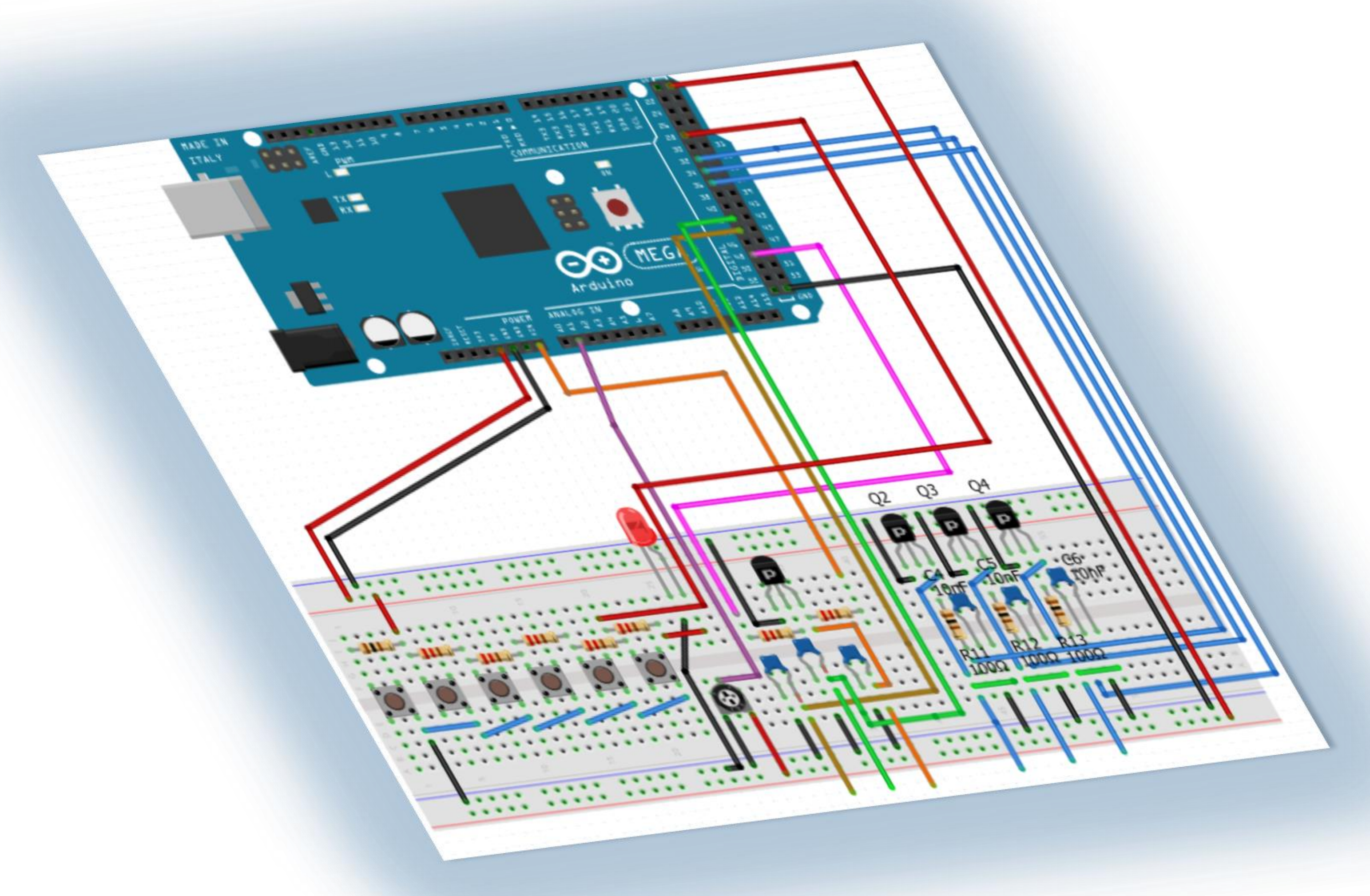
Manual for the K3NG keyer compatible with Arduino MEGA 2560

V 5.1



editor: Cees v.d.Laan, PA3CVI

Eindhoven, March 15 2018

INHOUDSOPGAVE **pag.**

Arduino CW-Keyer

1. Introduction.

An Arduino is a small, relatively inexpensive programmable computer that is incredibly flexible.
An Arduino project is recommended for everyone who is interested in computer programming or in tinkering with electronics. You can choose from a wide range of projects, and I being a radio amateur, ended up choosing the K3NG Arduino CW-Keyer. I have finished this project successfully with great pleasure.
My thanks to Ralph, PA1RB and Anthony Good, K3NG; without their help I would probably not have made it. Finally, I would like to thank Rein, PA0R, for the translation of this manual
Possibly this manual can stimulate working with Arduino projects. it is meant for beginners with some basic knowledge of electronics. You will gradually get acquainted with the handling of microcontrollers and the associated programming. This is a very nice project for hams interested in this type of devices.
K3NG is the designer of the Arduino CW-Keyer, based on the Arduino UNO microcontroller.
For the more advanced, a detailed description of its construction is available on his site.
His programming is of the "open source" type, which offers you the possibility to make your own changes to the program.

