it works (even if that LCD screen is displaying 'Hello world !')...It's better to try and fail and then to repeat until you get it and learn a lot in the process, than to get the "working code" from others without even scratching your brain on how to do it.*

Nevertheless, <u>in this project some code will be provided</u> to help others that might not have used some peripherals in Assembly. As in the port-map shown, PA0 will drive the MSD of display and PA1 the LSD display. Since the seven segment displays used are common anode, enabling and disabling each digit is done by providing or removing the power supply (pin 3 or pin 8 whichever is used). The simple circuit shown below does that:



7-Segment Driver

The BJT used in the project is BC547 (one for each digit).
Typical values for h_{FE} are 110 to 800 (depending on collector current)
For the displays used, ~ 20mA operating current selected and for the (measured) h_{FE} of 220 the base resistor Rb can be calculated from the base current needed, as Ib = 20mA / 220 = 90 μA and so base resistor will be Rb = 5V – V_{BE} / 90μA which gives approx. Rb = 47 kΩ.
Selected resistor can be smaller, since each digit will be driven at a frequency higher than DC (and intensity becomes less as frequency increases). Value of PAx uses positive logic (MCU Port = High to enable and Low to disable the display). V_{BE} = 0,65V (2nd approach) for calculations.

In this project a different connection scheme to 7-Segment displays was used. Bit 0 of MCU Port was connected to dot of the LSD display, bit 1 to MSB of displays (the 'g' pin) and bit7 to 'a' pin and so a different than the normal look-up table for this kind of connection had to be implemented. This was done because it was easier to implement in the PCB. Two sets of 7-Segment equivalent values were produced, one with the dot in the LSD (Least Significant Digit) enabled and one with disabled (the 'dot' of the Most Significant Digit is left unconnected. Dot will be alternatively on and off every 1 second, to show that project is running... Since common anode displays are used, negative logic is used (high ('1') means segment is off and low ('0') means segment is on). Also two more EEPROM address locations used to hold the temperature high and low limits... All the above sum-up to the following:

a) EEPROM addresses $00 to $09 will hold 7-Segment non-dotted values
b) EEPROM addresses $10 to $19 will hold 7-Segment dotted values
c) EEPROM address $20 will hold the LOW temperature limit
d) EEPROM address $21 will hold the HIGH temperature limit

Digit	a	b	c	d	e	f	g	dot	MCU Value
or Character	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	(HEX)
0	0	0	0	0	0	0	1	1	3
1	1	0	0	1	1	1	1	1	9F
2	0	0	1	0	0	1	0	1	25
3	0	0	0	0	1	1	0	1	D
4	1	0	0	1	1	0	0	1	99
5	0	1	0	0	1	0	0	1	49
6	0	1	0	0	0	0	0	1	41
7	0	0	0	1	1	1	1	1	1F
8	0	0	0	0	0	0	0	1	1
9	0	0	0	0	1	0	0	1	9

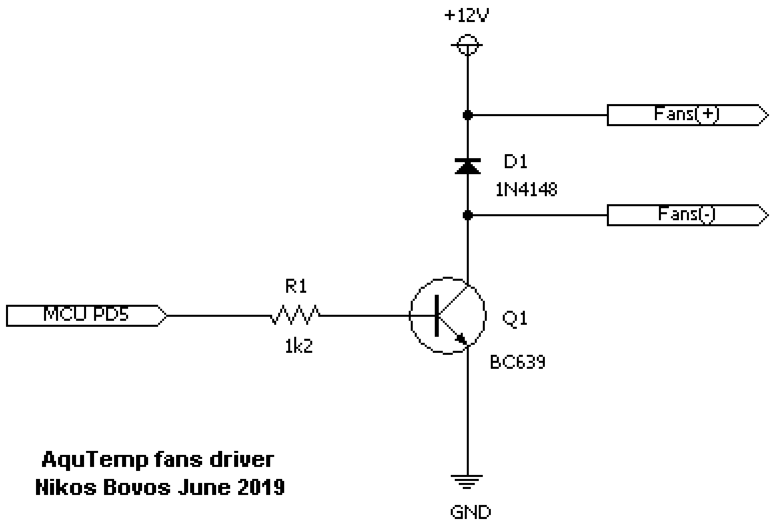
Table 1 – Non-dotted 7-Segment equivalent values

Digit	a	b	c	d	e	f	g	dot	MCU Value
or Character	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	(HEX)
0.	0	0	0	0	0	0	1	0	2
1.	1	0	0	1	1	1	1	0	9E
2.	0	0	1	0	0	1	0	0	24
3.	0	0	0	0	1	1	0	0	C
4.	1	0	0	1	1	0	0	0	98
5.	0	1	0	0	1	0	0	0	48
6.	0	1	0	0	0	0	0	0	40
7.	0	0	0	1	1	1	1	0	1E
8.	0	0	0	0	0	0	0	0	0
9.	0	0	0	0	1	0	0	0	8

Table 2 – Dotted 7-Segment equivalent values

Although ATtiny2313 (or other MCU) have enough power for led driving, the connection to the MCU ports for 'a' to 'g' and 'dot' segments, have current limiting resistors of 150 Ω (since 5V is the MCU output and 2V – 2,5V the dropout voltage of segments, for about 20mA forward current (for all displays segments ON – value for DC operation), we need a resistor of $R_{SEG} = (V_{MCU} – V_{DROP})/I_F$ ~= 150 Ω). Since successive driving is used, this value can be lowered (the higher the frequency of driving the lower the intensity of the display since the duty cycle decreases).

The fans-driver circuit is also a simple BJT switch, driving two 12V/0,14A DC fans.



The value of base resistor calculated in the previous pages (and it's a good practice to have it a little less than calculated to equalize the variations of dc current gain of the bipolar transistor, for the collector current). The anti-parallel diode protects from any back emf from the fans when they switch off.

The operation follows positive logic, when PD5=High then the two fans will work and when PD5=Low the two fans will stop. Maximum current is 280mA, so a slightly “bigger” transistor selected to operate without any problems (maximum collector power dissipation for BC639 is 1W and so for 280mA in worst-case scenario of $V_{CE(SAT)}$ = 0,5V total power dissipation will be about 150mW which is more than fine (even BC547 could be used instead, but I was afraid that in long term – and if the case is not well ventilated – and if the fans are operated for a long time during the day (this is held true here in Greece for the high temperatures in summer), it might have problems dissipating the produced heat to free air around it and thus decreasing collector current (or worse burn!) – it might also not, I tend to place slightly bigger components to be always without worries and frankly all these years no project ever needed service.

The diode used (1N4148) is an “all-around” good diode for low power applications and have been used in numerous circuits without any problems so far.

All other schematics not shown, can be easily derived from the port-map in Page 3...

