MASARYKOVA UNIVERZITA
Česká republika

Javier Carrasco Reyes

Advisors: Ing. Zbyněk Bureš, Ph.D.
Prof. Ing. Václav Přenosil, CSc.

Department of Information Technologies

Masaryk University

LOCALIZATION OF ENTITIES AND CONFIGURATION OF AN ENVIRONMENT BY WIRELESS TECHNOLOGY

Contents

1. Introduction

The implemented system consists in controlling Jetsurf races. The track is delimited by buoys that are floating in the water, and consequently not in a fix position.

Every buoy has a PIC microcontroller connected a GPS to know its exact position, 2 IQRF modules to allow the communication with the rest of the system and a battery to supply electric power the electronic components. This system must have to take into account the low voltage as one the most important criteria, for this purpose the sent messages will be reduce in order to use the less power consumption possible.

As it is said, for the communication part every buoy has 2 IQRF modules: one is used to get the position of the drivers if there are close to the buoy and the other one works in the mesh. For the mesh, every buoy is considered a node and them all work together in order to send the packages to the coordinator. In this way, the coordinator can set and get information about all the buoys and about the riders.

The coordinator can send commands in broadcast or unicast mode. These commands can get information about the GPS (status and data), get information about the battery and switch on/off the LEDs.

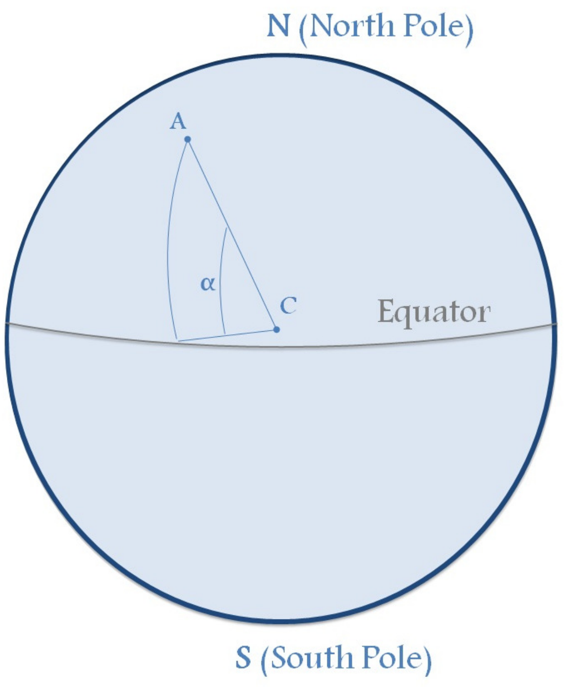
2. Geographic coordinate system

The coordinate system can place a point in every position over the Earth surface. It is composed by latitude and longitude.

2.1. Latitude

The latitude represents the angle between the Earth equator plane and the wanted position regarding the Earth's centre and which hemisphere (North or South). The latitude 0º corresponds to the equator, the latitude 90º N corresponds to the North Pole and 90º S corresponds to the South Pole.

The lines of constant latitude are called "parallels", for example, the Equator is the parallel with latitude 0º.



Latitude for a certain point

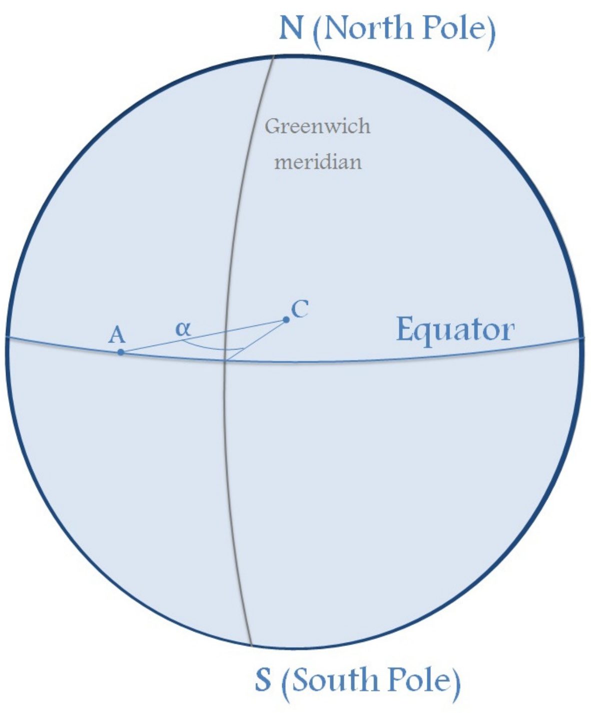
Where:

- A: Certain point in the Earth
- C: Earth's centre
- α: Latitude

2.2. Longitude

First of all, we must know what a meridian is: a line completely perpendicular to all the parallels from North Pole to South Pole. The meridian through Greenwich (England) is the prime meridian.

Longitude represents the angle between prime meridian and the wanted position regarding the Earth's centre and which part (East or West). The longitude 0º corresponds to the meridian pole and the latitude 180º corresponds with the back part of the prime meridian.



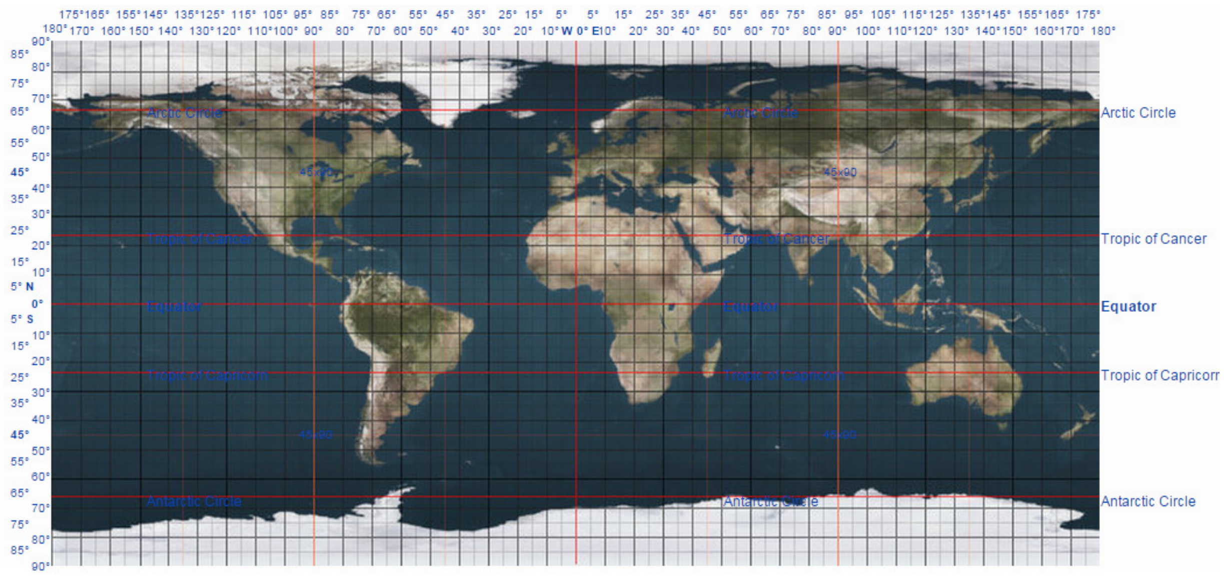
Latitude for a certain point

Where:

- A: Certain point in the Earth
- C: Earth's centre
- α: Longitude

2.3. Map with parallels and meridians

It is interesting to realize what latitude corresponds to the real world, because the kinds of race that this system would not be held for example in the poles because of the cold temperatures. We will see later how the latitude and does it affect to the system.



World map with latitudes and longitudes

[Source: Wikipedia[23]]

2.4. Coordinate representation

The coordinate representation use to be in the order: latitude, longitude. There are many notations to express it, but we will focus in the more common used that contains the one we will work with. Those notations are degree-minute-seconds and decimal degrees. We will use the coordinates of the city Brno (Czech Republic) as example.

2.4.1. Degrees minute seconds

The coordinates in degrees-minute-seconds system are expressed in this way:

49° 12′ 0″ N, 16° 37′ 0″ E

Like in the time system, every degree has 60 minutes, and every minute has 60 seconds.

To transform from the degrees-minutes-seconds to decimal degrees is that simple as:

$$decimal\ degrees = degrees + \frac{minutes}{60} + \frac{seconds}{3600}$$

2.4.2. Decimal degrees

The coordinates in degrees-minute-seconds system are expressed in this way:

49.2000º N, 16.6167º E

But, it could be represented like positive and negative coordinates:

- Latitude:
  - Positive → North
  - Negative→ South
- Longitude:
  - Positive → East
  - Negative→ West

Therefore, the example in this type of notation would be:

+49.2000º, +16.6167º

To transform from the decimal degrees to degrees-minutes-seconds is that simple as:

$$degrees = \lfloor decimal\ degrees \rfloor$$

$$minutes = \lfloor 60 \cdot (decimal\ degrees - degrees) \rfloor$$

$$seconds = \left\lfloor 3600 \cdot (decimal\ degrees - degrees - \frac{minutes}{60}) \right\rfloor$$

3. Requirements

Before starting to work, we need to know all the requirements of the system in order to made a system that satisfies what is actually wanted. For doing this we detail all the requirements ordered in functional, non-functional and technical.

3.1. Functional requirements

- Create a mesh
  - Bond node
  - Unbond node
- Send messages to the buoys
  - Unicast + Broadcast
  - Send orders
    - Switch on/off the LEDs and set its duty
    - Make sleep the nodes
    - Set GPS commands
  - Get information
    - Get LEDs status
    - Get GPS data
    - Get battery status
    - Get riders position

3.2. Non-functional requirements

- Documentation: The full specification is in this document.
- Efficiency (resource consumption): The buoys require batteries and the PCB have to take into account the power consumption.
- Modifiability: The system has several layers, keeping the specification protocol it is possible modify it.

- Extensibility: The system allows to add futures implementations, there are reserved commands.
- Reliability: The mesh network have repeaters which add redundant links in order to make the system reliable.
- Fault tolerance: The mesh network is reliable and allow the failure of some nodes.
- Network topology: The mesh topology has to be as tree. The reasons are the reliability and is detailed in the point 3.3.2.
- Response time: The system needs real-time data.
- Scalability: The system allows from 1 to 239 nodes, more than enough to cover a regular race.

3.3. Technical requirements

The environment will be hard because it will be the water. This point makes that obviously the system must be waterproof.

3.3.1. Waterproof

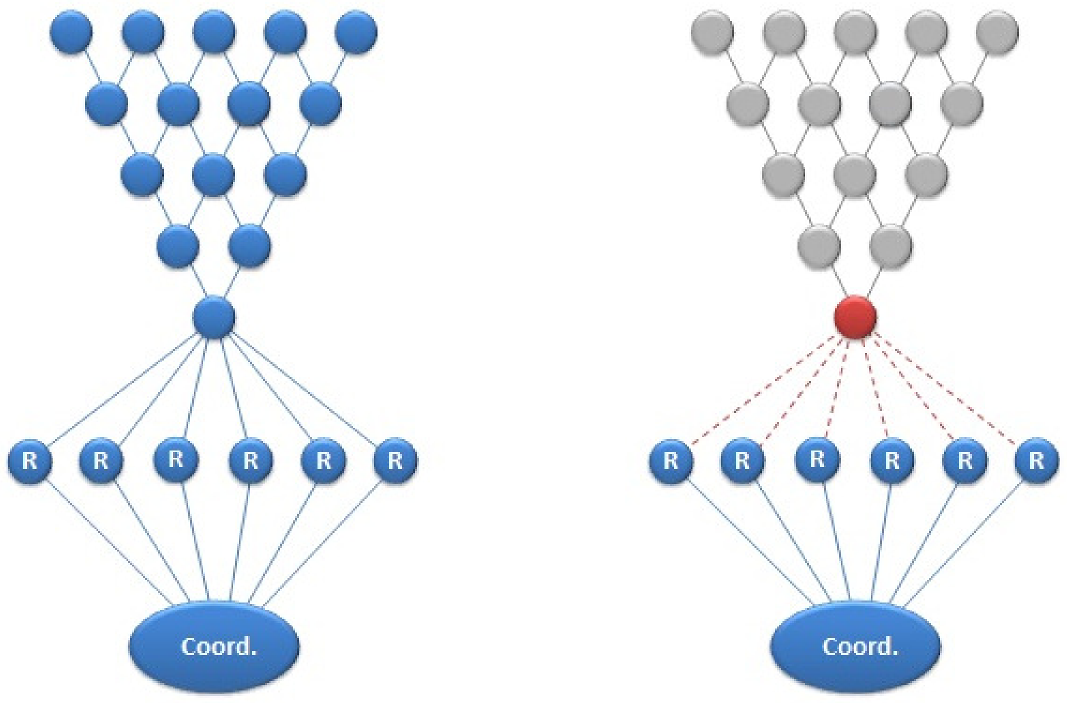
The hardware consists in electronic must be covered and closed to prevent that the water provoke a short-circuit.

3.3.2. Wireless

We cannot use cables in the water because, as it is said in the previous point, the system works in the water. For this reason it is better to work using wireless technology. There are 2 kinds of networks: centralized or distributed. As we want to cover 1 km^2, even in the water without walls disturbing the connection is too much distance and we would need big antennas and big batteries to supply them. Thus the best option is to implement a mesh where the nodes create a structure with redundancy which provide the system with reliability. All the nodes have to know how to redirect every packet to make it get its destination that can be another node or the coordinator.

There are 2 topologies to implement the network: triangular from vertex and triangular from base.

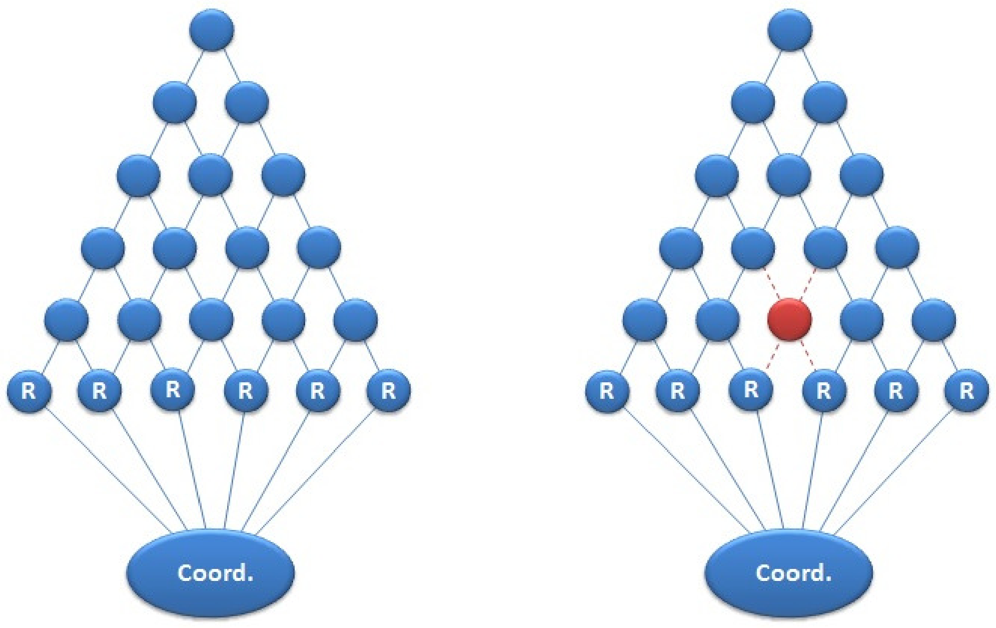
3.3.2.1. Triangular from vertex mesh topology



Triangular from vertex mesh topology

If the mesh has a topology like this and the critical node falls, then the full mesh would be disabled.

3.3.2.2. Triangular from base



Triangular from base mesh topology

In the other hand, if the mesh topology follows this criteria and one node falls, the rest of the mesh is not affected at all. The package which were sent by a route through the fallen node have other options to get their destination.