https://blog.radioartisan.com/arduino-cw-keyer/

Because I am in possession of the Arduino MEGA 2560, fig.1, I went searching for a proper manual for it. After some effort I finally found a site by KF4BZT which offered me the possibility to start this project.
The difference between the UNO and the MEGA is in a different pin assignment and KF4BZT adapts this based on the basic description by K3NG.

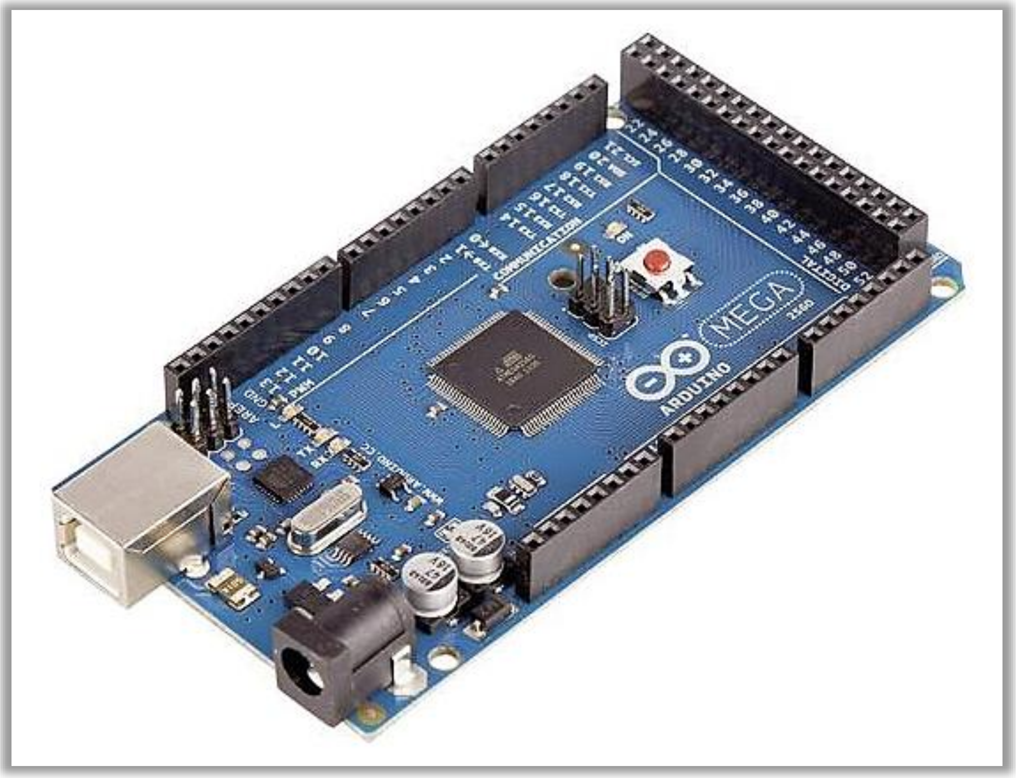


Figure 1, the Arduino MEGA 2560

In the manual of KF4BZT, the CW-Keyer is built up in stages, making use of the Arduino MEGA 2560, laced with many (un)clear pictures.

https://kf4bzt.wordpress.com/2015/08/06/arduino-cw-keyer-project/

Why then this manual you will ask . Well, I being a dummy, knew little about the Arduino phenomenon and I soon got stuck in his description. On the one hand, the description has a comprehensive explanation, on the other hand I miss the beginning of "how to start". Moreover, on his site he is rather sloppy with some circuit diagrams, which contain errors. First of all what does this keyer have to offer:

1.1 Features of the K3NG CW-keyer.

CW speed adjustable from 1 to 999 WPM
Up to six selectable transmitter keying lines
USB or PS2 Keyboard Interface for CW keyboard operation without a computer
Logging and Contest Program Interfacing via K1EL Winkey 1.0 and 2.0 interface protocol emulation
Optional PTT outputs with configurable lead, tail, and hang times
Optional LCD Display – Classic 4 bit mode , Adafruit I2C RGB display or YourDuino I2C LCD Display
Up to 12 memories with macro
Serial numbers
CW keyboard (via a terminal server program like Putty or the Arduino Serial program)
Speed potentiometer (optional – speed also adjustable with commands)
QRSS and HSCW
Beacon / Fox mode
Iambic A and B
Straight key support
Single Paddle
Ultimatic mode
Bug mode
CMOS Super Keyer Iambic B Timing
Paddle reverse
Hellschreiber mode (keyboard sending, memory macro, beacon)
Farnsworth Timing
Adjustable frequency sidetone
Sidetone disable / sidetone high/low output for keying outboard audio oscillator
Command mode for using the paddle to change settings, program memories, etc.
Keying Compensation
Dah to Dit Ratio adjustment
Weighting
Callsign receive practice
Send practice
Memory stacking
“Dead Operator Watchdog”
Autospace
Wordspace Adjustment
Pre-configured and Custom Prosigns
Non-volatile storage of most settings
Modular code design allowing selection of features and easy code modification
Non-English Character Support
CW Receive Decoder
Rotary Encoder Speed Control