The 1-Wire© Temperature sensor used in the project is Maxim's DS18B20 (placed in a water-proof container) and it shown in the picture (Yellow = 1-Wire Data Pin, Red=+5V and Black = Common ground):

This sensor has the ability to report the measured temperature in Celsius degrees with 9 up to 12 bits resolution. The selected resolution (programmable through the configuration register of DS18B20) and maximum conversion times are:

a) For 9-bits resolution maximum conversion time is 93,75 ms
b) For 10-bits resolution maximum conversion time is 187,5 ms
c) For 11-bits resolution maximum conversion time is 375 ms
d) For 12-bits resolution maximum conversion time is 750 ms

Since aquarium's water temperature variations are not extreme and the margin of High and Low settings is never that small, the 9-bit resolution was used (by programming the configuration register of DS18B20 - more on this later). Why using DS18B20 and not the classic DS1820 with 9 bits resolution ? The answer is because at the time or project construction, I couldn't find DS1820 but only DS18B20 in waterproof container (or I didn't search enough...)

Because only one 1-Wire device is used, the steps of requesting and identifying devices in the 1-Wire bus omitted (Search ROM and Match ROM commands). Before each use of sensor, the 1-Wire interface has to be initialized with the Reset procedure described in the data sheet of the devices and application notes from the manufacturer. The 1-Wire Reset Sequence used in the project is shown below :

a) Drive bus low

b) Delay at least 480 μs

c) Release bus

d) DS18B20 will delay 15-60μs to give a presence pulse (low 60us to 240us)

e) Sample bus (pulled low from device, or left high if device not found)

f) Delay at least 480 μs to complete the Reset cycle

Also, as the 1-Wire protocol instructs, a pull-up resistor of 4k7Ω is placed on the Data pin of DS18B20 (in this project all 3 pins are used, no parasite power to sensor). During various projects with these sensors, I have found however that many times a stronger pull-up resistor is needed (i.e 2k2Ω) than the recommended 4k7Ω because most of the times the total capacitance is larger than in 1-Wire specifications (25 pF) at least to my boards - highly depended on the PCB traces capacitance, cables etc). You can check your construction, noise, PCB traces, cable lengths etc to decide... To have “piece-of-mind” a 2k2Ω resistor placed as pull-up on the D_Q line of the temperature sensor

1-Wire bus designed by Dallas Semiconductor (now acquired by Maxim Integrated)

Although the above procedure is more or less straight-forward, the code to Reset the 1-Wire bus and discover the device is shown below and it is implemented as subroutine:

```
_1WireReset:
     sbi      _1WDDR, _1WBit     ; Set 1Wire Port Bit as Output
     cbi      _1WPort ,_1WBit    ; Clear Port Register (drive bus Low)
     rcall    delay480us         ; Delay 480us
     cbi      _1WDDR, _1WBit     ; Make Port as Input
     cbi      _1WPort, _1WBit    ; Tri-state (High-Z), no internal pull-up
     rcall    Delay70us          ; Delay 70us (15-60us = delay for presence)
     ;
     ; Sample Bus - If A Device Is Available (=Low) or no device (=High)
     ;
     sbic     _1WPIN, _1WBit     ; Skip Next If bit _1WBit in _1WPin is cleared
     rjmp     _1WireNotFound     ; Else Pin still High, jump To NOT Found
     clr      genio              ; Load 0 to genio (device found)
     rjmp     _1WireResetDelay   ; Jump below To Final Protocol Reset Delay

_1WireNotFound:
     ldi      genio, $01         ; Load genio with 1 (Device not found)

_1WireResetDelay:
     rcall    Delay480us         ; Delay 480 microseconds to complete protocol
     ret                         ; Return From Subroutine (came with rcall)
```

The declarations of the above constants used, are according to the Port-Map (the declaration of register genio used is shown below also). Of course (you already know that) all declarations are placed before anything is used (in the very beginning of the program):

```
.equ     genio         = r19    ; General I/O register

.equ     _1WPort       = PORTD  ; 1-Wire device will be connected to PORTD
.equ     _1WBit        = PORTD6 ; Specifically to PORT-D6 bit
.equ     _1WDDR        = DDRD   ; Data Direction Register for PORTD
.equ     _1WPIN        = PIND   ; Register to read value from PORTD Pins
```

After the above procedure the value of register genio (0 or 1) is saved to SRAM to be used later throughout the project. This value is checked every time a new temperature measurement is needed (if for some reason the temperature sensor is not found or an error occur, then the two 7-Segment displays show - - in order the user to check what is wrong with sensor or anything else). The value on SRAM is updated every time a 1-Wire communication is taking place, so to have always the correct setting for device existence.

Since PORTD6 = PIND6 = 6 in the definitions file of ATtiny2313, there is no need to declare another constant above (the instruction sbic _1WPin, _1WBit is the same as sbic _1WPin, 6 or sbic _1WPin, _1WPin6, when _1WPin6 is declared as PIND6.

Communication with 1-Wire devices is done bit-by-bit with LSB first as instructed by the 1-Wire protocol. The procedures to transmit '1' or '0' to the bus is described there also. The only “problem” is the many different delays (slots) that has to be implemented for the entire 1-Wire communication scheme. This is one of the reasons for choosing internal 4 MHz clock frequency for the system clock, instead of the more “easier” and by-default clock that any ATtiny2313 is shipped with (1 MHz). The two subroutines shown below are the Write_0_Bit and Write_1_Bit that used in the project:

```
;*****************************************************************************
;* Subroutine     : Write 0 Bit Subroutine
;* Name           : Write_0_Bit
;* Purpose        : Write a “zero” to 1-Wire bus
;* Use Registers  : None
;* Implementation : (1) Drive Bus Low
;*                : (2) Delay 70us (6us + 64us)
;*                : (3) Release Bus
;*                : (4) Delay 9us
;*****************************************************************************
Write_0_Bit:
     sbi      _1WDDR, _1WBit             ; Set PD6 as Output
     cbi      _1WPort, _1WBit            ; Out Value = 0
     rcall    delay70us                  ; Delay 70 microseconds
     cbi      _1WDDR, _1WBit             ; Make pin = Input
     cbi      _1WPort, _1WBit            ; No pull-up (PORTx=0), release bus
     rcall    delay9us                   ; Delay 9 microseconds
     ret                                 ; Return from subroutine
```

```
;*****************************************************************************
;* Subroutine     : Write 1 Bit Subroutine
;* Name           : Write_1_Bit
;* Purpose        : Write a “one” to 1-Wire device
;* Use Registers  : None
;* Implementation : (1) Drive Bus Low
;*                : (2) Delay 6us
;*                : (3) Release Bus
;*                : (4) Delay 64us
;*****************************************************************************
Write_1_Bit:
     sbi      _1WDDR, _1WBit             ; Set PD6 as Output
     cbi      _1WPort, _1WBit            ; Out Value = 0
     rcall    delay6us                   ; Delay 6 microseconds
     cbi      _1WDDR, _1WBit             ; Make pin = Input
     cbi      _1WPort, _1WBit            ; No pull-up (PORTx=0), release bus
     rcall    delay64us                  ; Delay 64 microseconds
     ret                                 ; Return from subroutine
```

Please note! The above routines are the actual used, but the entire 1-Wire implementation is not shown. There are also routines to send a byte bit-by-bit (that call the above routines), routines to read from device (again bit-by-bit with LSB first) etc. The delays needed by the protocol are found on the protocol by Maxim Integrated (and earlier by Dallas Semiconductor). Please refer to DS18B20 datasheet for further information.