There are the same number of nodes but there are more path redundancy that make the system more reliable.

3.3.3. Buoy nodes self power supplied and energetically efficient

The fact of every node is connected to the other by wireless technology make that every must be self power supplied by its own battery. This batteries can't be very big because of its weight (it could make the buoy sink). This forces to make the nodes energetically efficient using techniques like making sleep what is not being used (a part, like the GPS, or the whole buoy). Another important point to make the system more energetically efficient is trying to send less amount of bytes needed. For this reason all the packages will work in binary instead of characters. There are some other techniques to reduce some message that will be explained later.

4. Material for learning and developing

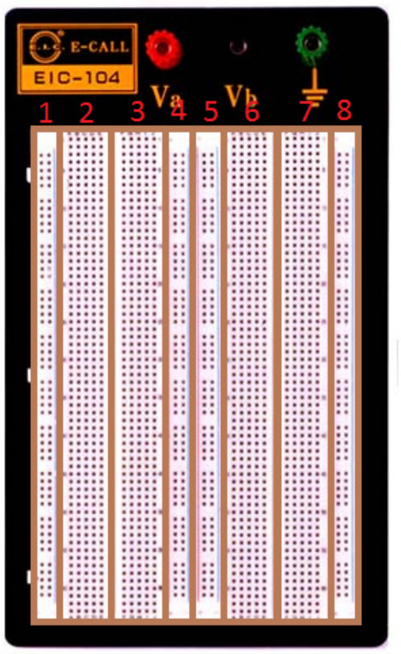
Before learning about a complex microcontroller it is recommendable to approach the technology learning with a simple one and going step by step from that to a complex one.

4.1. Developing material

Before making the Printed Circuit Board is necessary to test the different devices connected each other. For doing this we will use a Protoboard. In addition, the connection by cables allows the fact of connecting an oscilloscope and see what if something is being transmitted (and its times).

4.1.1. Protoboard

Protoboard is a board that gives to the user the possibility of design and change the circuit easily. This is very interesting for the developing process, but it is interesting for the process of learning as well. The chosen is this:



Protoboard E-CALL EIC-104

The groups 1, 4, 5 and 8 have 2 columns each one and this columns are connected (vertically) from the first row to the last. Every column is independent.

The groups 2, 3, 6 and 7 have 63 rows each one and this rows are connected (horizontally). Every group has 5 rows and are independent of the other groups (for instance: the first row is of the group 2 is not connected with the first row of the group 3.

4.1.2. Oscilloscope

An oscilloscope is an instrument to measure varying signals. In our case we will use it to detect when the outputs are enabled or disables (for example, switching on and off a LED) and when something is transmitted.

Oscilloscope Tektronix TDS 2012B

4.2. GPS

A GPS (Global Positioning System) is basically a device that gives the position over the Earth's surface. The information is given in the coordinate system detailed in the point 2 (latitude and longitude).

The GPS device we will work with is: "Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates - Version 3". It is able to give the information in several types of GPS NMEA messages. It can be configured by PMTK NMEA commands messages.

4.2.1. NMEA messages

All the specification for all the NMEA GPS and NMEA PMTK messages has the same pattern:

$CODE,DATA$_1$,...,DATA$_N$*CC

Where:

- $ = Start mark of the message
- CODE = Protocol code
- DATA$_1$,...,DATA$_N$ = Data of each protocol (comma-separated)
- * = End mark for the message
- CC = Mandatory checksum

4.2.1.1. NMEA GPS messages

All the NMEA GPS messages has the following pattern:

$GPXXX, DATA$_1$,...,DATA$_N$* CC

Where:

- $ = Start mark of the message
- GPXXX = Protocol code
- DATA$_1$,...,DATA$_N$ = Data of each protocol (comma-separated)
- * = End mark for the message
- CC = Mandatory checksum

The one we choose is our case is GPRMC. This message is by definition: Recommended minimum specific GPS/Transit data. We will use this because, as we said, one of the most important points is trying to reduce the power consumption sending as less characters as possible.

Using the same example than in the point 2, the city of Brno using GPRMC message would be:

$GPRMC,225446,A,49.2000,N,16.6167,E,000.5,054.7,191194,020.3,E*6C

Where:

- $ = Start mark of the message
- GPRMC = GPRMC protocol code
- 103648 = Time of fix: 10:36:48 UTC
- A = Indicates if the device is ready (A) or warming (V)
- 49.2000,N = Latitude 49.2000 degrees North
- 16.6167,W = Longitude 16.6167 degrees East
- 000.5 = Speed over ground (in knots)
- 054.7 = True course
- 150514 = Date of fix: 15 May 2014
- 020.3,E = Magnetic variation: 20.3 degrees East
- * = End mark for the message
- 68 = Mandatory checksum

4.2.1.2. NMEA PMTK message

All the NMEA PMTK messages has the following pattern:

$PMTKXXX,VALUE*CC

Where:

- $ = Start mark of the message
- PMTK = Protocol code
- XXX = Command code
- VALUE = New value to set
- * = End mark for end of the message
- YY = Mandatory checksum

For example, to set the baud rate, looking for in the NMEA specification we find that the command PMTK_SET_NMEA_BAUDRATE corresponds with the code 251:

```
$PMTK251,38400*27
```

Where:

- $ = Start mark of the message
- PMTK = Protocol code
- 251 = Command code
- 38400 = New baud rate value to set
- * = End mark for end of the message
- 27 = Mandatory checksum

Other example very useful related with the power consumption is to make the GPS enter in standby mode, the command is PMTK_CMD_STANDBY_MODE and its code is 161:

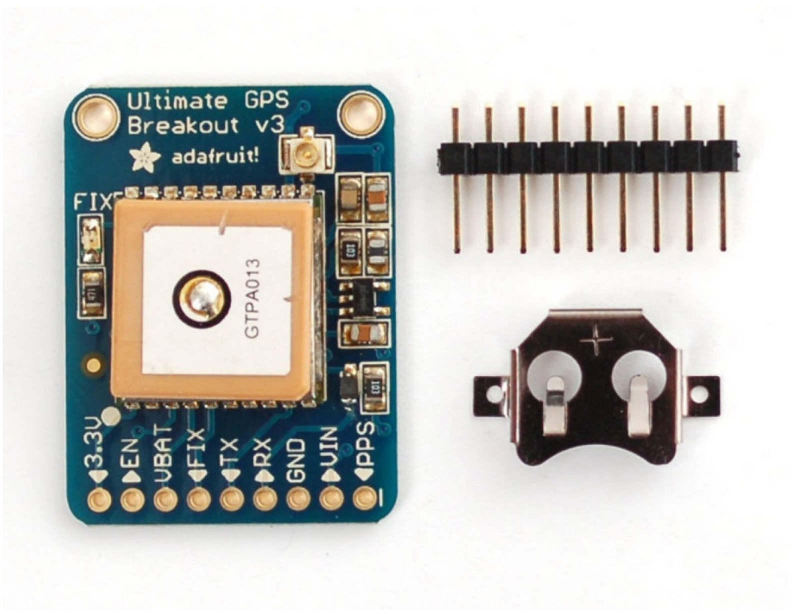
```
$PMTK161,0*28
```

Where:

- $ = Start mark of the message
- PMTK = Protocol code
- 161 = Command code
- 0 = New value (sleep mode)
- * = End mark for end of the message
- 28 = Mandatory checksum

4.2.3. Adafruit Ultimate GPS Breakout v3

This is the chosen GPS device:



Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates - Version 3

[Source: Adafruit's website[11]]

It provide the GPS information in all the NMEA GPS messages formats and it uses UART (detailed in the point 4.3.2.1). It update frequency is from 1Hz to 10 Hz. Its power consumption is very low, only 20mA, which makes this GPS device very suitable for this project.

4.3. PIC microcontrollers

PIC is a kind of microcontrollers made by Microchip Technology Inc. They are used in a lot of embedded systems.

4.3.1. Programming PIC

For programming and testing the codes developed for PIC will be used the Asix Presto connected to the pins.

ASIX PRESTO

[Source: ASIX's website[6]]

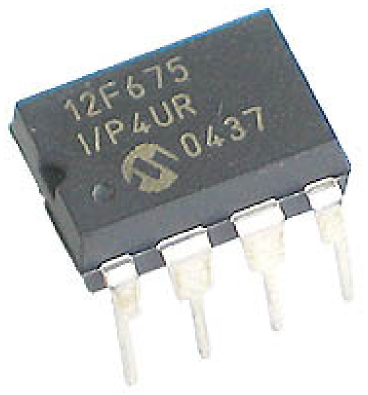
4.3.2. Learning about PIC

For learning we will focus in the family 12F that is one of the simplest family of PICs in order to get the first approaching.

The learning process consists in doing *miniprojects* with very simple codes but it is necessary to keep in mind that some register must to have a certain value, otherwise it will not work even using the right code and everything is rightly connected.

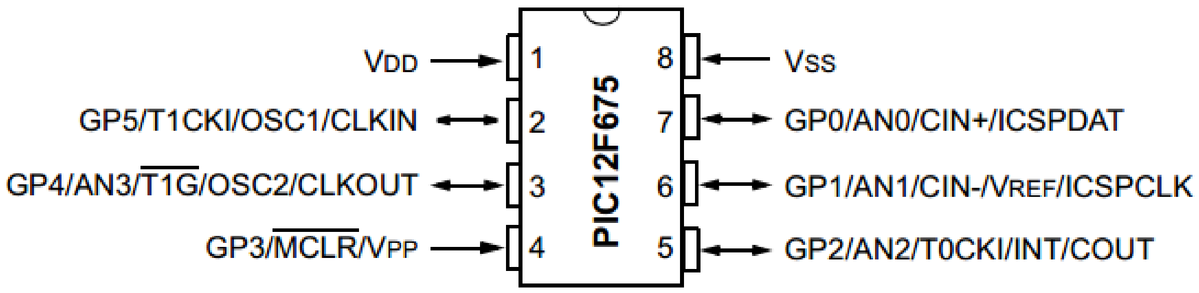
4.3.2.1. Introducing in PIC - PIC12F675

This microcontroller PIC12F675 is one of the simplest in its own family. It has only 8 pins.



Picture of the PIC12F675

[Source: Microchip's website[1]]


VDD
GP5/T1CKI/OSC1/CLKIN
GP4/AN3/$\overline{T1G}$/OSC2/CLKOUT
GP3/$\overline{MCLR}$/VPP
1
2
3
4
PIC12F675
8
7
6
5
VSS
GP0/AN0/CIN+/ICSPDAT
GP1/AN1/CIN-/VREF/ICSPCLK
GP2/AN2/T0CKI/INT/COUT

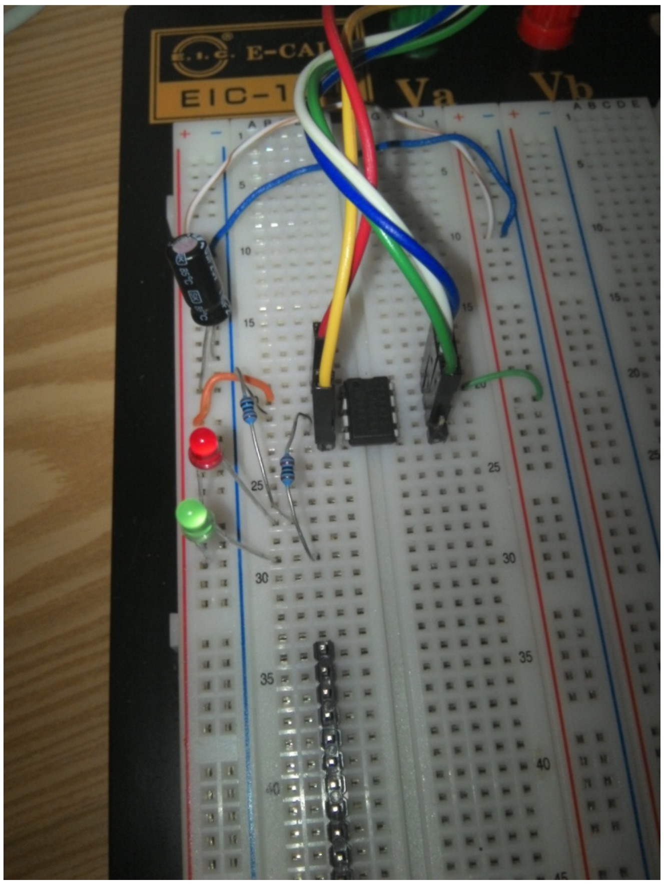
PIC12F675 pin diagram

[Source: PIC12F629/675 *Data Sheet[2]]*

This microcontroller is very simple and has 2 pins for power supply (VDD and VSS) and the rest can be configured as input or output but the pin 4 that only can be configured as input.

The first miniproject consists in make the LEDs blink alternatively the red and green LEDs, its behaviour is cyclically:

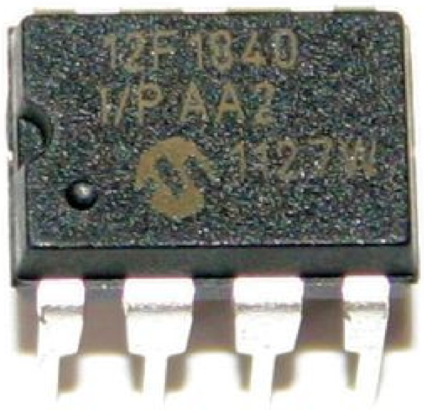
1. Red ON, Green OFF
2. Red OFF, Green ON



Circuit for the first miniproject working with PIC12F675

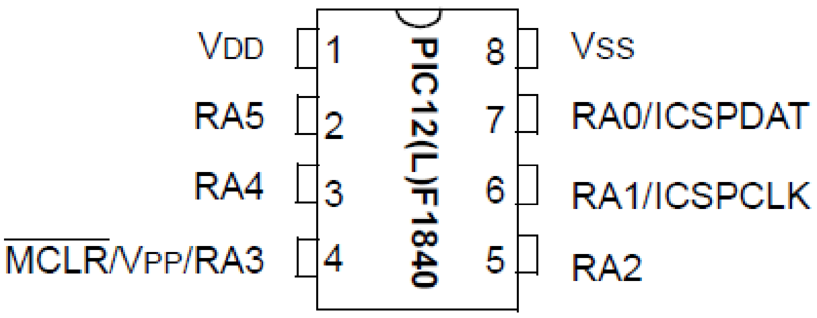
4.3.2.1. Using UART and GPS - PIC12F1840

This microcontroller is very similar to the previous one but it has an UART module.



Picture of the PIC12F1840

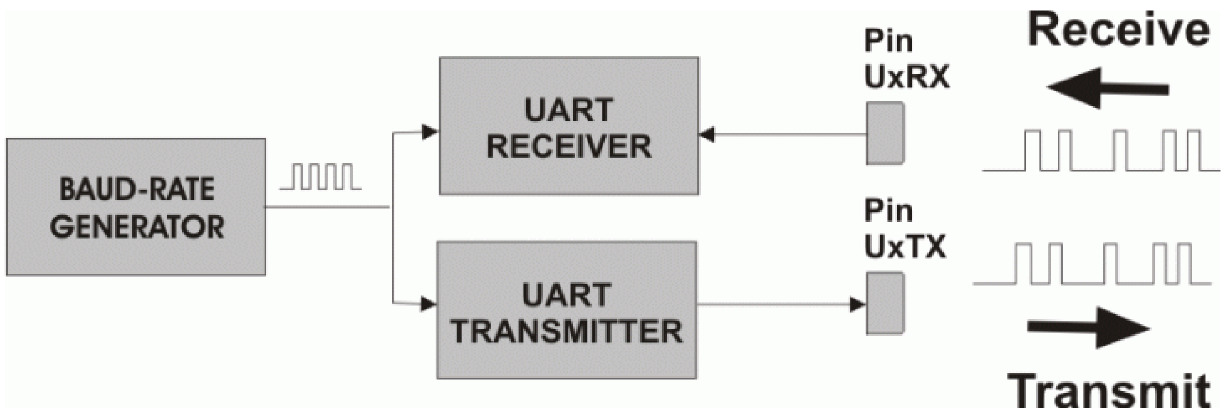
[Source: Microchip's website[1]]



PIC12F1840 pin diagram

[Source: PIC12(L)F1840 *Data Sheet[3]]*

UART (Universal Asynchronous Receiver/Transmitter) is a basic module that allows asynchronous communication. For the correct working in a communication both modules has to work at the same bit rate. This is controlled by the register UxBRG which controls the Baud-rate generator.