Sleep Mode
USB Mouse Support
Alphabet Sending Practice
QLF / “Messy” Straight Key Emulation. (NEW)
USB Keyboard HID (Human Interface Device) Interface (Keyer = keyboard for your computer (NEW))
Web Interface
Training Module

Below are the Digital / PWM / Analog Pins that are available on three Arduino models which are often used:

Arduino Nano Pins.
Digitale I / O Pins 14
PWM Pins 6
Analoge I / O-pins 8

Arduino Yun Pins.
Digitale I / O Pins 20
PWM Pins 7
Analoge I / O-pins 12

Arduino Mega 2560 Pins.
Digitale I / O-pinnen 54
PWM Pins 15
Analoge I / O-pins 16

Arduino UNO and Mega 2560.

The advantage of the Arduino Mega 2560 is that it has 256KB of flash memory (of which 8KB is used by the boot loader). Here you get space for all the possibilities the firmware ofers. Below you can see the Digital / PWM / Analog pins that are available on the Arduino MEGA 2560. With the Mega you can make many more connections, the more connections, the better.
Not all pins will be used for this, since there are ways to link connections with each other via a single line, such as ground and the 5 volt line. Below is a comparison with the Arduino UNO.

Arduino Nano Pins.
Digitale I / O Pins 14
PWM Pins 6
Analoge I / O-pins 8

Arduino Mega 2560 Pins.
Digitale I / O-pinnen 54
PWM Pins 15
Analoge I / O-pins 16

1.2 ARDUINO MEGA 2560 specifications.

Important are the digital and analog pins, and the clock speed (CW Decoding).

Technicalspecificaties:

Microcontroller ATmega2560
Operating voltage 5V
Input voltage (recommended) 7-12V
Input voltage (limit) 6-20V
Digital I/O pins 54 (of which 15 give PWM output)
Analog input pins 16
DC current per I/O pin 20 mA
DC current for 3.3 V pin 50 mA
Flash memory 256 KB of which 8 KB used by bootloader
SRAM 8 KB
EEPROM 4 KB
Clock speed 16 MHz
Length 101.52 mm
Width 53.3 mm
Weight 37 g

As you can see below in fig.2, there is a separate digital section starting with pin 22 until/incl. pin 53.
The analog pin version, bottom left, is labeled with A0 to A15. The power supply is on pin 5Volt with, a.o. two ground (GND) pins.