As mentioned, 9-bits resolution was selected. This is done by programming the configuration register of DS18B20 before instructing a temperature conversion (the 1-Wire command for temperature conversion is $44). The scratchpad in SRAM of DS18B20 has 9 bytes, which are slightly different (the lower bytes) than the DS1820 and similar devices and shown in the following table (the table was taken from DS18B20 datasheet Rev.042208):

Byte 0	Temperature LSB ($50 on power-up)
Byte 1	Temperature MSB ($05 on power-up)
Byte 2	T_H Register of User Byte 1
Byte 3	T_L Register of User Byte 2
Byte 4	Configuration Register
Byte 5	Reserved ($FF)
Byte 6	Reserved
Byte 7	Reserved ($10)
Byte 8	CRC

Scratchpad in SRAM of DS18B20

Default temperature value of DS18B20 on power-up is +85^{O}C. To configure DS18B20 for the desired resolution, the Write Scratchpad command is used ($4E). <u>Three bytes must be written</u> to DS18B20 after this command, first is written to Byte 2, second to Byte 3 and third to Byte 4 of scratchpad. <u>All three bytes must be send even though the alarm function is not used</u> (Bytes 2 and 3 are the Alarm bytes (High and Low). In the implementation of the project, the three bytes written are: $00 , $00 , $1F. Since alarm function is not used, high and low limits were written for 0 degrees and the value written to Configuration Register ($1F) is according to the following table:

0	R1	R0	1	1	1	1	1

Configuration Register (Byte 4) of DS18B20

Where R1 and R0 are the only two configurable bits to select the desired resolution as shown below(table from same datasheet also), other bits must have the values shown:

R1	R0	RESOLUTION (BITS)	MAX CONVERSION TIME	
0	0	9	93.75ms	tconv/8
0	1	10	187.5ms	tconv/4
1	0	11	375ms	tconv/2
1	1	12	750ms	tconv

The Copy Scratchpad command is NOT used in the project (this one copies the contents of above 3 bytes to EEPROM) to avoid excessive DS18B20 EEPROM writing. As long as the DS18B20 is operating these settings are active in SRAM and the MCU is re-sending on every Reset, so always the sensor has the correct resolution selected without writing to EEPROM of DS18B20 any time.

The DS18B20 has two bytes to report the temperature (Byte 0 and Byte 1 in scratchpad). When it is configured for 12-bits resolution then all 16 bits of the two bytes contain valid data. For 11-bits the bit0 is undefined, for 10-bits, bit1 and bit0 are undefined and for 9-bits resolution bit2, bit1 and bit0 are undefined. The full 16 bits of the two DS18B20 temperature registers is presented below (S is the sign bit and it is S=0 for positive temperatures and S=1 for negative temperatures):

S	S	S	S	S	2^6	2^5	2^4		2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}

Temperature MSB (Byte 1 of Scratchpad) **Temperature LSB (Byte 0 if Scratchpad)**

Since DS18B20 is configured for 9-bits resolution, bit 2, bit 1 and bit 0 of the Least Significant Byte are undefined. So there are the first 5 usable bits in LS Byte and 3 last in the MSB, since temperatures to measure are always positive (negative temperatures for the aquarium are never to occur), always S=0 and will be ignored all the times...

It is easily seen that since last 3 bits are useless in LSB and last 3 bits are useful in MSB, the only thing we need to do is to shift-out the 3 bits of the LSB and shift-in the 3 bits of the MSB. **ROR** instruction exist (Rotate right through carry), which shifts-in the value of Carry flag to bit 7 of the register and shifts-out bit 0 of the same register to Carry flag in one clock cycle. The following subroutine is the one used to convert the two bytes to the final temperature value (1 byte). Decs and Unit are the MSB and LSB for bytes T_H and T_L respectively, declared as r17 and r18. Starting the rotation with the MSB we accomplish the task in few clock cycles (actually in 15 clock cycles, or 3,75μs including call and return). AquTemp declared as r16:

```
ConvertTemp:
     ror       Decs            ; Rotate Right Through Carry Decs (C=b0 of Decs)
     ror       Unit            ; b7=C=b0 of Decs, bit0 of Unit to Carry
     ror       Decs            ; ROR (C=b1 / Discard previous carry - was bit0 of Unit)
     ror       Unit            ; b7=b1 of Decs, b6=b0 of Decs, move bit 0 in C
     ror       Decs            ; ROR (C=b2 / Discard previous carry (was bit1)
     ror       Unit            ; b7=b2, b6=b1, b5=b0 of Decs, bit0 of Unit in C
     lsr       Unit            ; Unit = Unit / 2
     mov       AquTemp, Unit   ; Aqutemp <- Unit (copy final result to AquTemp register)
     ret                       ; Return to caller (came with rcall)
```

AquTemp register has the temperature value that will be converted for 7-Segment display equivalents and displayed to the two 7-Segment displays. Converting from binary to BCD is performed by the “standard” 8-bit Binary to BCD routine as described by Atmel in Application Note 204. The last lsr Unit is used to discard the 0.5 degrees bit and produce the final value. Also AquTemp register is converted to ASCII string when requested by the PC-Software (more on this later).

Reading of the scratchpad register of DS18B20 can be terminated any time by issuing a 1-Wire Reset. For simplicity reasons and not to have too many readings to do that are out of interest, 1-Wire Reset is performed after reading the first two bytes (T_L and T_H registers). CRC is not used and not re-calculated here.

A quick example to explain the above process:

- For +25ºC the two scratchpad bytes (byte 0 – T_L and byte 1 - T_H) are:
  T_L = 10010001 and T_H = 00000001 and the last 3 bits of T_L will be discarded since are invalid for the 9-bits resolution.

- By shifting out 3 bits and shifting in 3 bits (the three pairs of **ror** above) the final 8-bit value will be Unit = 00110010 in binary (or 50 in decimal)

- The last **lsr Unit** will divide the value by two, thus producing 25 which is the actual temperature value (in binary is 00011001).

- If 0,5 degrees is needed (and 3 7-Segment displays used), then the carry flag (last bit shifted) can be checked for ,0 or ,5 degrees display.

Similarly, for +10ºC the two bytes are T_L = 10100010_2 and T_H = 00000000_2 and so after three shifts the value of Unit = 00010100 in binary (or 20 in decimal) and the last **lsr** will divide that value by 2 producing the final AquTemp = 00001010_2 (10 decimal).