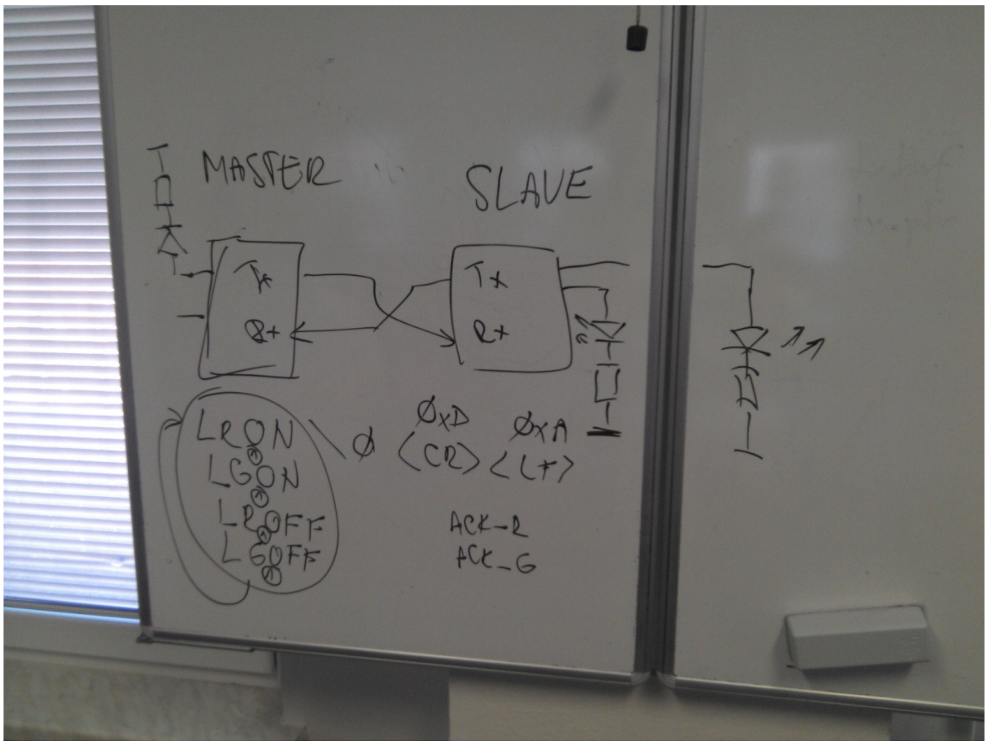


Simplified diagram of UART module

[Source: MikroElectronika's website[9]]

2 microcontrollers like this are used in order to send data from the one microcontroller to the other one.

Again, a simple miniproject to work with this microcontroller and UART bidirectional communication with this scheme:



Draft circuit diagram for the miniproject working with PIC12F1840 using UART

The first microcontroller will roll as master and has to send the command messages cyclically every second to the second one that roll as slave. The messages are the following:

1. $LRON* → *Switch on the red LED*
2. $LGON* → *Switch on the green LED*
3. $LROFF* → *Switch on the red LED*
4. $LGOFF* → *Switch on the green LED*

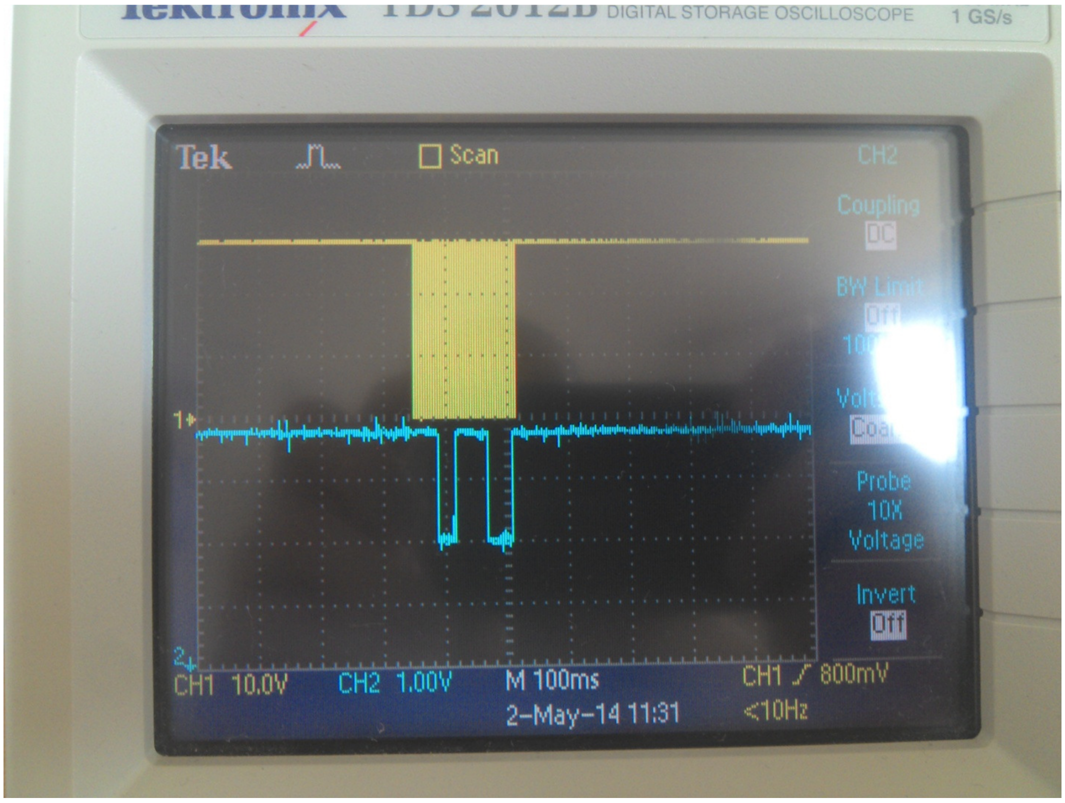
The slave hast to obey switching on/off the led red/green according to the message received and send an acknowledgment to the master:

- $ACK_R* → Acknowledgment *for red LED commands*
- $ACK_G* → Acknowledgment *for green LED commands*

Once this miniproject is done, the next consists in master sending GPRMC to Slave every second. Slave has to parse and store the information in its memory and then switch the red or green LED according if the gotten information is valid, in other word, if the GPMRC has an A or V in its validation data byte.

When it is done, then the next step is trying with the real GPS device (the one detailed in the point 4.2.3).

Then what happened is that the LED blinked 4 times every second. Analysing with the oscilloscope the result was this:



Analysing the signals of the miniproject circuit:

data sent by the GPS (yellow) and the LED behaviour (blue)

Thanks to the oscilloscope was easy to realize that is like the message was sent like 4 times. This is because the GPS by default had configured to send the information in 4 kind of NMEA GPS messages and the code only took into account the start and end marks.

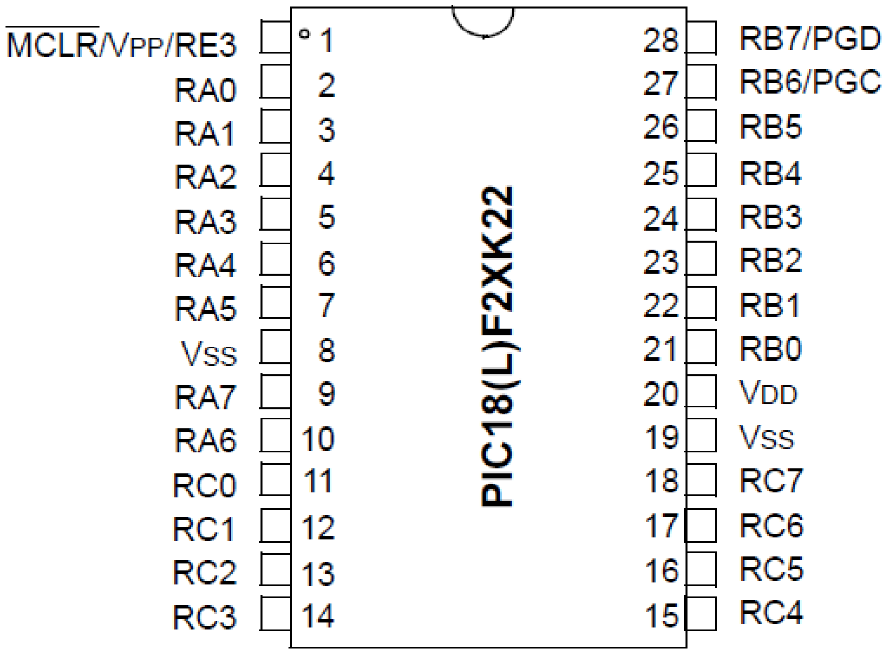
The solution was, first filtering the message detecting the GPRMC code, and then, configuring the GPS to send only the information in GPRMC format.

4.3.3. The definitely microcontroller for the project - PIC18F24K22

And finally the PIC18F24K22 that belongs to 18F family, which is more complex than 12F but is not very hard to adapt the code we have from the code for 12F family. This microcontroller has 16 pins, 2 UART modules and SPI interface.

Picture of the PIC18F24K22

[Source: Microchip's website[1]]



PIC18F24K22 pin diagram

[Source: PIC18(L)F2X/4XK22 *Data Sheet*[4]*]*

With 2 UART modules we are able to connect to each buoy 2 modules IQRF (detailed in the point 4.4) and the GPS device.

4.4. IQRF

IQRF is a wireless technology packet-oriented for low speed and low power consumption made by Microrisc s.r.o. It works via radio frequency in low bands. It can work either point-to-point networks and complex mesh networks. The meshes use IQMESH protocol and IQRF provides many facilities to create a meshes in a simple way.

This technology is very interesting for uses like telemetry and controlling as it is our case.

CK-USB-04A
DK-EVAL-04A
TR-52DA

IQRF devices

[Source: IQRF's website[14]]

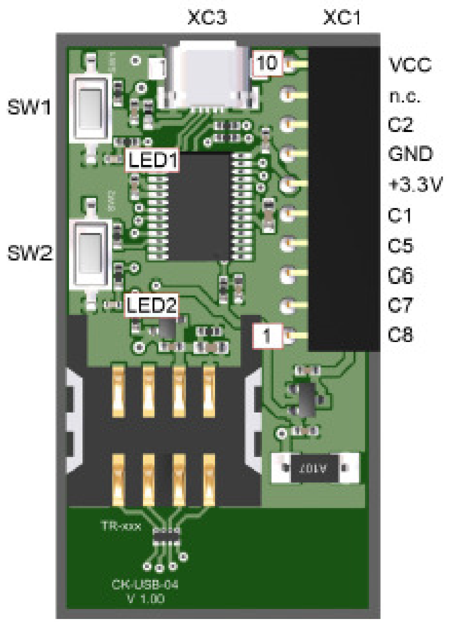
These devices are detailed later, but we need to know at least what is for each one of this devices. The IQRF modules is actually the "Smart transceiver" (in this case TR-5DA). It can be programmed using the device CK-USB-04A. The device DK-EVAL-04A cannot program the smart transceiver but has a battery and is very useful for testing.

4.4.1. Programming IQRF - CK-USB-04A

The device CK-USB-04A does not have battery and only works connected to the computer through a cable microUSB ↔ USB.

CK-USB-04A

[Source: IQRF's website[14]]



CK-USB-04A pin diagram

[Source: CK-USB-04A User guide[19]]

From the software IQRF IDE we can program the device and see what is being sending through the SPI interface. Programming the Smart receiver is very easier to resend these message via wireless (and the same for receiving wireless messages and forwarding them via SPI to watch what is being received). Some IQRF examples are codes about working in this way. CK-USB-04A allows debugging as well.

It has 2 buttons: SW1 is a programmable button and SW2 is reset.

The XC3 port is the microUSB and the ports of XC1 are the pins for making connections (similar to PIC pins).

4.4.2. Learning about IQRF

Every IQRF device kernel is actually a PIC microcontroller with built-in operating system which makes easier and faster the process of learning how to program it.

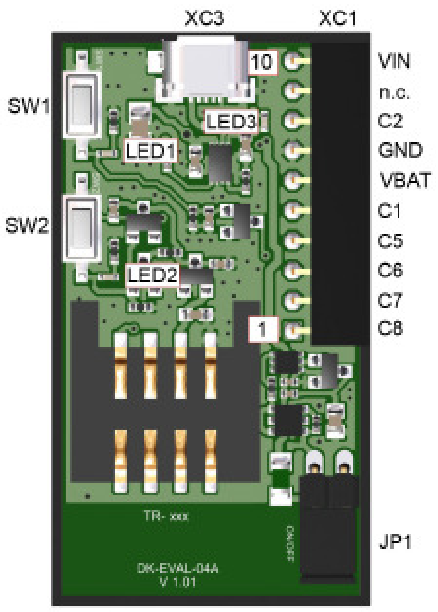
IQRF has many examples in its website for beginners and a quick start guide. This and what has been learned of PIC is enough to start programming IQRF simple point-to-point networks with only 1 transmitter and 1 receiver.

IQRF has an UART module and SPI interface which allows the connection with the microcontroller we chose.

4.4.2.1. Using self power supplied nodes - DK-EVAL-04A

DK-EVAL-04A

[Source: IQRF's website[14]]

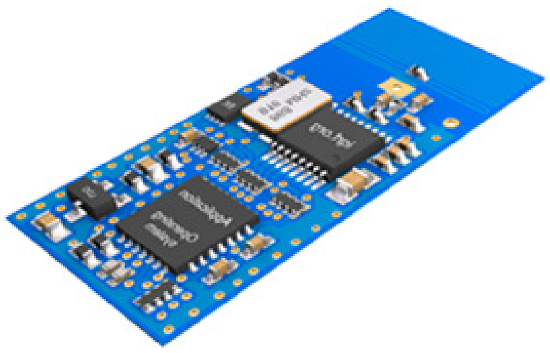


DK-EVAL-04A pin diagram

[Source: DK-EVAL-04A User guide[20]]

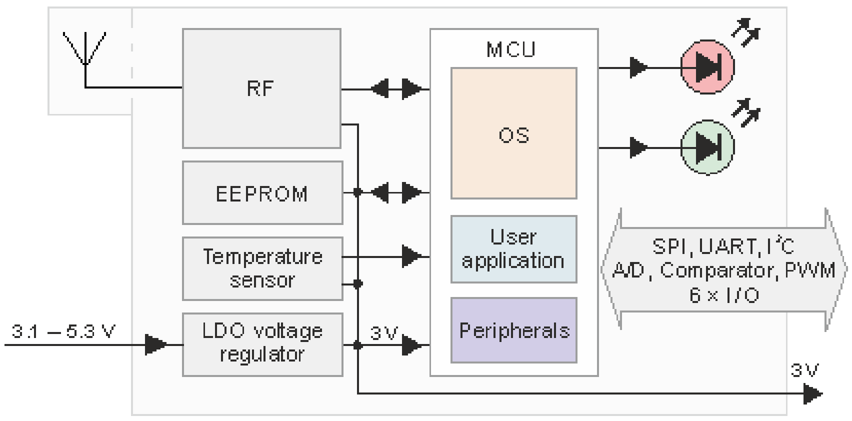
This device is very similar to the previous one (CK-USB-04A) but it has a battery and a jumper (JP1) for switching it ON/OFF. Its XC3 port is only for charging (it cannot program smart transceivers).