AquTemp register holds the temperature value (25 and 10 in the examples). This value is checked against the defined Low and High limits. The subroutine is called every time a new temperature measurement is taken – about every 10 seconds (I think again that is no real point to show this, but.....share the knowledge ???):

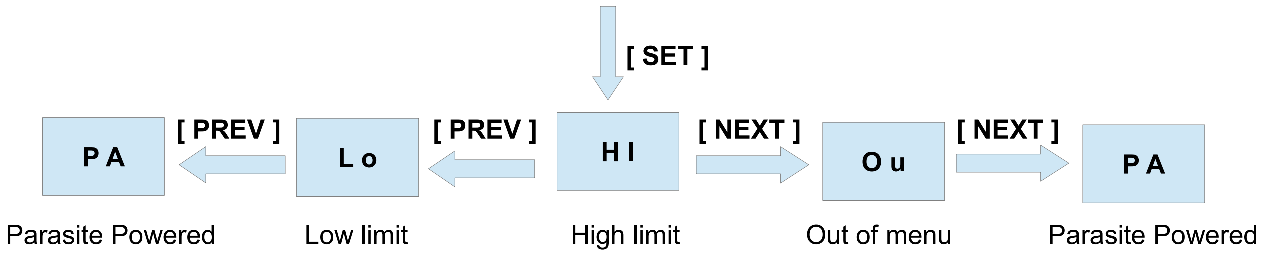
```
;************************************************************************
;* Subroutine   : CheckTempLimits
;* Purpose      : Check current water temperature (Aqutemp) with tempLOW and
;*              : tempHIGH and decide what to do with the fans (start / stop)
;*              : Since limits ARE included to valid temperatures, then
;*              : i will check for lower only on tempLOW and tempHIGH
;* Called by    : rcall , so will return with RET
;************************************************************************
CheckTempLimits:
     cp        tempHIGH, AquTemp       ; Compare tempHIGH with AquTemp
     brbs      0, Check_Enable         ; Branch if bit0=C in SREG=1 (tempHIGH < AquTemp)
     cp        AquTemp, tempLOW        ; Compare AquTemp with tempLOW
     brbs      0, Check_Disable        ; Branch if bit0=C in SREG=1 (AquTemp < tempLOW)
     rjmp      CheckTemp_Out           ; Else we are within limits, jump out...
Check_Enable:
     sbi       fansPORT, fansBIT       ; Set fans = ON
     rjmp      CheckTemp_Out           ; And leave this subroutine
Check_Disable:
     cbi       fansPORT, fansBIT       ; Set fans = OFF
CheckTemp_Out:
     ret                               ; Return from subroutine (came with rcall)
```

AquTemp (r16) has the current temperature, tempLOW (r7) and tempHIGH (r8) are low and high desired temperature limits (retrieved from EEPROM on startup, saved to EEPROM by the menu or by PC command). fansPORT = PORTD and fansBIT=PORTD5 as shown in the port-map on Page 3.

BRLO instruction could be used instead, since both are equivalent (BRLO Check_XXX is the same as BRBS 0, Check_XXX the usage is a matter of taste and style).

The Menu System of AquTemp project

Three push-buttons are connected to PORTD in the bits shown on the port-map. The menu implementation is shown below and it is started by pushing the SET button. Since we have 2 displays, all menus are truncated to fit these - the operation is “cyclical” using the buttons (HI-->[Next]-->Ou-->[Next]-->[PA]-->[NEXT]-->[HI] etc etc etc):



By selecting Lo and Hi settings with SET buttons the Low and High temperature limits can be set using PREV and NEXT buttons. Supported temperatures for the project are from 10 to 40 degrees Celsius (50 to 104 Fahrenheit) and probably they are a little “stretched” for this kind of project... Temperature check for validity of low and high settings is NOT performed (obviously Low must be lower than High limit, if they are selected different the fans will work all the time without ever closing (or not start at all). It's of course user's responsibility to set the desired limits correct...As always !).

Selected High and Low values are stored to EEPROM of the MCU for future reference and loaded to dedicated registers after RESET. If it is not desired to be saved in EEPROM while in menu, then RESET must be performed, there is No Exit button available. It could be implemented in the lowest or highest temperature (before 10 or after 40 as a selection, for instance 'Ou') but really I didn't bother to do that (EEPROM has an endurance of 100.000 Write/Erase cycles, so a few more saves doesn't mind that much...)

The above low and high limits if you write your own software, can be extended to any value you need and so use this project for other purpose as well (these “extended” settings must be send by the PC-software, the firmware does not support other values than the ones shown, but since there is no check for validity, any value can be set (at least for positive temperatures, since the Sign flag is not checked within the firmware, it is always “assumed” to be S=0). The two limits are included in the checking procedure (we check for lower). This means that if high limit is set to 25^{o}C then the fans will start at 26^{o}C. Similar when low is set on 22^{o}C fans will stop at 21^{o}C.

PA menu is used to set if the DS18B20 sensor is using Parasite Power (set to 1) or if V_{DD} is connected to +5V (set to 0 – the default value) so for any type of sensor connected to be used. This will change the method of requesting the end-of-temperature conversion as instructed by DS18B20 datasheet, providing the necessary delay and power (in Parasite powered - DS18B20PAR you can't acquire the state of conversion. Powered DS18B20, report Low while converting temperature and High when done).

Communication and rest of functions implemented

Watchdog timer is used in this project also, configured for Reset after expiration (to help if runaway code is executed or any other occasion the program stuck somewhere). The Watchdog timer in ATtiny2313 is clocked from an internal 128 kHz oscillator independent of the clock selected and it is configured for 4 seconds time-out period, (or 512 kCycles). The procedure to enable and select the desired prescaler is described in ATtiny2313 datasheet. The Watchdog timer is reset inside the program in various points to avoid it's expiration and thus Reset the MCU.

Communication of MCU to the AquTemp Software is done at 19200bps with 8-data bits, 1-stop bit and Even-parity. The USART Rx Complete interrupt is used, whenever a command is received from the AquTemp Software. The baud rate selected is the most reliable (and higher) for the selected clock. The table of supported commands for firmware 1.00 is shown in the following table:

Command	Description of command
!TH=xx	Set High Temperature Limit to xx (Send by PC, MCU Responds !OK)
!TL=xx	Set Low Temperature Limit to xx (Send by PC, MCU Responds !OK)
!TEM	Request current temperature (Sent by PC, MCU responds !T=xx)

Every command starts with '!' ASCII character and ends with Carriage Return character ([CR] = $0D / ENTER). Any other command or character received other that the ones shown above will be ignored by the MCU (it will not report anything or respond with my usual '!ERR' that I use in other projects when communicating with PC software – for instance in A8Eprog project). In the moments these lines are written I haven't decide yet...

The **!TEM** command is used for the logging function of the PC-Software, more in the next pages in the software presentation...

On USART Rx Complete interrupt, the MCU stores the byte received from PC to a dedicated area (buffer) in SRAM, until [CR] received or the buffer length exceeded. In this case, received byte is overwrite the last byte, to avoid excessive SRAM usage and eventually STACK corruption (STACK is located on the top of the available SRAM – which is 128 Bytes in ATtiny2313). Maximum length of command is 6 bytes plus one of the [CR] byte, so there is always enough space for Stack usage and no corruption might ever occur under normal program execution. Interrupt Service Routine also checks for the three supported errors that the MCU USART Module support: Frame Error, Parity Error, Data Overrun and discards the received byte in case of any of the above errors occur. For communicating to PC a DB9-USB-D5-F module from Future Technology Devices International Ltd was available and used and the appropriate drivers were installed to PC, alternatevely the "all-time-classic" MAX232* circuit can be used (and a little more PCB space).

MAX232 is a dual receiver/transmitter by Maxim Integrated to convert RS232 signals to TTL levels

All the above mentioned operations, built in 1848 words (approximately 90% of MCU flash memory). Also an eeprom file containing 23 bytes to be programmed produced also. This is more than good for all these stuff...Using a high-level language code probably would not produce code that fit to MCU memory...Ah the power of Assembly !