4.4.3. TR-52DA



TR-52DA

[Source: IQRF website[14]]



TR-52DA block diagram

[Source: TR-52D Data Sheet[21]]

This is the most important part of the IQRF systems. Is what is programmed and, in our case, it can be programmed as coordinator or node. It has a temperature sensor, a battery sensor, an RF antenna and 2 LEDs: red and green, which is very useful for developing.

5. Low power consumption

As we said, the low power consumption is one of the most important points of the project.

5.1. Techniques to reduce the power consumption

The most intuitive way of reducing power consumption is switching off part or all the modules connected to batteries in order to make it decrease. The 3 devices we have in every buoy (PIC, IQRF and GPS) have standby mode.

Another way is trying to reducing the wireless connections because they use to spend so much power. If we add the fact that every message has to be resent through the mesh nodes, this makes that the nodes closer to the coordinator will work almost all the time while the race is being held. Thus sending as less bytes as possible is one of the priorities to reduce power consumption.

According to Amdahl's law, to get the maximum improvement in the whole system improving a part of it. In our case, we have to focus in what is occupying the most common message sent, and it is the coordinates (buoys and riders position), so our aim is to reduce the GPRMC message.

5.2. Reducing the GPRMC message

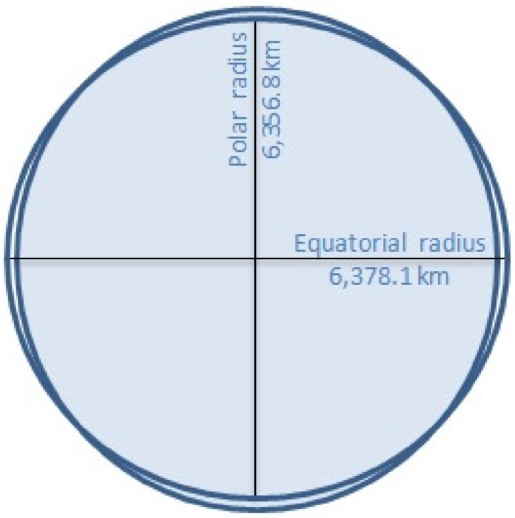
We need to cover the geopositioning of some objects in a region with size 1000 x 1000 meters and try to send as less amount of characters as possible. For this reason we need to know which part of the GPRMC message will be transmitted and which will not.

The objectives are to remove the non-necessary information and calculate how less is needed to represent the wanted area. In the first approach the calculations will be done in decimal to simplify the concept and then transformed and adapted to binary.

This system will work just over the sea. This fact makes the calculations more accurate because experimental the radius will be very close to the real.

Things to take into account:

- The GPS which we work with has 4 precision decimals
- R_p = Polar radius (6378.1370 km)
- R_e = Equatorial radius (6,356.7523 km)



Earth's shape according its radius

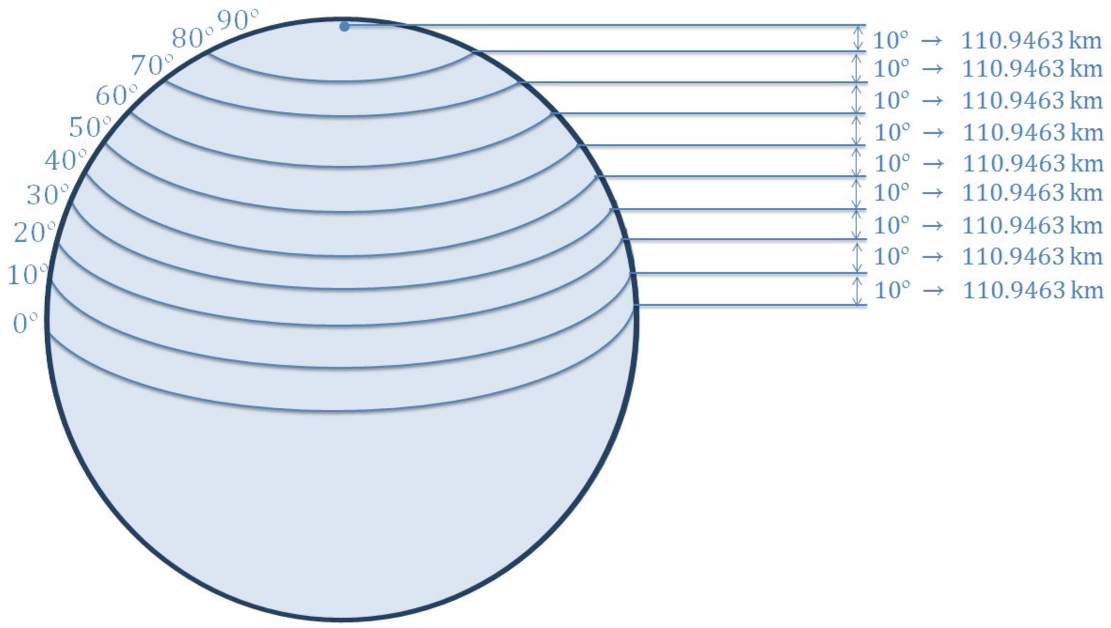
What we have to find out is how many degrees are necessary at least to represent 1 km for the worst case.

1. If we choose the biggest radius, the perimeter will be bigger, therefore the ratio km/º will be bigger as well, then the ratio º/km will be obviously smaller.
2. Analogously, if we choose the smallest radius, the perimeter will be smaller, therefore the ratio km/º will be smaller as well, then the ratio º/km will be obviously bigger.

The last case is the critical one because it needs a bigger number to represent 1 km. For this reason the calculation will be done using the Equatorial radius.

5.2.1. Latitude

The longitude is constant for all the latitudes of the Earth as we can see in the next image:



km/º ratio according the longitude

It makes the calculations very simple:

$$\mathrm{R_e} = \text{Equatorial radius } (6{,}356.7523 \text{ km})$$

$$Perimeter = 6{,}356.7523\, km \cdot 2 \cdot \pi = 39{,}940.6527 \text{ km}$$

$$\frac{39{,}940.6527 \text{ km}}{360º} = 110.9463 \text{ km}/_{º}$$

This means that every degree of longitude represents 110.9463 km over the Earth surface. With only the decimal part it is possible to represent this distance (from 0.0000º to 0.9999º we cover exactly 1º or 110.9463 km). Therefore we will need only some digits of the decimal part to represent 1 km.

To get how many degrees we need to cover 1 km is as simple as calculate the inverse:

$$\frac{1}{110.9463 \ {}^{\text{km}}/_{\text{º}}} = 0.0090 \ {}^{\text{º}}/_{\text{km}}$$

We know that we always will have fixed number of decimals: 4, because our GPS that we are working with gives the information in the format XX.XXXXº. To avoid decimals we will multiply by 10,000 (to simplify this process will be called “normalize degrees” from now on and the unit will be called "normalized degrees [nº]").

$$0.0090 \ {}^{\text{º}}/_{\text{km}} \cdot \frac{10{,}000 \ n^{\text{º}}}{1 \ \text{º}} = 90 \ {}^{\text{nº}}/_{\text{km}}$$

Then in chars it is only necessary 2 chars to represent 1 km.

Now is time to convert the information we have to binary. The normalization makes this process simpler because we can work with positive integers.

$$90_{10} \ {}^{\text{nº}}/_{\text{km}} = 1011010_2 \ {}^{\text{nº}}/_{\text{km}}$$

We need only 7 bits to represent 1 km in the longitude. In total, with 7 bits:

$$2^7 \ \text{nº} = 128 \ \text{nº}$$

$$128 \ n^{\text{º}} \cdot \frac{1 \ \text{º}}{10.000 \ n^{\text{º}}} = 0.0128^{\text{º}}$$

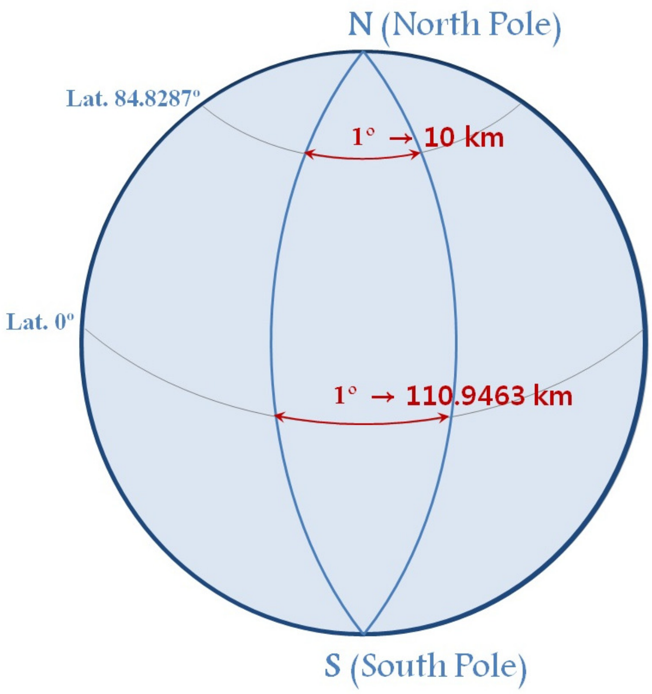
$$0.0128^{\text{º}} \cdot \frac{110.9463 \ km}{1^{\text{º}}} = 1.42 \text{ km}$$

As we just saw, with 7 bits it is possible to represent until 1,42 km for longitudes.

5.2.2. Longitude

As we just have seen, the radio km/º in the longitude is constant, in other words, no matter which latitude are we, 1º of longitude always will be 110.9463 km.

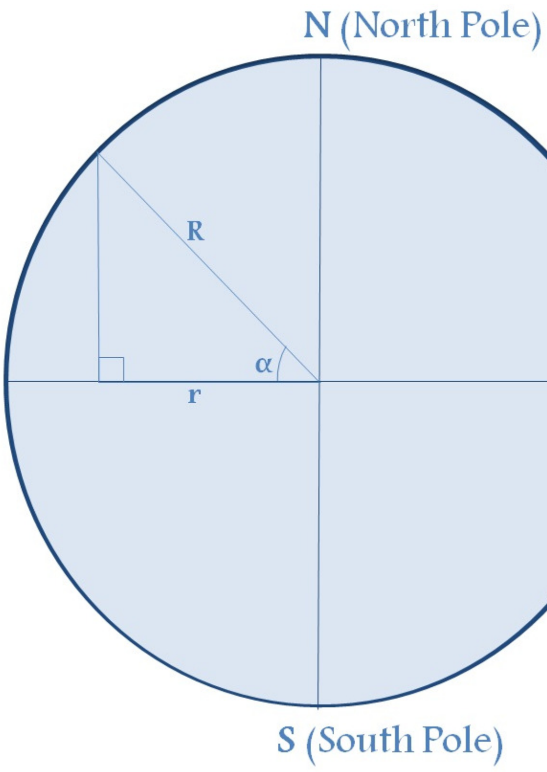
Nevertheless, the ratio "km/º" in the longitude changes according to the latitude:



km/º ratio according the latitude

For this reason, we need to know from which latitude (until the poles) the ratio º/km becomes 1 (1 º = 1km).

For the calculations we have to assume that Earth is a perfect sphere (as it is said before, it will be used the equatorial radius).



Calculating the Earth radius according the latitude

Where:

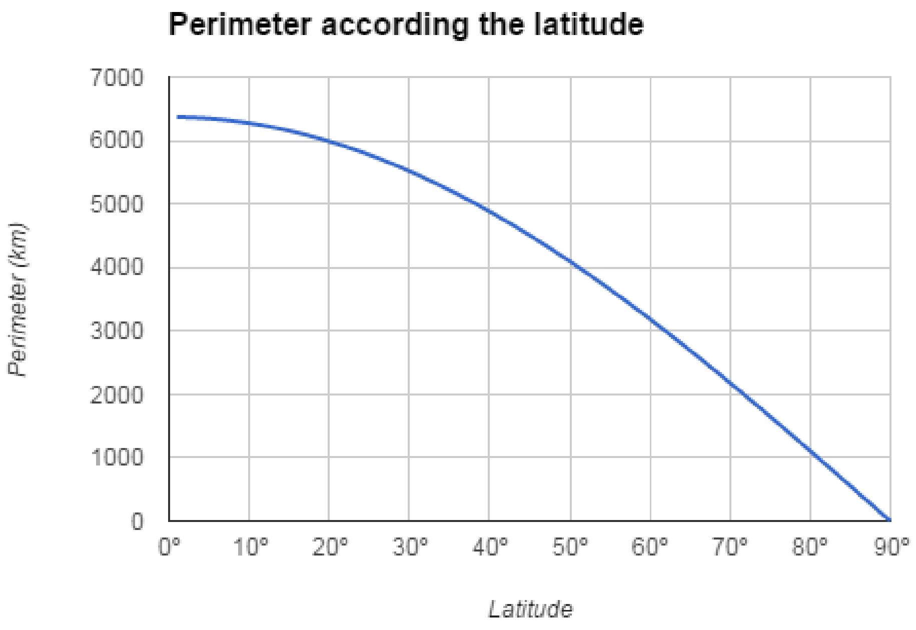
- R_e = Equatorial radius (6,356.7523 km)
- r = radius according latitude
- α = latitude

To know the radius according to latitude:

$$cos(\alpha) = \frac{r}{R_e}$$

$$r = R_e \cdot cos(\alpha)$$

In the next graphic we can see how the radius size is decreasing from the equator (0º) to one of the poles (90º).



Earth's perimeter according the latitude

First of all, we have to find out the latitude where the ratio "km/º" reaches the value 1. If 1º = 1 km and the Earth has 360º, then logically:

$$360º \cdot \frac{1km}{1º} = 360\ km$$

When the Earth perimeter decreases until 360 km, in that point 1º = 1 km. Then we need the radius (r_c) of the Earth when its perimeter measures 360 km:

$$360\ km = 2 \cdot \pi \cdot r_c$$

$$r_c = \frac{360\ km}{2 \cdot \pi}$$

Replacing in the equation we had:

$$cos(\alpha) = \frac{r_c}{R_e}$$

$$cos(\alpha) \ = \frac{\frac{360\ km}{2 \cdot \pi}}{R_e}$$

$$cos(\alpha) \ = \frac{360\ km}{2 \cdot \Pi \ \cdot \ R_e}$$

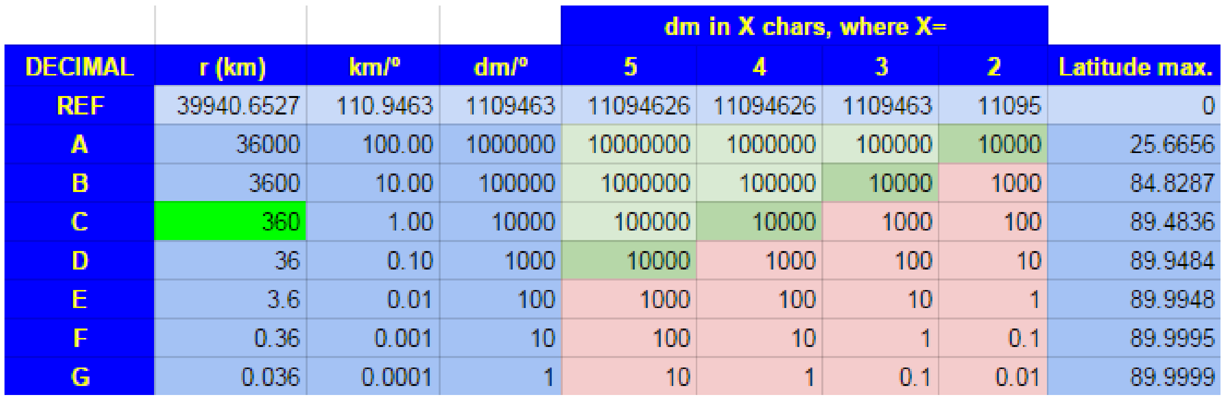
$$arccos(\frac{360\ km}{2 \cdot \Pi \ \cdot \ R_e}) = arccos(\frac{360\ km}{2 \cdot \Pi \cdot \ 6{,}356.7523\ km}) = 89{,}4836^{º}$$

The latitude 89,4836º corresponds to the poles and any race will be held there because of the cold temperatures.

Taking into account that with 1º (0º - 0.9999º) we can represent 1 km until 89,4836º, then the next step is to find out how many chars we could send as minimum. The applied technique consists in see how much decimetres (to avoid decimals) we can represent according how many chars are sent and which would be the maximum latitude that can be represented for each radius (multiplied and divided by powers of 10, because every char can represents from 0 to 9).

As we saw in the previous section in the calculations of the longitude, the technique to avoid decimals will be similar, but in this case we will work with decimetres (dm).

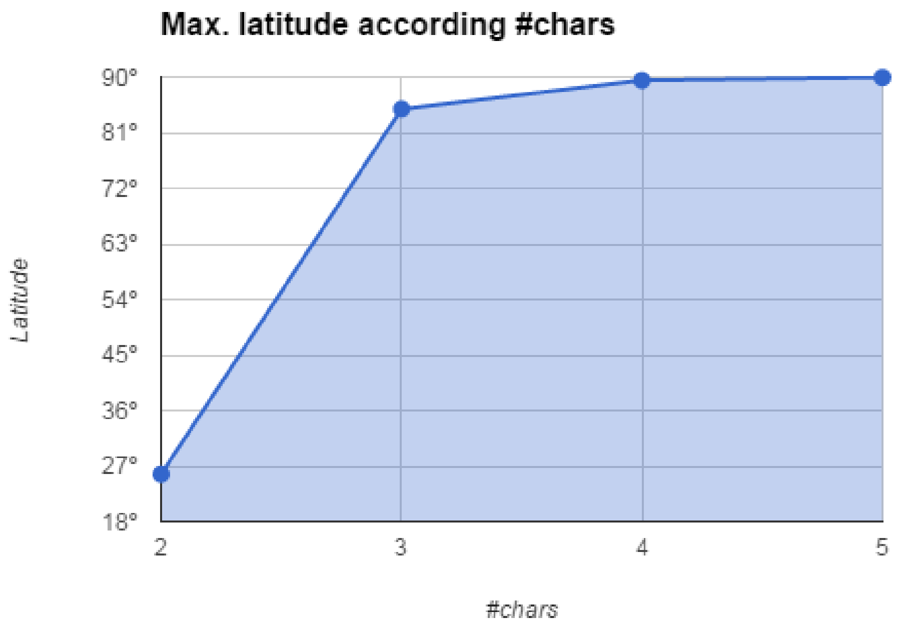
This is the result table:



				dm in X chars, where X=				
DECIMAL	r (km)	km/º	dm/º	5	4	3	2	Latitude max.
REF	39940.6527	110.9463	1109463	11094626	11094626	1109463	11095	0
A	36000	100.00	1000000	10000000	1000000	100000	10000	25.6656
B	3600	10.00	100000	1000000	100000	10000	1000	84.8287
C	360	1.00	10000	100000	10000	1000	100	89.4836
D	36	0.10	1000	10000	1000	100	10	89.9484
E	3.6	0.01	100	1000	100	10	1	89.9948
F	0.36	0.001	10	100	10	1	0.1	89.9995
G	0.036	0.0001	1	10	1	0.1	0.01	89.9999

dm that can be represented according the number

of available chars (and its maximum latitude)

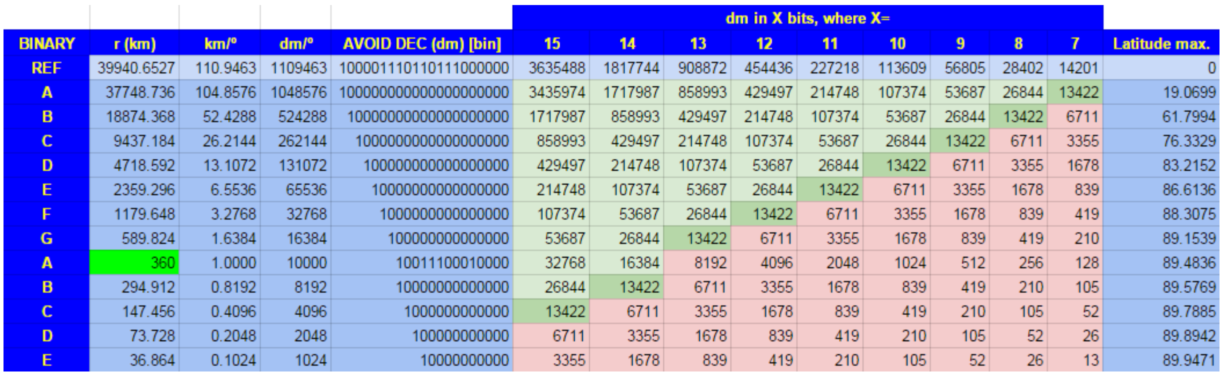


Maximum latitude according the number of available chars

The minimum we need to cover is 1 km = 10,000 dm. The green cells means that it can be represented and red cannot. In this case the best choice would be to send 3 chars because 84.8287º is more than enough. Sending 2 chars would be insufficient because this would means that the system only would work between 25,6665º N and 25,6665º S and it is a very strong restriction (it only would work more or less between the tropics).

But to optimize even more we will work in binary, so the same methodology is repeated to represent the maximum dm according the number of bits sent.

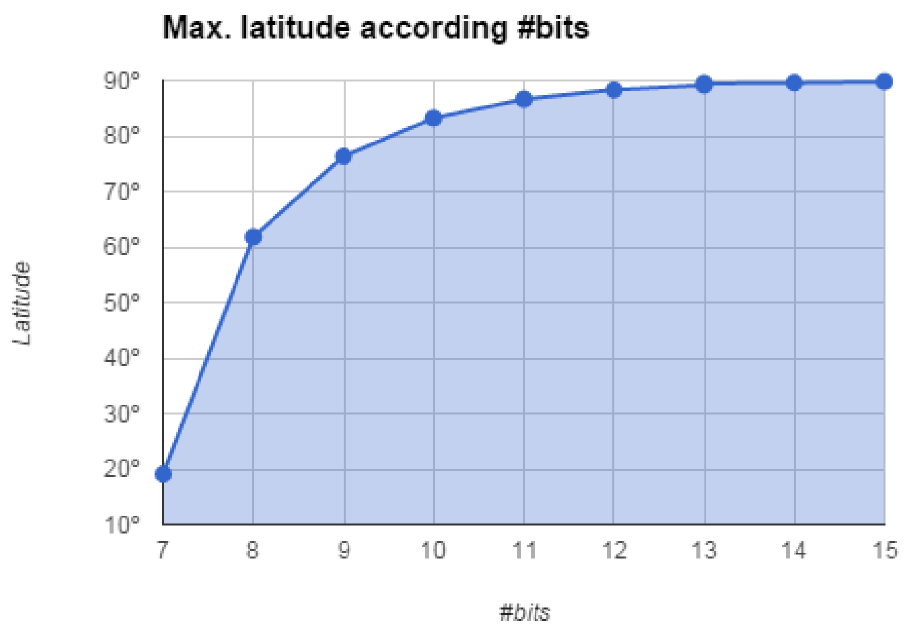
This is the result table:



					dm in X bits, where X=									
BINARY	r (km)	km/º	dm/º	AVOID DEC (dm) [bin]	15	14	13	12	11	10	9	8	7	Latitude max.
REF	39940.6527	110.9463	1109463	100001110110111000000	3635488	1817744	908872	454436	227218	113609	56805	28402	14201	0
A	37748.736	104.8576	1048576	100000000000000000000	3435974	1717987	858993	429497	214748	107374	53687	26844	13422	19.0699
B	18874.368	52.4288	524288	10000000000000000000	1717987	858993	429497	214748	107374	53687	26844	13422	6711	61.7994
C	9437.184	26.2144	262144	1000000000000000000	858993	429497	214748	107374	53687	26844	13422	6711	3355	76.3329
D	4718.592	13.1072	131072	100000000000000000	429497	214748	107374	53687	26844	13422	6711	3355	1678	83.2152
E	2359.296	6.5536	65536	10000000000000000	214748	107374	53687	26844	13422	6711	3355	1678	839	86.6136
F	1179.648	3.2768	32768	1000000000000000	107374	53687	26844	13422	6711	3355	1678	839	419	88.3075
G	589.824	1.6384	16384	100000000000000	53687	26844	13422	6711	3355	1678	839	419	210	89.1539
A	360	1.0000	10000	10011100010000	32768	16384	8192	4096	2048	1024	512	256	128	89.4836
B	294.912	0.8192	8192	10000000000000	26844	13422	6711	3355	1678	839	419	210	105	89.5769
C	147.456	0.4096	4096	1000000000000	13422	6711	3355	1678	839	419	210	105	52	89.7885
D	73.728	0.2048	2048	100000000000	6711	3355	1678	839	419	210	105	52	26	89.8942
E	36.864	0.1024	1024	10000000000	3355	1678	839	419	210	105	52	26	13	89.9471

dm that can be represented according the number

of available bits (and its maximum latitude)



Maximum latitude according the number of available bits

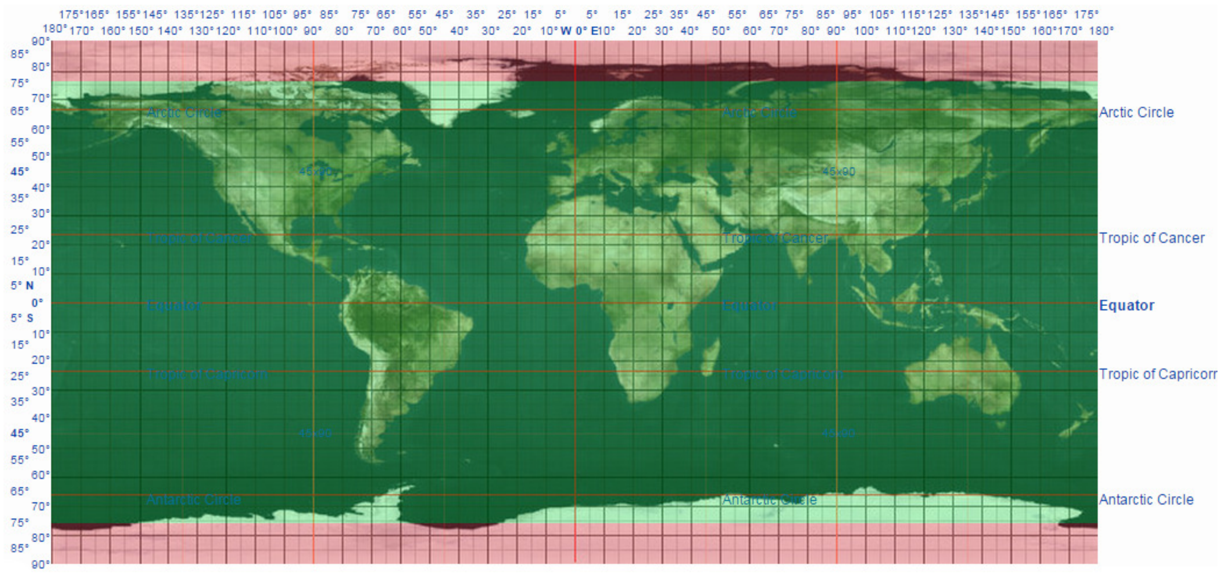
5.2.3. Conclusions

First of all, working in bits we can reduce the numbers significantly and the only thing we have to assume is multiply by 10,000 to avoid decimals (and then divide by 10,000 when the data will be treated).

To represent the latitude number we need only 7 bits, and to represent North or South we would need another bit (N→0, S→1).

We can assume that all the races will be held at least in the latitudes between 0º and 76.3329º (N and S), therefore we need as a minimum 9 bits. To represent East or West we would need another bit (E→0, W→1).

The fact of send or not the North/South and East/West bits could be optional, but if a race is held just in the equator or in the prime meridian, then the system would not work because there would be same value for 2 different points (for example: 0.0001 N and 0.001 S, and analogously 0.0001 E and 0.0001 W) and this system must be prepared for all the realistic latitudes and longitudes where a race could be held.



Zone where the system works (green) and where not (red)

[Source: Wikipedia[23]] + edited

Taking the same example, Brno had this NMEA GPS message:

$GPRMC,225446,A,49.2000,N,16.6167,E,000.5,054.7,191194,020.3,E*6C

Assuming that the data is valid and not warming (character A):

- Latitude: 49.2000_{10},N
- Longitude: 16.6167_{10},E

Multiplied by 10,000:

- Latitude: 492000_{10},N
- Longitude: 166617_{10},E

Transformed to binary:

- Latitude: 1111000000111100000_{2}, 0
- Longitude: 101000101011011001_{2}, 0

And taking only 7 bits for latitude (+ 1 for North/South hemisphere) and 9 bits for longitude (+ 1 for East-West hemisphere):

- Latitude: $1100000_{2,}$ 0
- Longitude: $011011001_{2,}$ 0

So with this format system we need only:

$$(7+1)+(9+1)=18\ bits$$

$$\left\lceil 18\ bits \cdot \frac{1\ bytes}{8\ bits} \right\rceil = 3\ bytes$$

Only 3 bytes with 6 free bits to add some extra information (for example, if the data is valid or the GPS is warming with 1 bit (A→1, V→0).

We reduced from 66 characters (66 bytes) to only 3 bytes. It is a very important reduction and it allows to reduce critically the power consumption. In addition, this also improves the speed of the system because the messages are send (and resend by every node) very much faster.