In further tests and because Parasite Power sensors (DS18B20PAR) were not available, the code “refined” to exclude this option. This compacted the code to 1648 words (approximately 80% of MCU flash memory). This code is now used in my aquarium, but for “fairness” the full code .hex and .eep files are provided because I don't have parasite power sensors, one other might have and since I started with this in mind the files that accompany this document are the “full” ones. One note, I wasn't able to test the parasite powered sensor operation, since I don't own one...Probably will work as the guidelines from the datasheet were followed. If not, please contact me to correct it..

As you may see from my writing-style, every project have more remarks than instructions! That's why when constructing something that time is not an issue I always use assembly (works better with my brain) and for projects with deadlines or fast-delivery, I use BASCOM-AVR which is perfect for quick results (as with all Basic languages of course !). It's true that Assembly need more time to write the firmware than any high level language, but I still use it whenever time is not an issue because I feel more confident with it.

Because nowadays disassembling .hex files is easier than ever before, whatever you do (I recommend you don't) is at your own risk and blame. Please do not send me disassembled code that you don't understand and request me to explain it...

Timed temperature measurement

A temperature measurement is requested approximately every 10 seconds from the sensor (with Convert T command = $44). To accomplish the task the 16-bit Timer of ATtiny2313 (Timer-1) used, configured for overflow interrupt every 1 second. Every one second the LSD's dot is altered to show that the project is up and running. After 10 overflow interrupts (10 seconds approximately), a temperature measurement is requested from DS18B20 and the measured temperature is updated on displays. After measurement, the two limits are checked (and fans will start / stop or stay as is) and the execution continues...

Since main clock = 4MHz (250ns period), Timer-1 is configured for prescaler value= 1024 and so Timer-1 frequency is 4 MHz / 1024 = 3906,25 Hz (and the Timer-1 period is 256 microseconds). To accomplish an overflow every 1 second, Timer-1 will need to count: Tcnt = 1sec / 256us = 3906,25 times. Let's say 3906 for simplicity...

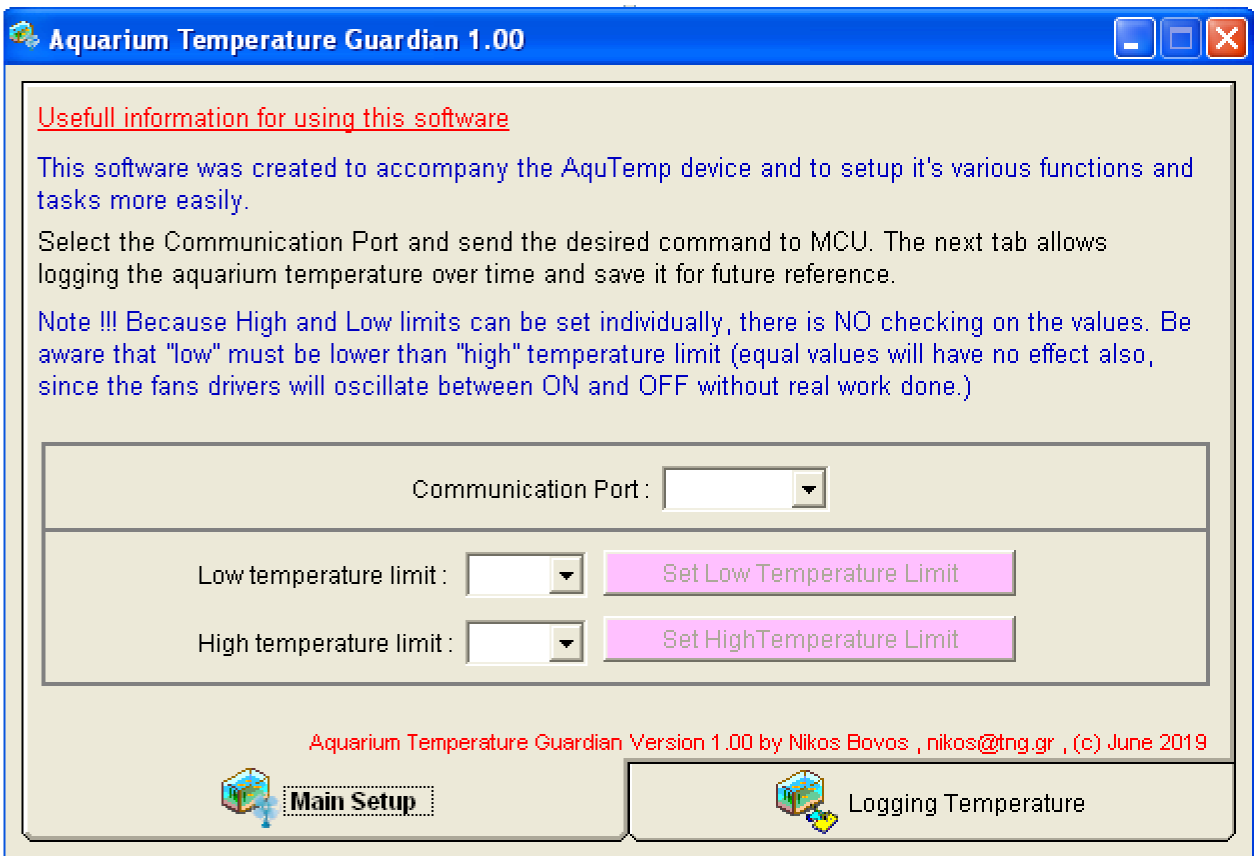
Timer-1 is a 16-bits Timer, the overflow will occur after increasing from 65535 (maximum 16-bit value) overflowing to 0. This means: 65535-3906 = 61629 must be the startup value of Timer-1, to overflow every 1 second. Inside the interrupt service routine this value will be loaded to Timer-1 for the next overflow interrupt to occur. The following code shows the initialization of Timer-1. Registers “genio” and “genioSEC” are high registers (r16-r31) that support “ldi” command. During the initialization process of peripherals used, all interrupts are disabled (default value for I bit in SREG on power-up) and enabled just before the main program starts. Another prescaler value could also be used, but since we accomplish the task, it's not necessary to investigate other prescaler values...

The following code is used to the initialization routine of the project to setup Timer-1 for the operations described:

```
ldi       genio, (1<<CS12) | (1<<CS10)  ; Shift 1 to CS12, CS10 bit positions
out       TCCR1B, genio                 ; Set Timer-1 prescaler value = 1024
ldi       genio, (1<<TOIE1)             ; Shift 1 to TOIE1 bit position (bit 7)
out       TIMSK, genio                  ; Enable Timer-1 Overflow Interrupt
ldi       genio, high(61629)            ; Load High byte of value 61629 to genio
ldi       genioSEC, low(61629)          ; Load Low byte of value 61629 to genioSEC
out       TCNT1H, genio                 ; Set High Byte of TIMER-1 Counter FIRST
out       TCNT1L, genioSEC              ; Set Low Byte of TIMER-1 Counter after.
```

Aquarium Temperature Guardian Software

This software runs in Microsoft Windows® Operating Systems (it's a 32-bit Visual Basic application, that can be executed in both 32-bit and 64-bit Windows Systems) and can be used to log the aquarium's temperature and as well as to configure the MCU for high and low temperature limits. The main program screen:


Aquarium Temperature Guardian 1.00

Usefull information for using this software

This software was created to accompany the AquTemp device and to setup it's various functions and tasks more easily.

Select the Communication Port and send the desired command to MCU. The next tab allows logging the aquarium temperature over time and save it for future reference.

Note !!! Because High and Low limits can be set individually, there is NO checking on the values. Be aware that "low" must be lower than "high" temperature limit (equal values will have no effect also, since the fans drivers will oscillate between ON and OFF without real work done.)

Aquarium Temperature Guardian Version 1.00 by Nikos Bovos , nikos@tng.gr , (c) June 2019

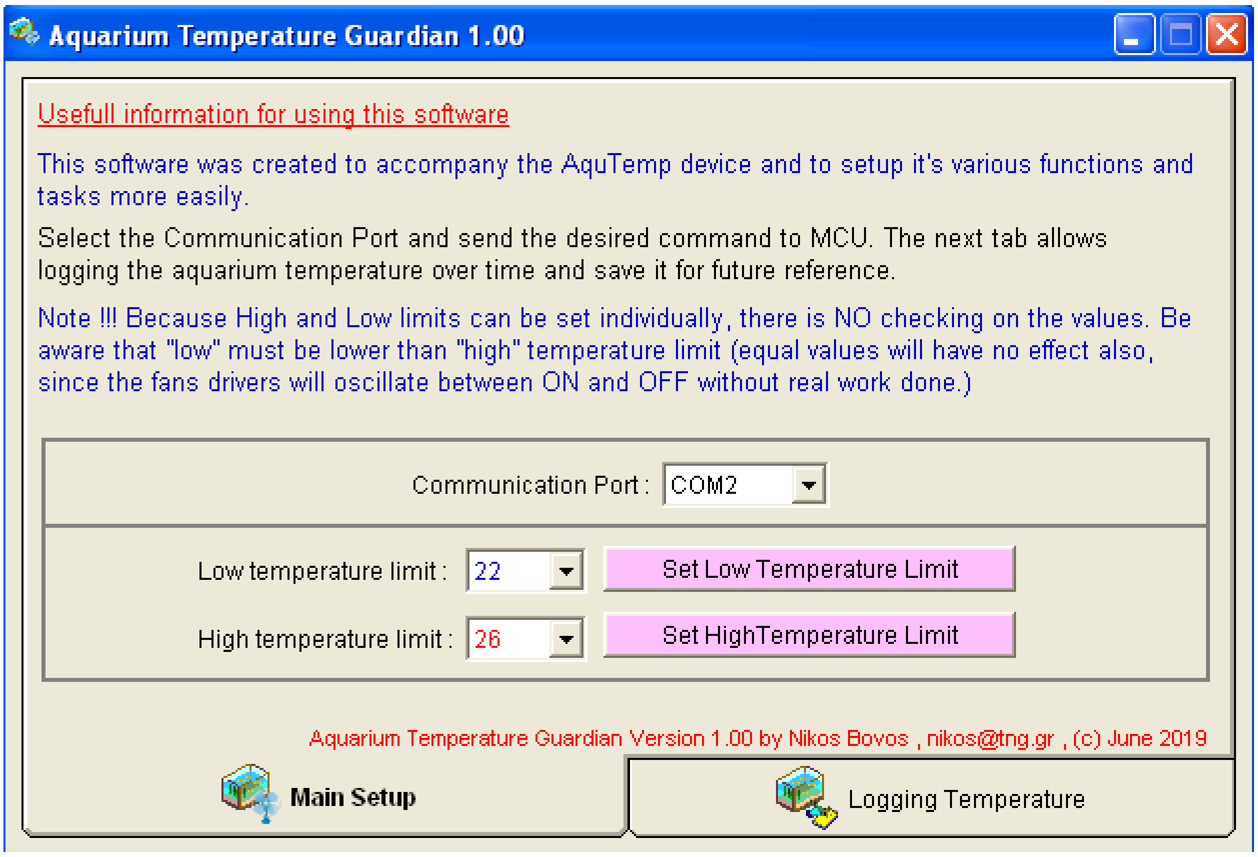
Select communication port and the desired low and high limits. The two command buttons send the "set temperature limit" (high and low) to MCU. Because each command is send to MCU individually by it's dedicated command button, the "limits checks" must be done by the user. If you send low limit higher than high, then the fans on the aquarium will work continuously. If you send equal values then the fans drivers will oscillate between ON and OFF state (and the fans will not even start...)

Supported temperature range is 10 – 40 degrees Celsius (probably for more than aquarium usage) and can be sent to MCU as long as the project is running and the MCU is connected to PC (there are no special requirements or special menu settings).

Setup package contain all the necessary Microsoft Visual Basic©* libraries that will be installed to the target machine if not already installed (usually it's highly unlikely that these libraries do not exist in a Windows machine). If prompted for older version versus the one installed, always keep the newer version in your computer.

Microsoft Visual Basic is a trademark of Microsoft corporation

The main settings that applied for my aquarium:



Aquarium Temperature Guardian 1.00

Usefull information for using this software

This software was created to accompany the AquTemp device and to setup it's various functions and tasks more easily.

Select the Communication Port and send the desired command to MCU. The next tab allows logging the aquarium temperature over time and save it for future reference.

Note !!! Because High and Low limits can be set individually, there is NO checking on the values. Be aware that "low" must be lower than "high" temperature limit (equal values will have no effect also, since the fans drivers will oscillate between ON and OFF without real work done.)