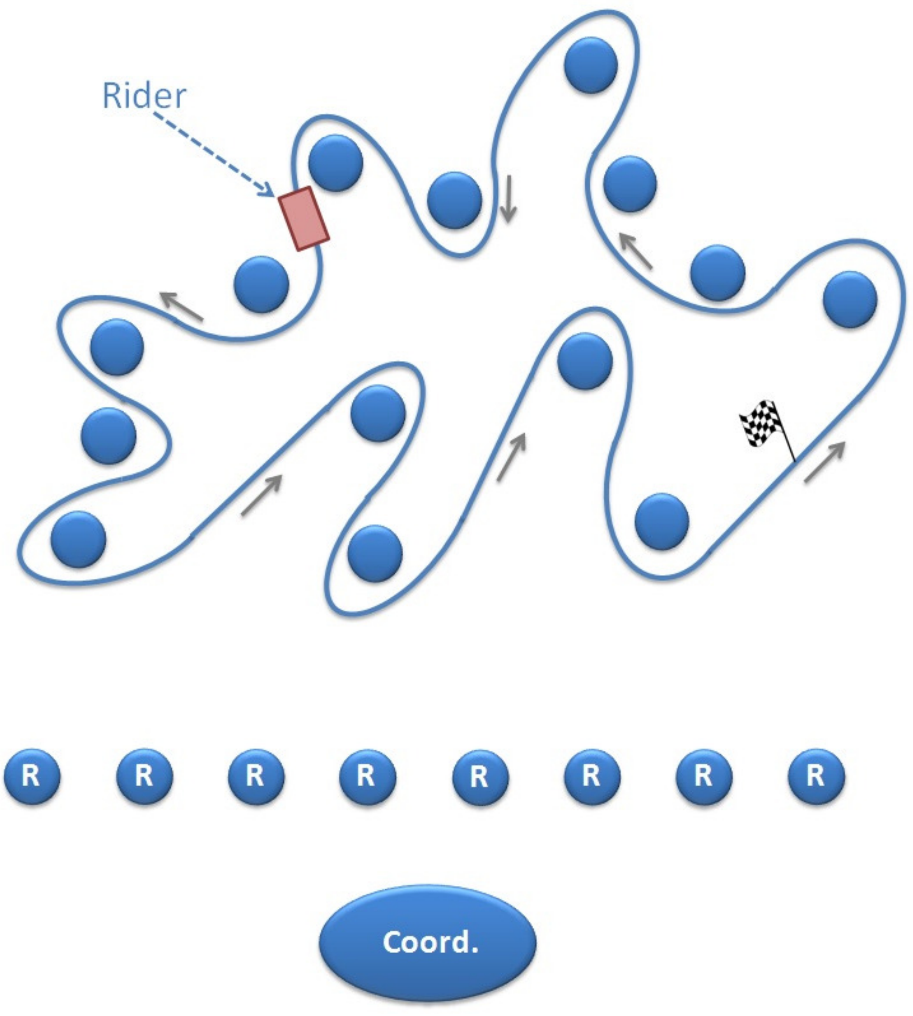
5.3. Sleep when there is nothing to do

As it is said before, switch off modules connected to the batteries reduce as well the power consumption. For doing this we have to see how to do it for every device:

Device	Make it sleep	Wake it up
Adafruit GPS	Sending to it the command "$PMTK161,0*28" via UART	Sending to it any byte via UART
PIC18F24K22	By code: OSCCON.IDLEN=0	By code: OSCCON.IDLEN=1
IQRF → TR	By code: iqrfSleep()	Power OFF/ON, watchdog timeout or on the C5 pin change
IQRF → RF IC	By code: setRFsleep ()	By code: setRFready()

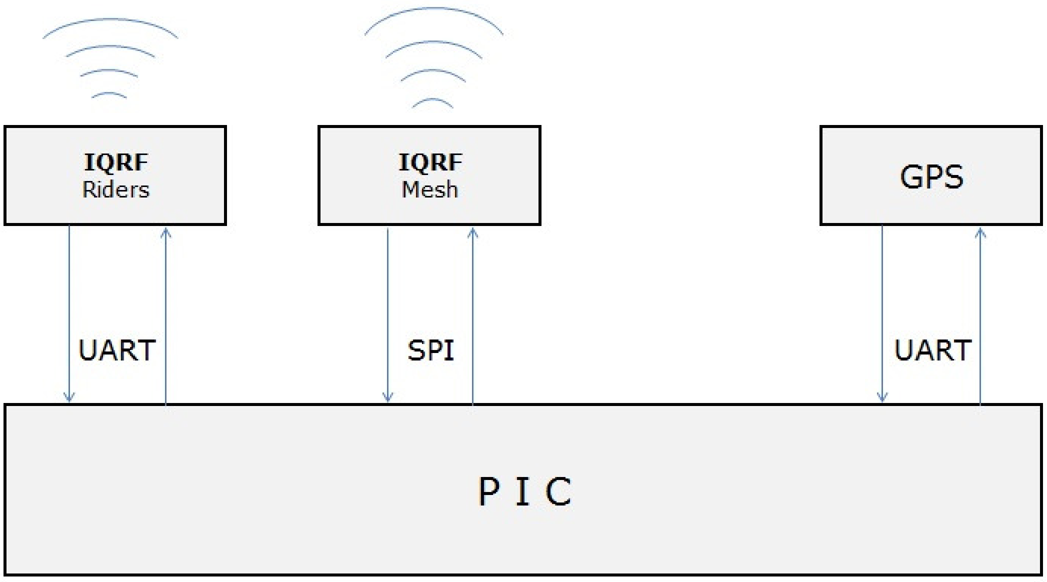
6. Scenario

The scenario of the races consists in buoys floating in the water in an 1000 x 1000 meters area as a maximum.



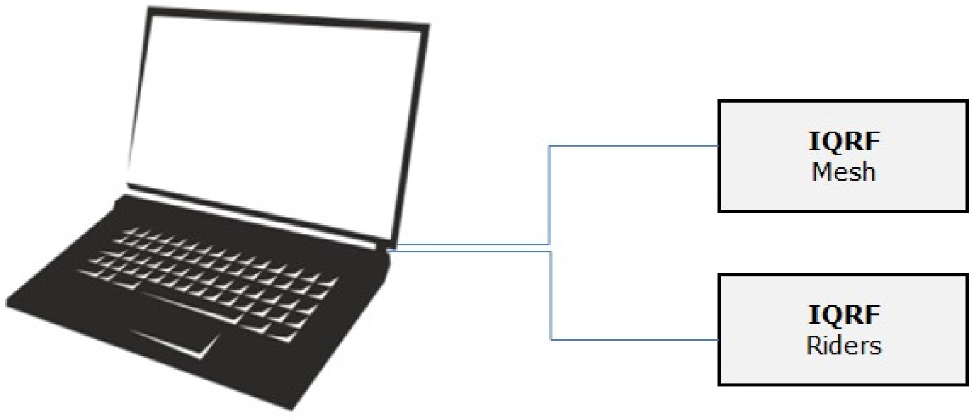
Scenario diagram

Every buoy have a PCB with a PIC, 2 IQRF modules working in 2 frequencies (1 for the mesh where every buoy is a node and another to get the closer riders).



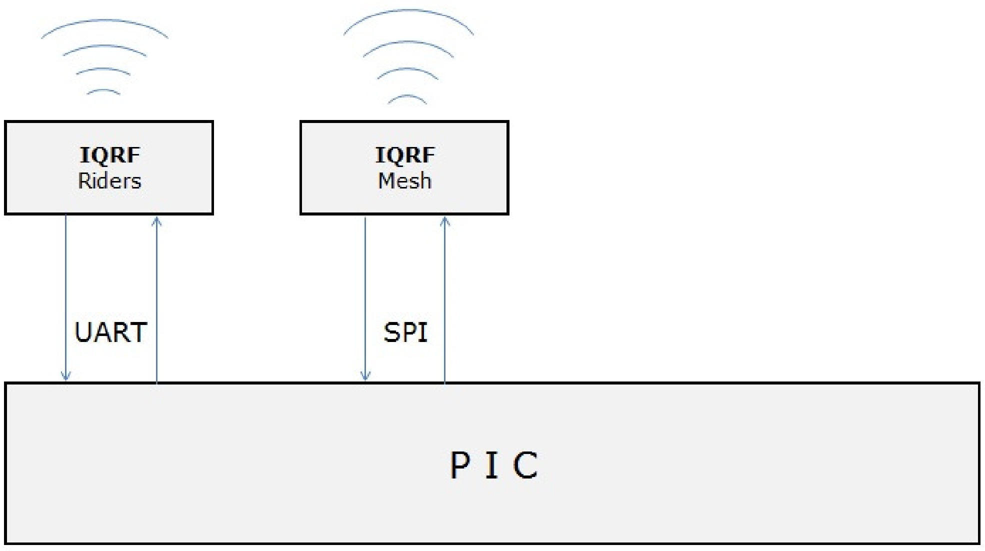
Buoy devices diagram

The coordinator consists in a pc in one of the sides of the race area connected to a 2 IQRF modules: one for the mesh and another one for the riders.



Coordinator diagram

In all of the side of the coordinator there are many repeater which are very similar to the buoys but without GPS, and their only task is to resend the messages. They are redundant and its use is just to make the mesh more reliable.



Repeater diagram

7. Specification

Following the criteria of sending as less data as possible, the commands are bit-level defined but trying to has an easy way of interpreting what kind of command is.

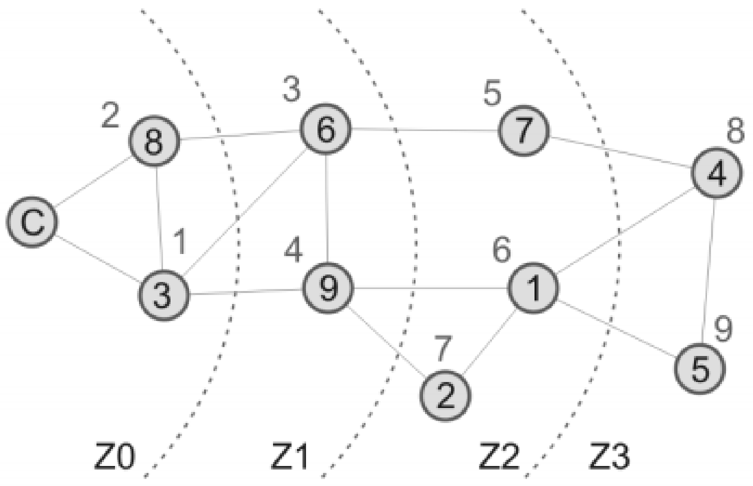
7.1. IQRF mesh

With all the buoys interconnected we get the IQRF mesh. The mesh is responsible of the synchronization.

7.1.1. DFM protocol

The chosen protocol is DFM (Discovered Full Mesh). It allows up to 240 devices and original random placement of the nodes. The way to work is the next:

1. Bond all the nodes
2. Run Discovery
    - It makes a routing backbone divided in zones according the minimum number of hops
3. If the topology changes, run Discovery again



DFM protocol graphic

[Source: IQRF OS User's Guide[17]]

In our case, once the buoys are set in their places they will stay more or less in the same place, so the topology of the mesh will not change.

7.1.2. Coordinator

Its tasks are (mainly):

- Manage the mesh
- Send messages Unicast + Broadcast
- Send orders and request data to/from the buoys
- Receive data from the riders

7.1.3. Buoy (node)

Its tasks are (mainly):

- Set its own address (bond) and unbond
- Set power-down (standby)
- Get (send) information about the battery status
- Forward messages to PIC from the coordinator
- Forward messages from PIC to the coordinator

7.2. PIC

The microcontroller is the kernel of the buoy. Its tasks are (mainly):

- LEDs:
    - ON
    - OFF
    - Blink
    - Duty (0% - 100%)
- GET
    - Data and status of the buoy's GPS
- Configure GPS
- Send riders' data

If data from the riders is received, this data is sent through the IQRF mesh synced according which driver is the data from. This is only as support to the IQRF network, because in some points of the race a rider can be very far from the coordinator.

7.3. IQRF riders

IQRF devices set in the Jetsurfs provide the data about their position (and maybe an extra telemetry information) using IQRF in another frequency.

7.3.1. Coordinator

Every 200 ms, get the information of each one of 5 riders (what implies that every IQRF rider has 40 ms to send its information).

7.3.2. Nodes

The nodes of every rider and synchronized and calculates (and send) its information during its "window time". For synchronization the parsing of the GPS message is used (the time 0 is considered when PIC reads the GPS code GPRMC).

7.4. Messages protocol

First of all, some key characters are defined for interpreting the commands:

- N: Does not matter, either 0 or 1
- V: Value (to set)
- X: Defined later

The structure of the commands is always:

CMD ADR [PAR]

Where:

- CMD: Command (1 byte)
- ADR: Destination address (1 byte)
- PAR: Parameters [only for setters and if it is needed] (1 or several bytes)

And the return policy is 1 or several bits. The coordinator adds "OK" or "ERR" in the firsts characters to know if the command have been correctly executed, and if it is "OK" then information about the sender.

7.4.1. Command byte

What is wanted to do is defined by only one byte:

X_7X_6 $X_5X_4X_3X_2X_1X_0$

Where:

- X_7X_6: Command group (set/get/mesh)
- $X_5X_4X_3X_2X_1X_0$: Specific command of the specified group

7.4.1.1. Set

The set commands have the following pattern:

00 XXXXXX

7.4.1.2. Get

The set commands have the following pattern:

01 XXXXXX

Get is only allowed for unicast.

7.4.1.3. Mesh management

The mesh management commands have the following pattern:

10 XXXXXX

And the available commands are:

- X_7X_6 $X_5X_4X_3X_2X_1X_0$ = 10 000000: Bond
- X_7X_6 $X_5X_4X_3X_2X_1X_0$ = 10 000001: Unbond
- X_7X_6 $X_5X_4X_3X_2X_1X_0$ = 10 000010: Clear all the bondings
- X_7X_6 $X_5X_4X_3X_2X_1X_0$ = 10 000011: Run "Discovery"

7.4.1.4. Reserved

There is a space for future implementations following this pattern:

11 XXXXXX

7.4.2. Buoy address byte

The whole second byte sent is to specify the destination address:

$X_7X_6X_5X_4X_3X_2X_1X_0$

The address can be:

- 0: autobonding (only used for bonding)
- 1-239: unicast
- 255: broadcast

7.4.3. Data byte(s)

The commands that need extra information are sent after the address in one or several bytes.

7.4.4. Commands list and their data

According to which command is sent, there will be answer or not, and the data information that every command needs will be different.

The return policy is different for every command, but basically is the requested information.

7.4.4.1. LEDs getter

The getter is the same for all the commands of the LED control:

- $X_5X_4X_3X_2X_1X_0$ = 0000NN
    - Get: The information about how is the LED (ON or OFF and the duty)
        - Returns 1 byte: X_7 $X_6X_5X_4X_3X_2X_1X_0$
            - X_7: Status of the LEDs (0→ OFF, 1 → ON)
            - $X_6X_5X_4X_3X_2X_1X_0$: Intensity (percent between 0 and 100)

7.4.4.2. Switch OFF the LEDs

- Command $X_5X_4X_3X_2X_1X_0$ = 000000
  - Set: Switch OFF the LEDs

7.4.4.3. Switch ON the LEDs

- Command $X_5X_4X_3X_2X_1X_0$ = 000001
  - Set: Switch ON the LED

7.4.4.4. Make the LEDs blink

- Command $X_5X_4X_3X_2X_1X_0$ = 000010
  - Set: Make the LED blinks

7.4.4.5. Set the duty of the LED

- Command $X_5X_4X_3X_2X_1X_0$ = 000011
  - Set: Set the duty of the LED
- Parameter: $X_7X_6X_5X_4X_3X_2X_1X_0$ = YYYYYYYY
  - Where YYYYYYY = % of duty to be set (from 0 to 100)
    If it is greater to 100, the duty will be set to 100%

7.4.4.6. Manage GPS

The getter gives information about the current position of the buoy

- $X_5X_4X_3X_2X_1X_0$ = 000100
  - Get: The information of the buoy position
    - Returns: 3 bytes
  - Set: Send command to the GPS
    - Several bytes in format NMEA PMTK

7.4.4.7. Get battery status

The getter gives information about the battery status:

- $X_5X_4X_3X_2X_1X_0$ = 000101
  - Get: The information about the battery status:
    - Returns: 1 byte
      - Percent about battery status (0% - 100%)