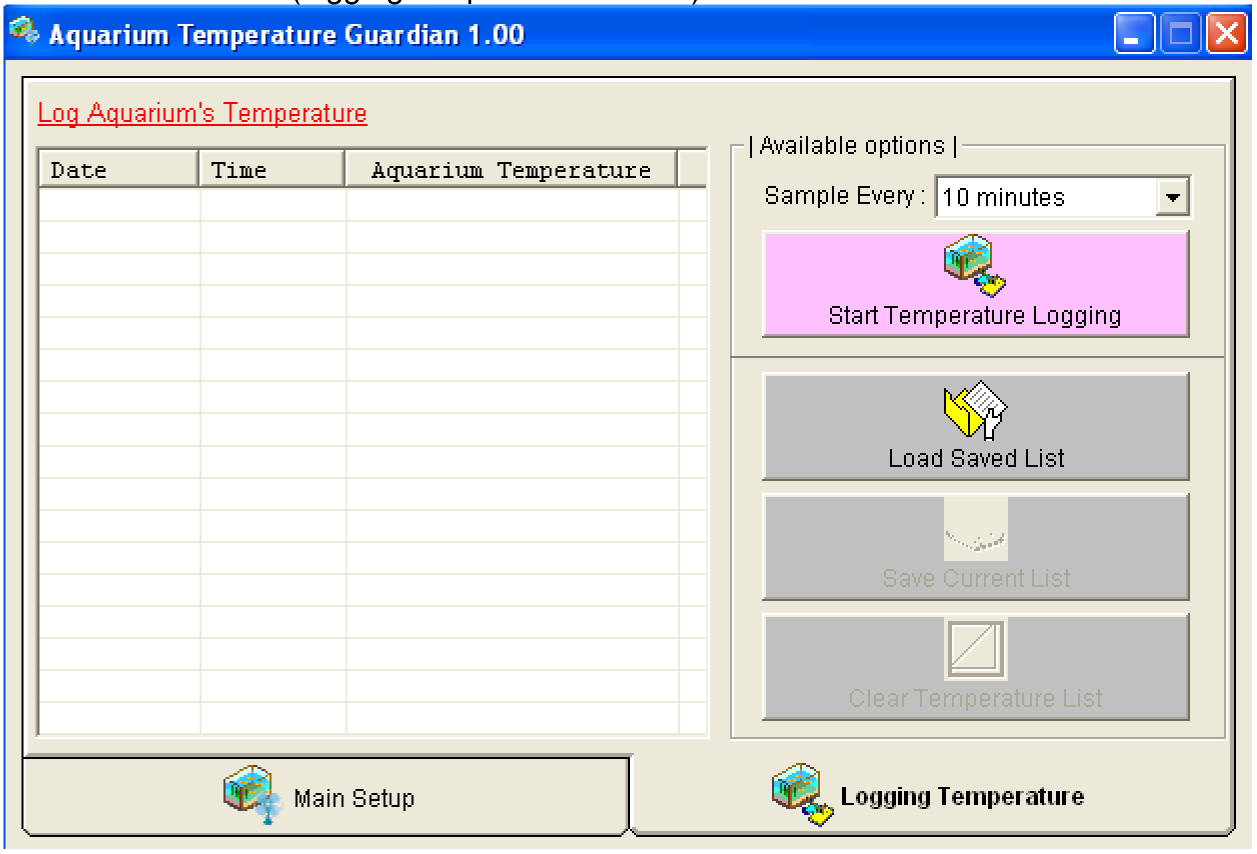
Communication Port : COM2

Low temperature limit : 22 — Set Low Temperature Limit

High temperature limit : 26 — Set HighTemperature Limit

Aquarium Temperature Guardian Version 1.00 by Nikos Bovos , nikos@tng.gr , (c) June 2019

The second tab (logging temperature screen):



Aquarium Temperature Guardian 1.00

Log Aquarium's Temperature

Date	Time	Aquarium Temperature

| Available options |

Sample Every : 10 minutes

Again, simple interface and functions to request temperature, load saved file and save the current list that fills in every sample from the mcu, along with the sample's date and time (uses system-configured format date and time format, for my setup date is dd/mm/yyyy and time is hh:mm:ss). Files are ANSI text files with 2 lines of header and then 3 fields separated by Greek comma (,). A simple view of a log file is shown below (temperature request every 30 seconds):

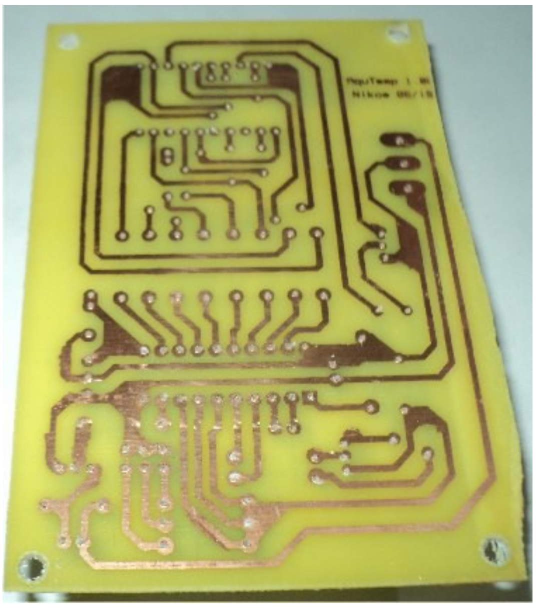
```
 AquTemp 1.xx Log File
-----------------------
08/06/2019,13:05:35,23
08/06/2019,13:06:05,23
08/06/2019,13:06:35,23
08/06/2019,13:07:05,23
08/06/2019,13:07:35,23
08/06/2019,13:08:05,23
```

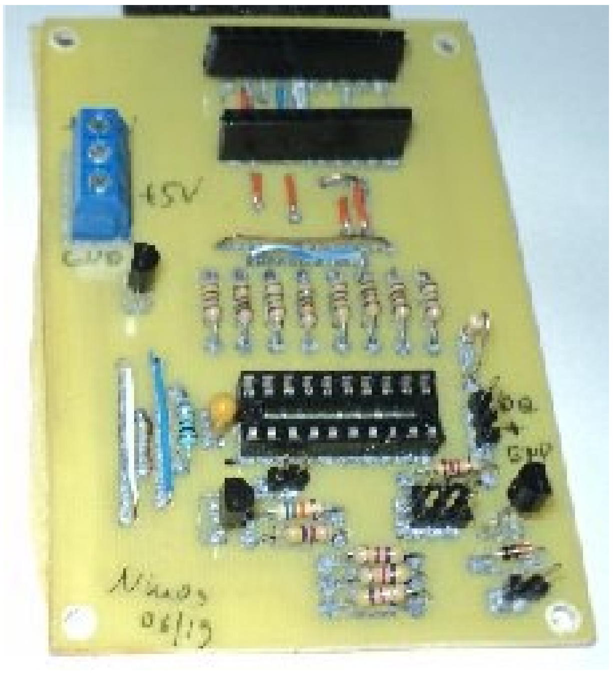
The 'sample every' combo box allows selecting the desired time period per sample and currently supports 30 seconds, 1,3,5,10,30 and 60 minutes.

If you write your own software you can add more features, this is a standard working software to configure the MCU and to log the current temperature in specified periods of time (supported commands were shown on previous pages and communication parameters also). Frankly the software rarely used - only during building of the project and the various tests/configuration, mainly constructed because there are many “computer-fish-freaks” out there, that would love to log the temperature of their aquarium (and even send the aquarium temperature every hour to their cell-phone...), so this project is for them as an idea to build their own (of course, the extra features must be written by whoever wants to in his/her own software implementation. Here supported commands and communication parameters were provided for usage...Don't ask me to do it, I will not !

Construction And Tests

The “Heavy Metal” part of each project is of course building of the hardware and start/test and work the project. For this one two PCBs used, one existed (the power supply from past project) and one created for AquTemp.