7.4.5. Set/get commands summary

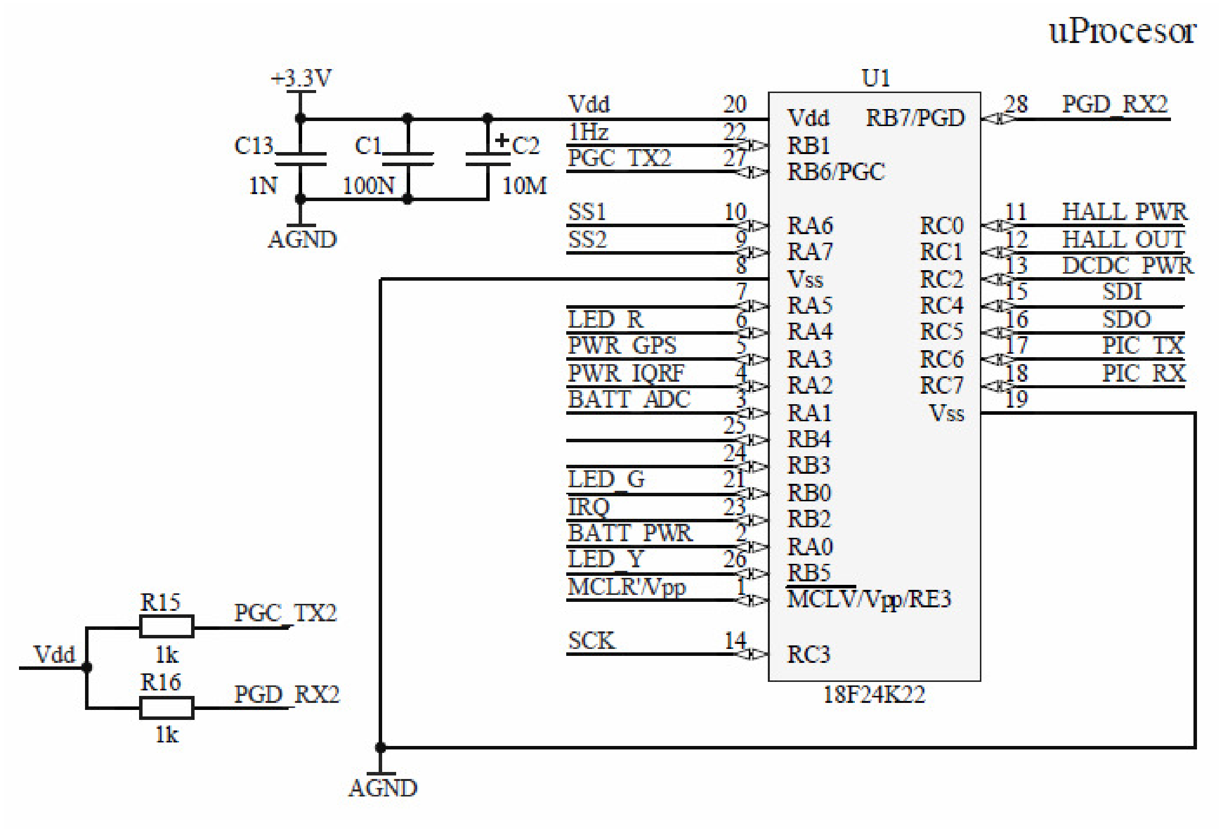
Command	Get	Set	Parameters
0000	Get status of the LED	Switch OFF the LEDs	
0001	Get status of the LED	Switch ON the LEDs	
0010	Get status of the LED	Make blink the LEDs	
0011	Get status of the LED	Set the duty of the LEDs	Duty of the LED
0100	Get GPS information	Set GPS command	Command for the GPS
0101	Get Battery information	---	
0111	---	Make buoy sleep	

8. Implementation

The implementation of the system consists in uses this system in PCBs (Printed Circuit Boards).

8.1. Hardware

This is the PIC microcontroller scheme inside of the PCB with all the connections.



PCB circuit scheme

This PCB is ready to be programmed and become a buoy of the system. With several buoy we can finally generate the wanted mesh to control the Jetsurf races.

8.2. Software

The software used in this project is specific for every device because the code is in a very low level.

8.2.1. Working environment

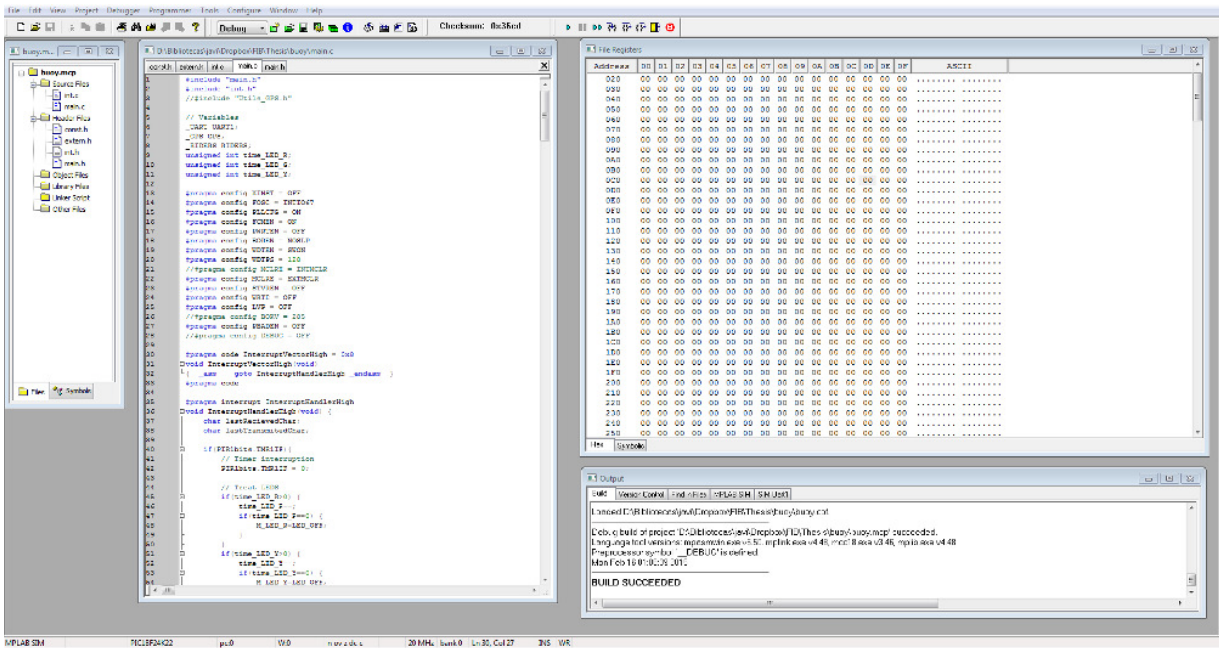
The used operating system has been Windows 7 because majority of the software for programming this devices are made for windows.

8.2.2. PIC

Programming PIC requires some software application: MPLAB to compile (and debug) the code, a C Compiler and Asix UP to upload the binaries to the microcontroller.

8.2.2.1. MPLAB IDE

Microchip provide its own IDE for working with their microcontrollers called MPLAB IDE. Is up to the user use another compiler, MPLAB IDE allows the user to choose another one compiling from the same application. It provides a debugger even with simulated UART.



```
#include "main.h"
#include "init.h"
//#include "Utils_GPS.h"

// Variables
_UART UART1;
_GPS GPS;
_RIDERR RIDERR;
unsigned int time_LED_R;
unsigned int time_LED_G;
unsigned int time_LED_Y;

#pragma config XINST = OFF
#pragma config FOSC = INTIO67
#pragma config PLLCFG = ON
#pragma config FCMEN = OF
#pragma config PWRTEN = OFF
#pragma config BOREN = NOSLP
#pragma config WDTEN = SWON
#pragma config WDTPS = 128
//#pragma config MCLRE = INTMCLR
#pragma config MCLRE = EXTMCLR
#pragma config STVREN = OFF
#pragma config WRTC = OFF
#pragma config LVP = OFF
//#pragma config BORV = 285
#pragma config PBADEN = OFF
//#pragma config DEBUG = OFF

#pragma code InterruptVectorHigh = 0x8
void InterruptVectorHigh(void)
{  _asm    goto InterruptHandlerHigh _endasm  }
#pragma code

#pragma interrupt InterruptHandlerHigh
void InterruptHandlerHigh(void) {
    char lastRecievedChar;
    char lastTransmitedChar;

    if(PIR1bits.TMR1IF){
        // Timer interruption
        PIR1bits.TMR1IF = 0;

        // Treat LEDS
        if(time_LED_R>0) {
            time_LED_R--;
            if(time_LED_R==0) {
                M_LED_R=LED_OFF;
            }
        }
        if(time_LED_Y>0) {
            time_LED_Y--;
            if(time_LED_Y==0) {
                M_LED_Y=LED_OFF;
```

```
Address  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   ASCII
 020     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 030     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 040     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 050     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 060     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 070     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 080     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 090     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 0A0     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 0B0     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 0C0     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 0D0     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 0E0     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 0F0     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 100     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 110     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 120     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 130     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 140     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 150     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 160     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 170     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 180     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 190     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 1A0     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 1B0     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 1C0     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 1D0     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 1E0     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 1F0     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 200     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 210     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 220     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 230     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 240     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
 250     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........ ........
```

```
Loaded D:\Bibliotecas\javi\Dropbox\FIB\Thesis\buoy\buoy.cof
----------------------------------------------------------------------
Debug build of project 'D:\Bibliotecas\javi\Dropbox\FIB\Thesis\buoy\buoy.mcp' succeeded.
Language tool versions: mpasmwin.exe v5.50, mplink.exe v4.48, mcc18.exe v3.46, mplib.exe v4.48
Preprocessor symbol `__DEBUG' is defined.
Mon Feb 16 01:00:09 2015
----------------------------------------------------------------------
BUILD SUCCEEDED
```

```
MPLAB SIM    PIC18F24K22    pc:0    W:0    n ov z dc c    20 MHz    bank 0    Ln 30, Col 27    INS    WR
```

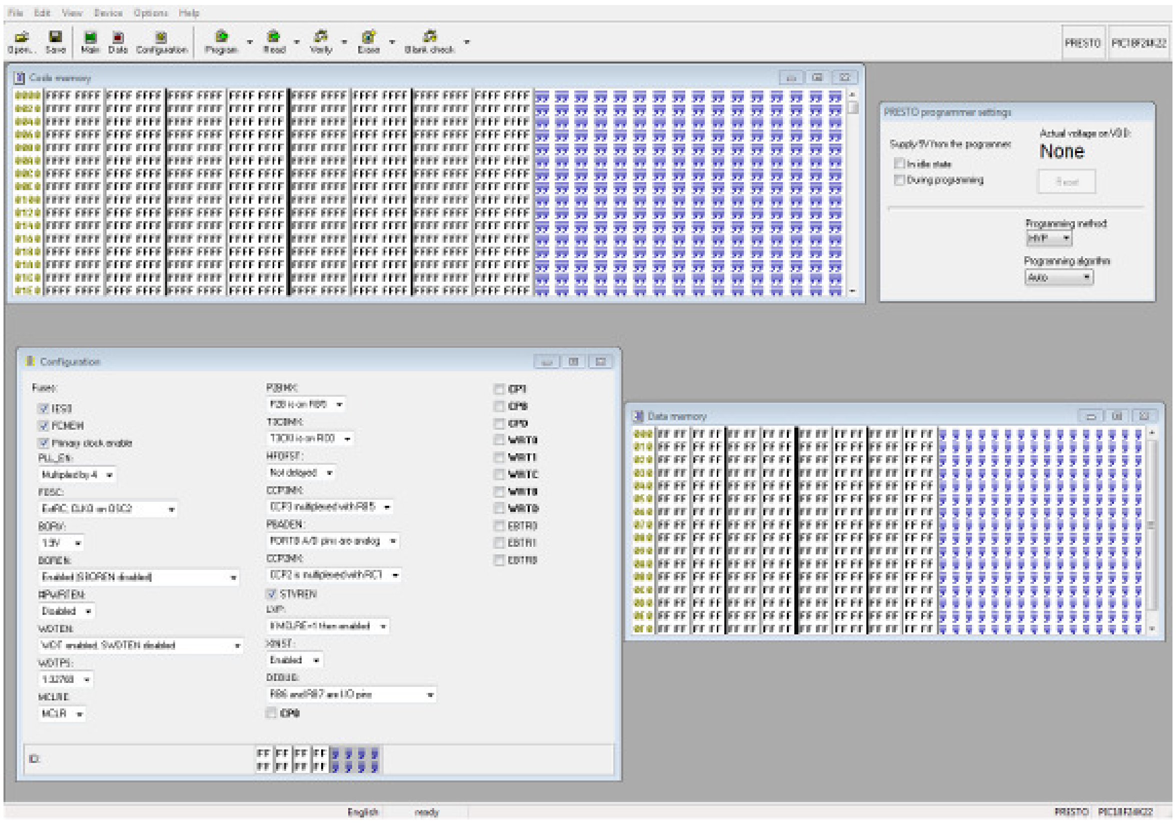
Working with MPLAB IDE

8.2.2.2. C Compiler

For the learning PICs of the family 12F was used the compiler B Knudsen Data (BNKD from now on) CC5X. Nevertheless, for the PIC used in the buoy of the family 18F was used the compiler BNKD CC8E.

8.2.2.3. ASIX UP

ASIX UP is an application that uses the ASIX PRESTO device to program PIC microcontrollers. It allows to configure some parameters from its interface.



File Edit View Device Options Help
Open... Save Main Data Configuration Program Read Verify Erase Blank check
PRESTO PIC18F2422
Code memory
Data memory
English ready

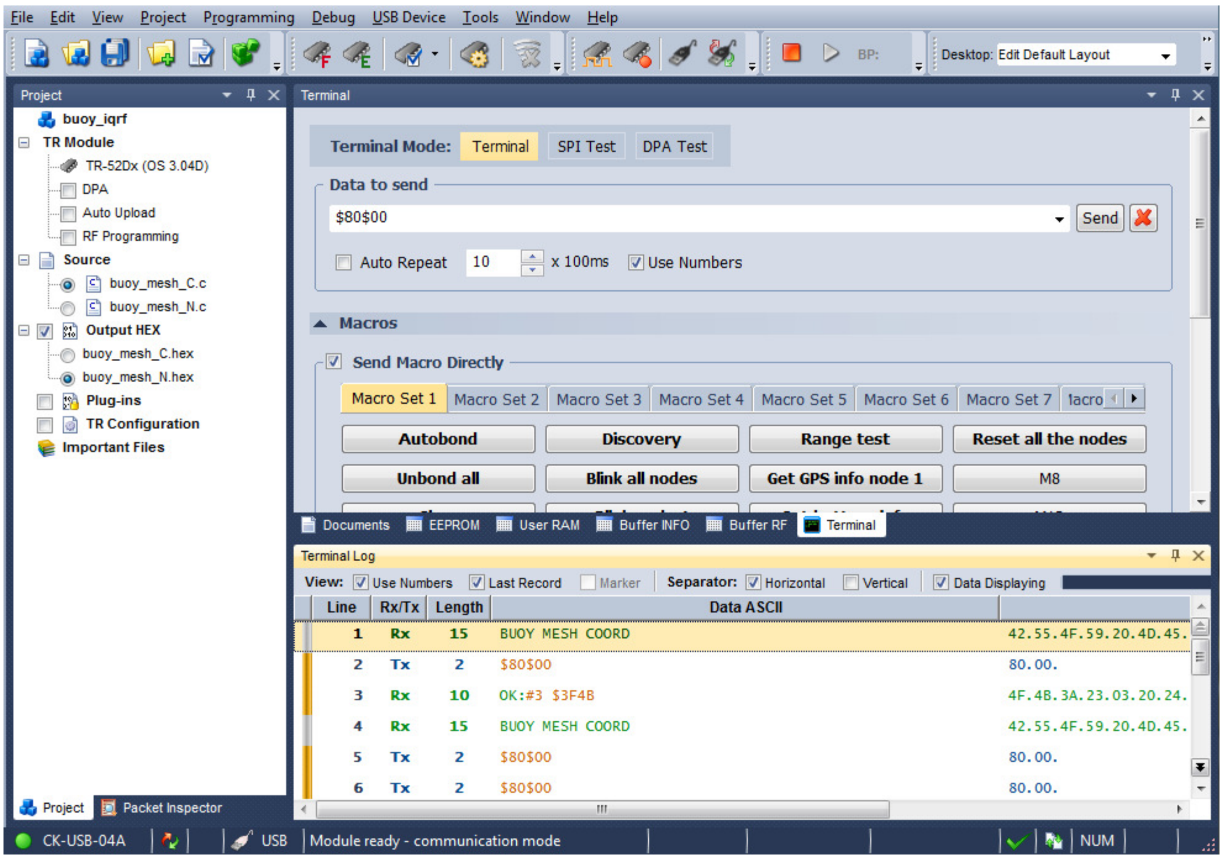
PRESTO programmer settings
Supply 5V from the programmer:
☐ In idle state
☐ During programming
Actual voltage on VDD: None
Programming method: HVP
Programming algorithm: Auto

Configuration
Fuses:
☑ IESO
☑ FCMEN
☑ Primary clock enable
PLL_EN: Multiplied by 4
FOSC: ExRC, CLKO on OSC2
BORV: 1.9V
BOREN: Enabled (SBOREN disabled)
#PWRTEN: Disabled
WDTEN: WDT enabled, SWDTEN disabled
WDTPS: 1:32768
MCLRE: MCLR
P2BMX: P2B is on RB5
T3CMX: T3CKI is on RC0
HFOFST: Not delayed
CCP3MX: ECP3 multiplexed with RB5
PBADEN: PORTB A/D pins are analog
CCP2MX: CCP2 is multiplexed with RC1
☑ STVREN
LVP: If MCLRE=1 then enabled
XINST: Enabled
DEBUG: RB6 and RB7 are I/O pins
☐ CP0
☐ CP1
☐ CPB
☐ CPD
☐ WRT0
☐ WRT1
☐ WRTC
☐ WRTB
☐ WRTD
☐ EBTR0
☐ EBTR1
☐ EBTRB
ID: FF FF FF FF FF FF FF FF

ASIX UP

8.2.3. IQRF IDE

IQRF provides an IDE for programming and debug their devices.



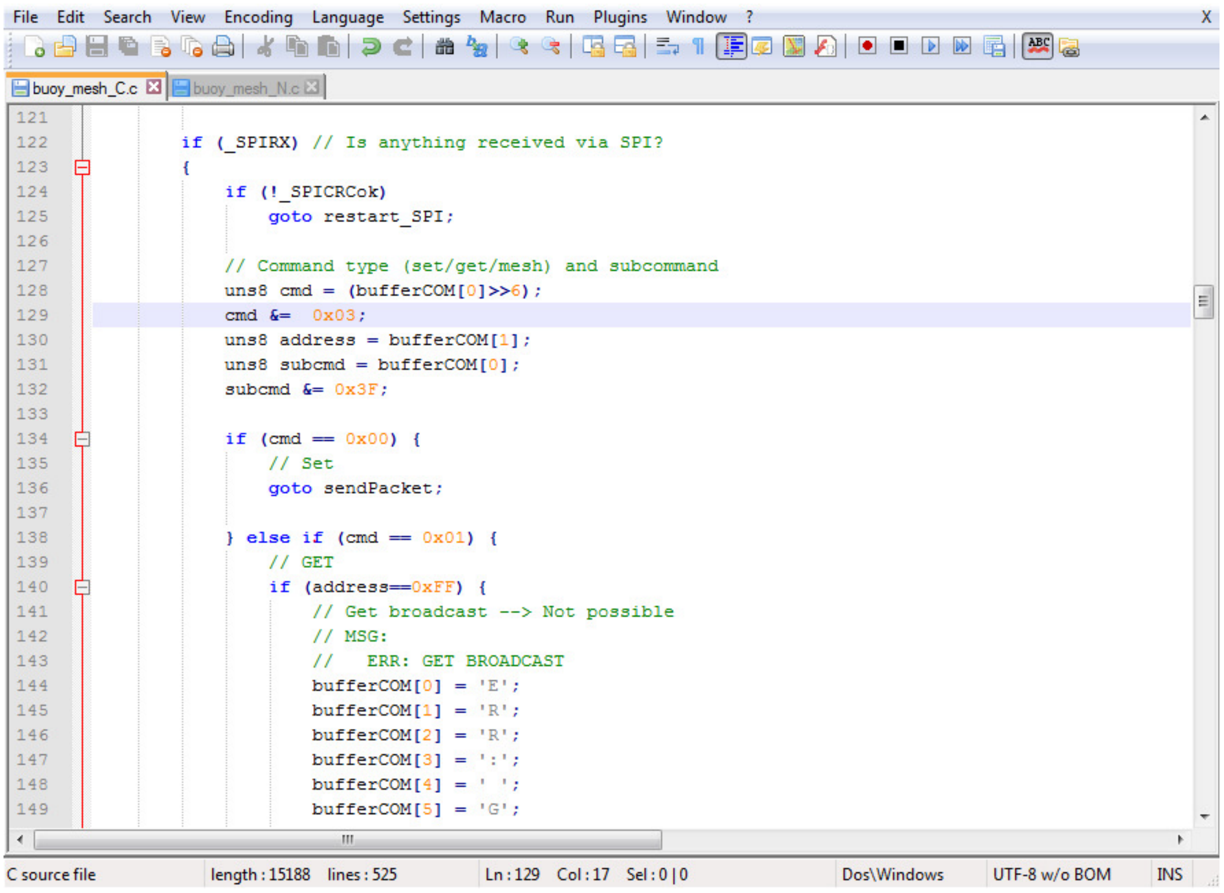
Data to send: $80$00 — Auto Repeat 10 x 100ms

```
Line  Rx/Tx  Length  Data ASCII
   1  Rx     15      BUOY MESH COORD      42.55.4F.59.20.4D.45.
   2  Tx     2       $80$00               80.00.
   3  Rx     10      OK:#3 $3F4B          4F.4B.3A.23.03.20.24.
   4  Rx     15      BUOY MESH COORD      42.55.4F.59.20.4D.45.
   5  Tx     2       $80$00               80.00.
   6  Tx     2       $80$00               80.00.
```

Working with IQRF IDE

Debugging is not simulated, what means that is possible to analyse real scenarios. This interfaces is very intuitive and useful, it can send and receive messages through SPI to a IQRF device connected via microUSB.

To program the code the chosen editor was Notepad++ which is very friendly for IQRF IDE. If an error is found during the compilation, just clicking in the line in IQRF you are redirected to the file and line in Notepad++.


File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
buoy_mesh_C.c
buoy_mesh_N.c
C source file
length : 15188 lines : 525
Ln : 129 Col : 17 Sel : 0 | 0
Dos\Windows
UTF-8 w/o BOM
INS

```

        if (_SPIRX) // Is anything received via SPI?
        {
            if (!_SPICRCok)
                goto restart_SPI;

            // Command type (set/get/mesh) and subcommand
            uns8 cmd = (bufferCOM[0]>>6);
            cmd &=  0x03;
            uns8 address = bufferCOM[1];
            uns8 subcmd = bufferCOM[0];
            subcmd &= 0x3F;

            if (cmd == 0x00) {
                // Set
                goto sendPacket;

            } else if (cmd == 0x01) {
                // GET
                if (address==0xFF) {
                    // Get broadcast --> Not possible
                    // MSG:
                    //   ERR: GET BROADCAST
                    bufferCOM[0] = 'E';
                    bufferCOM[1] = 'R';
                    bufferCOM[2] = 'R';
                    bufferCOM[3] = ':';
                    bufferCOM[4] = ' ';
                    bufferCOM[5] = 'G';
```

Working with Notepad++

8.3. Code IQRF

The code of the IQRF is mainly related with the communication.

8.3.1. Coordinator

The coordinator basically treat and analyses the information. If there are wrong commands (for example, make a get in broadcast) it just shows and error and does not send anything in order to avoid send invalid commands. If the command is valid, then it is send to the mesh.

8.3.2. Node

The node receive orders and execute them. If the command is something that the own IQRF module can do (for example, check the battery status or something related with the mesh level) it is just done. If the node is not its task, then the command is forward to the PIC microcontroller.

8.3.3. Repeaters

Repeaters are just a *dumb* version of the nodes and their only task is just to resend forward the messages to the rest of the mesh.

8.4. Code PIC

The PIC microcontroller has to obey the orders received by the IQRF module and obey them, and answer if it is needed. In addition, It has to store and update the information of the GPS every time is gotten. If an information of any rider is gotten, then it has to send it through the mesh according the fixed criteria.

9. Results

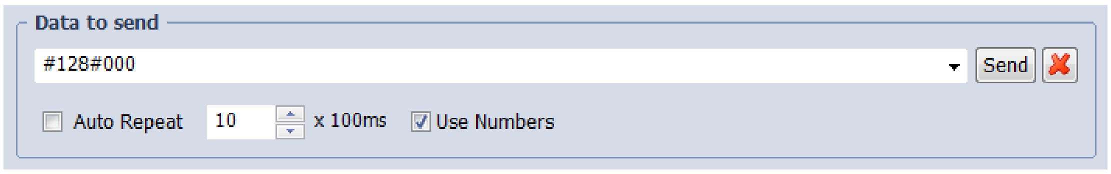
9.1. How it works

We have one coordinator and many nodes of the network. We will assume that all the devices are already programmed and the coordinator is connected to the computer. The first step is bonding all the nodes to create a non-structured mesh. To do this we will use the command bond with the address 000 in order to assign addresses sequentially (autobonding).

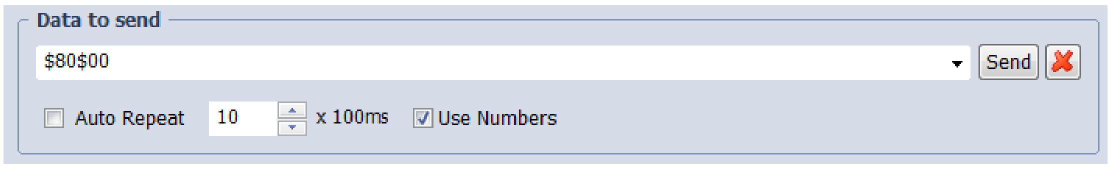
The bonding string will be:

- Command: $1000\ 0000_2 = 80_{16} = 128_{10}$
- Address: $0000\ 0000_2 = 00_{16} = 000_{10}$

So, in the IQRF IDE we could send (both are valid and the same):

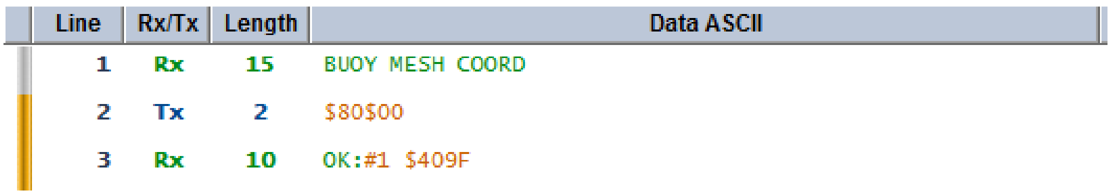

Data to send
#128#000
Send
Auto Repeat 10 x 100ms ✓ Use Numbers

or:


Data to send
$80$00
Send
Auto Repeat 10 x 100ms ✓ Use Numbers

It is easier to transform from binary to hexadecimal, so we will work in hexadecimal from now on.

When the command is sent, then the LED of the coordinator will start to blink. While this time, we have to press the button of the node which we want to bond. If everything is done correctly we will see:

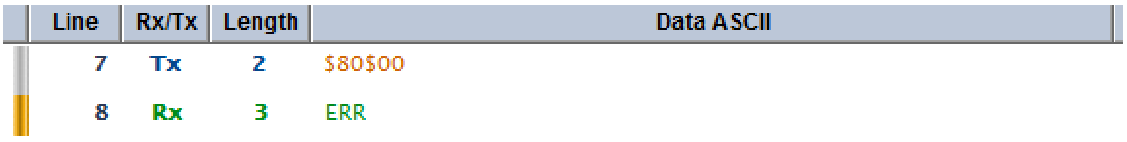


Line	Rx/Tx	Length	Data ASCII
1	Rx	15	BUOY MESH COORD
2	Tx	2	$80$00
3	Rx	10	OK:#1 $409F

Those lines means:

1. Welcome message
2. Autobond
3. Bond correct. Bond address = #1, node id = $409F

If an error happens, then we would see:



Line	Rx/Tx	Length	Data ASCII
7	Tx	2	$80$00
8	Rx	3	ERR

In this way all the nodes (and repeaters) must be bonded to make all of them part of the mesh. Once the mesh is done, it is time to place every buoy in their place and run the "discovery" command in order to make the topology:

- Command: $10\ 000011_2 = 83_{16} = 131_{10}$

It is done. The system is created and working. To send the rest of commands would be in the same way.

If we press the IQRF's button during 2 seconds it will sleep. If this button is pressed during 5 seconds, it will unbond.

10. Glossary

Baud rate: unit used for symbol (symbols per second) or modulation rate (pulses per second. In our case, we work with digital systems, so: $1\,Bd = 1\ {}^{bit}/_{second}$

Jetsurf: is a high-tech motorized surf-board made by MSR Engines. It can reach speeds about 50 km/h.

normalized degree [nº]: corresponds to the degrees which represents the latitude and longitude multiplied by 10,000 in order to avoid decimals. In this way, the information is sent in binary as a positive integer.

PIC microcontroller: *Peripheral Interface Controller.* It is a small computer on a single integrated chip made by Microchip Technology Inc

GPS: (Global Positioning System) Is basically a device that gives the position over the Earth's surface.

IQRF module: *Intelligent Radio Frequency*. A programmable radio frequency transceiver with UART and SPI interfaces.

PCB (Printed Circuit Board): Board with the circuit printed on itself. It is used as base for physically supporting and wiring surface-mounted and socketed components.

NMEA: *National Marine Electronics Association.* A organization which develops electrical and data specifications for marine electronics.

PMTK NMEA: *Packet MediaTek/ETEK.* Standard message protocol to configure some GPS devices.

GPS NMEA: Standard message protocol of GPS notation.

GPRMC: *Recommended minimum specific GPS/Transit data*. One of the GPS MNEA messages protocol.

UART: *Universal Asynchronous Receiver/Transmitter.* Basic module that allows asynchronous communication. It has its own protocol.

SPI interface: *Serial Peripheral Interface*. It is a bus that allows synchronous serial communication interface. Is commonly used for short distance communication, primarily in embedded systems.

IDE: *Integrated development environment.* It is a software application for developing (programming, debugging and compiling) source code that provides comprehensive facilities to the user.

11. Bibliography

[1] Microchip Technology Inc. - Website

[http://www.microchip.com]

[2] Microchip Technology Inc. - PIC12F629/675 Data Sheet

[3] Microchip Technology Inc. - PIC12(L)F1840 Data Sheet

[4] Microchip Technology Inc. - PIC18(L)F2X/4XK22 Data Sheet

[5] MPLAB® IDE - Quick start guide

[6] ASIX s.r.o. - Website

[http://www.asix.net]

[7] B Knudsen Data - CC5X User's Manual

[8] B Knudsen Data - CC8E User's Manual

[9] MikroElectronika - Website

[http://www.mikroe.com]

[10] MikroElectronika - Website

[http://www.mikroe.com]

[11] Adafruit® - Website

[http://www.adafruit.com]

[12] Adafruit® - Adafruit Ultimate GPS Guide

[13] GlobalTop Technology Inc. - PMTK command packet (Rev. A08)

[14] Microrisc s.r.o. - IQRF Website

[http://www.iqrf.org]

[15] Microrisc s.r.o. - IQRF Quick Start Guide

[16] Microrisc s.r.o. - IQRF OS User's Manual (Version 3.04D for TR-5xD)

[17] Microrisc s.r.o. - IQRF OS User's Guide (Version 3.04D for TR-5xD)

[18] Microrisc s.r.o. - IQRF OS Reference Guide (Version 3.04D for TR-5xD)

[19] Microrisc s.r.o. - CK-USB-04 IQRF Programmer and Debugger - User's Guide (Firmware v1.00)

[20] Microrisc s.r.o. - DK-EVAL-04A IQRF development kit - User's Guide

[21] Microrisc s.r.o. - TR-52D Data Sheet

[22] Microrisc s.r.o. - SPI Implementation in IQRF TR modules

[23] Wikimedia Foundation, Inc - Wikipedia Website [http://www.wikipedia.org]