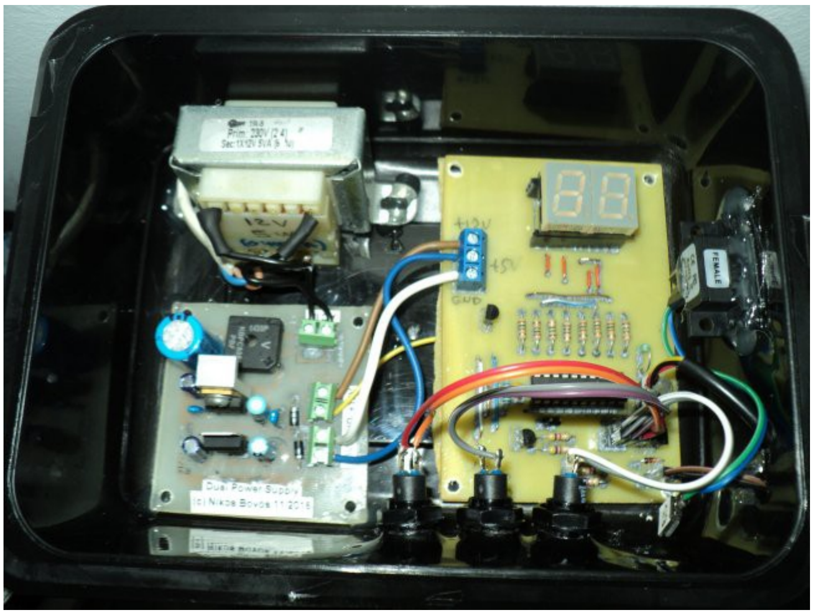


Copper side and components side of the AquTemp PCB. Dimensions of the PCB are: Width = 7cm, Height = 10cm (some paper left from the toner transfer method of the components side but it doesn't hurt that much...I'm still working on a good method to remove the paper without removing the toner from the top side...)

This is the dual power supply, borrowed from a previous project of mine. It conforms more or less to the schematic presented here (this one is missing the two extra protection diodes along the voltage regulators and have two extra capacitors for improved high frequency ripple rejection and was used in a multi coin selector project with ATtiny2313 also).

This one will provide the necessary +5V and +12V voltages to the main board (the heat-sink is placed on the 7812 regulator which is the one to drive the fans also (and have to dissipate the total +12V and +5V power). Dimensions are Width=5,5cm, Height=5,5cm

The total cost of the project is zero, since all parts existed and it was a good idea to use some to create something useful, but if I have to calculate for future reference the cost of components used, then (by the prices here always !!!) it would be about 20,00 € without the cost of a box. Metal boxes are fairly expensive here and I didn't want to buy one, so a plastic container used instead...

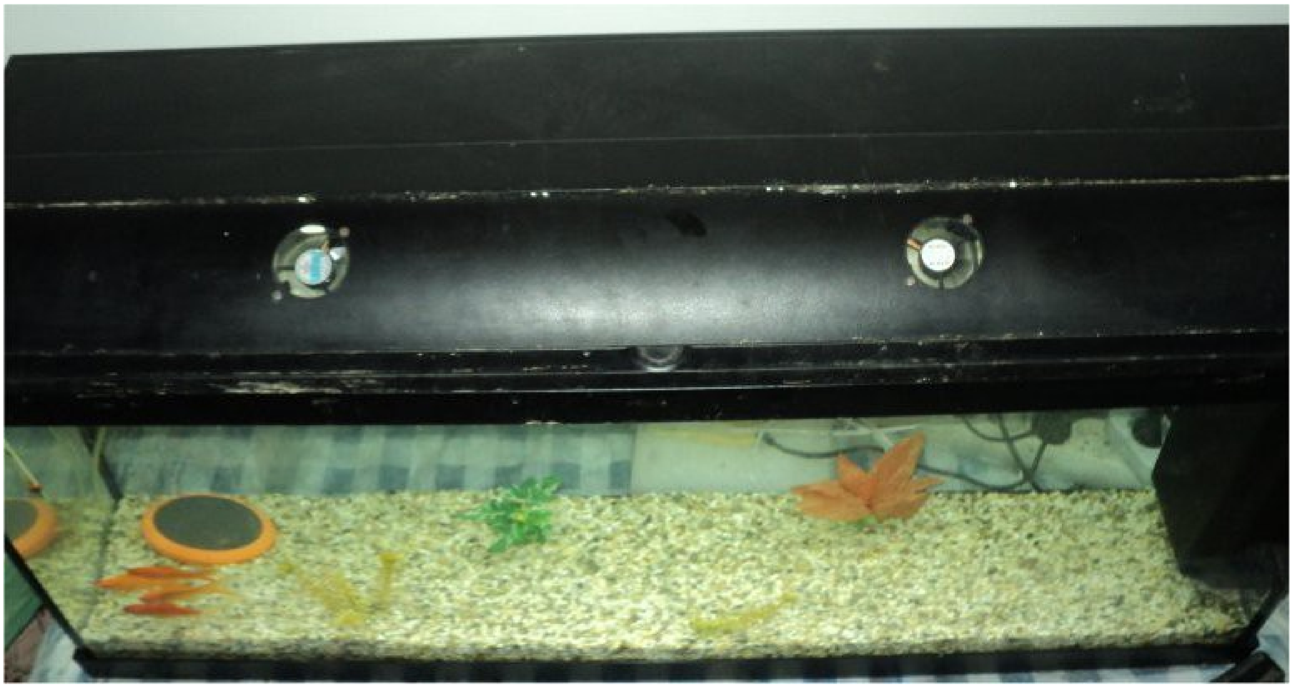


The final "box" of the project during installation

The "box" (plastic container) was covered on top with a transparent Plexiglas, glued in the four corners (to have isolation from the environment and most importantly to protect against accidental drop of water !).

The DS18B20 sensor was placed in the back (left) in the middle of the aquarium, on the opposite side of the heater (which is inside the black box in the right part) and behind the standard aquarium temperature meter shown in front (left). The two fans shown on the top cover are 50mm 12V/0,14A DC, mainly left from various PC-components – mainly older CPU fan modules...

Due to weather conditions here at the time of constructing, when the project started the water temperature was measured at 30°C and the fans started to work...


Top/Front view of the two fans and the spaghetti of cables on the back :-)

The two fans are sucking the hot air from inside and blow it outside. With this way the temperature is falling more slowly than with "drastic measures" (I.e placing ice cubes inside the aquarium !), but at least there isn't any shock to the fishes, because the temperature is changing slowly. After about 1 hour of work (room temperature was 33 degrees and water temperature was 30 degrees – without the use of air conditioning), one degree fall was measured. Time will show if the construction will hold and the temperature will be held well within limits (the use of air-conditioning or fans in the room always help to have less room temperature and thus less water temperature - and cool down quicker)...

Because this project is to work on a 24-hour basis (on summer of course), there are some factors that limit it's performance. For instance, lights are automatically turn-on at 5 in the afternoon and switch-off at midnight, so they add to existing water temperature, making the two fans work for more time but it's ok...I prefer lights on and fans to work more rather to have lights on in the morning that I am out for work ... Also as the fans work, they heat-up and add to the air-heat themselves, but you can't have it all...I can say that I'm satisfied with the operation of this small project so far...

Maybe the only thing that I don't like so much is the noise of the two working fans, but there is nothing more I can do - both fans were screwed on the top cover of the aquarium and anti-shock material (anti-static foam) used to lessen the produced noise – but if you are close to aquarium you can still hear them... Nobody is perfect... As I told to my fishes when the fans started : "bare the noise or you will boil in the hot water and make a good fish-soup...Ha !"

Thanks for reading and baring with all my bla-bla-blas...Take care, have fun, see ya!