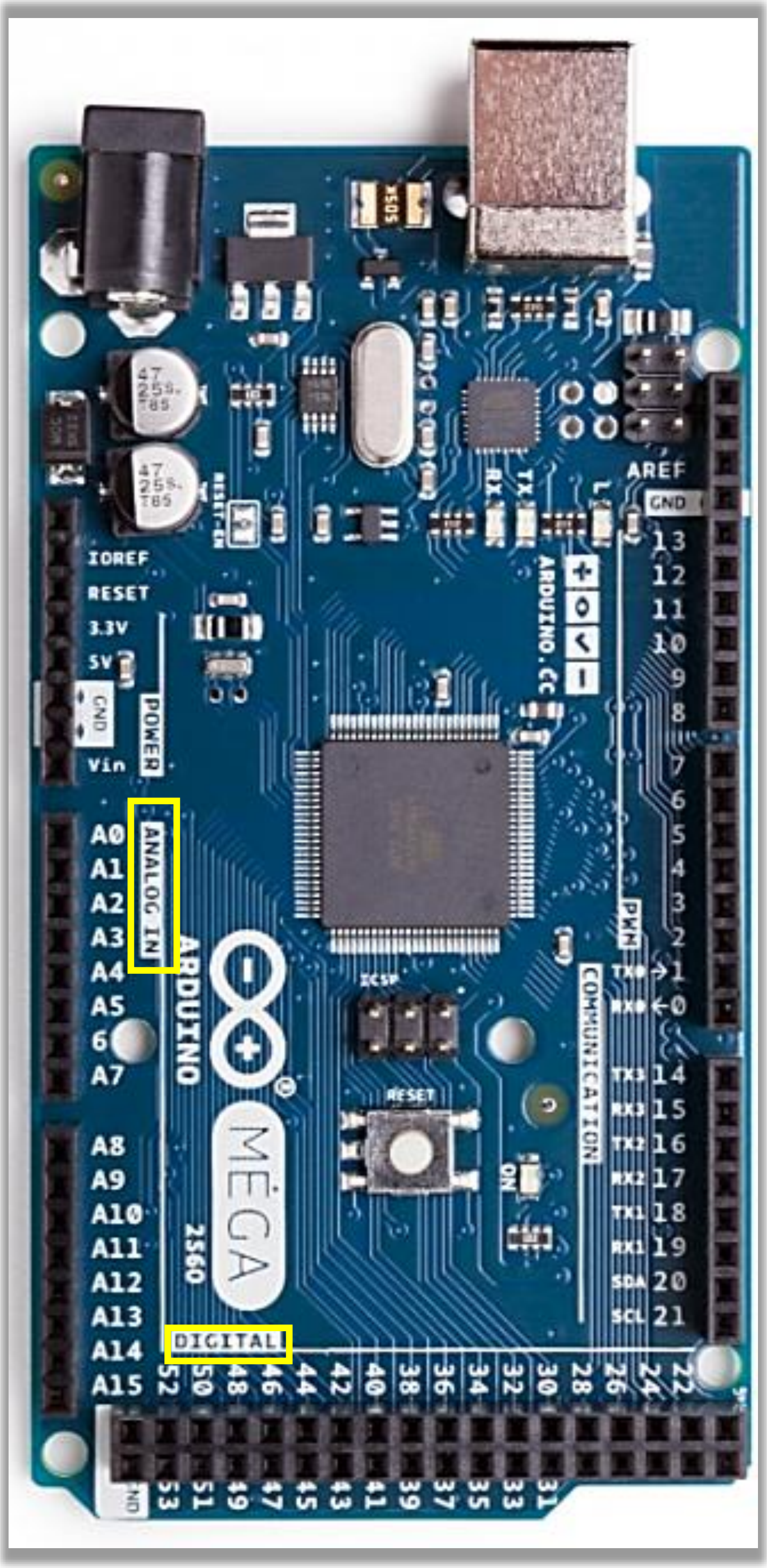


Figure 2, The Arduino MEGA 2560

1.3 Arduino

Based on an input, an Arduino circuit can perform independent actions by providing digital and analogue output signals. From the many types of Arduino microcontrollers in this case you use an Arduino MEGA 2560 microcontroller with a Morse key or a keyboard to control a transceiver. You can also use this to make morse code text visible on a display.
The Arduino Mega 2560, with the Atmega 2560 chip, has a programmable memory of 256 KB and 70 programmable connections.

1.4 The most fundamental parts .

What do you need for this in the first instance:

a. A so-called breadboard, fig.3a

Two breadboards are needed for this project. In Fig. 3b you can see how the connections underneath the breadboard have been made.

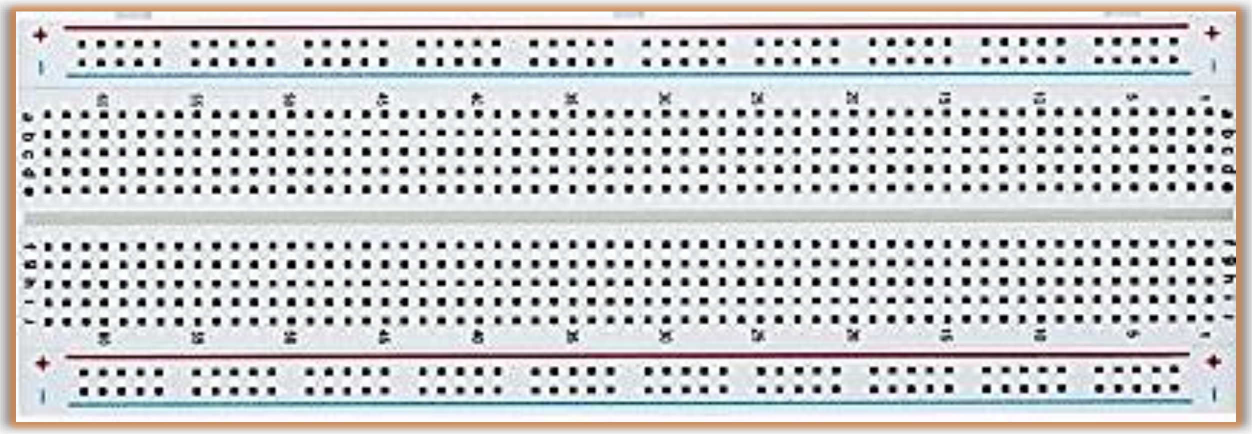


Figure3a, front of breadboard

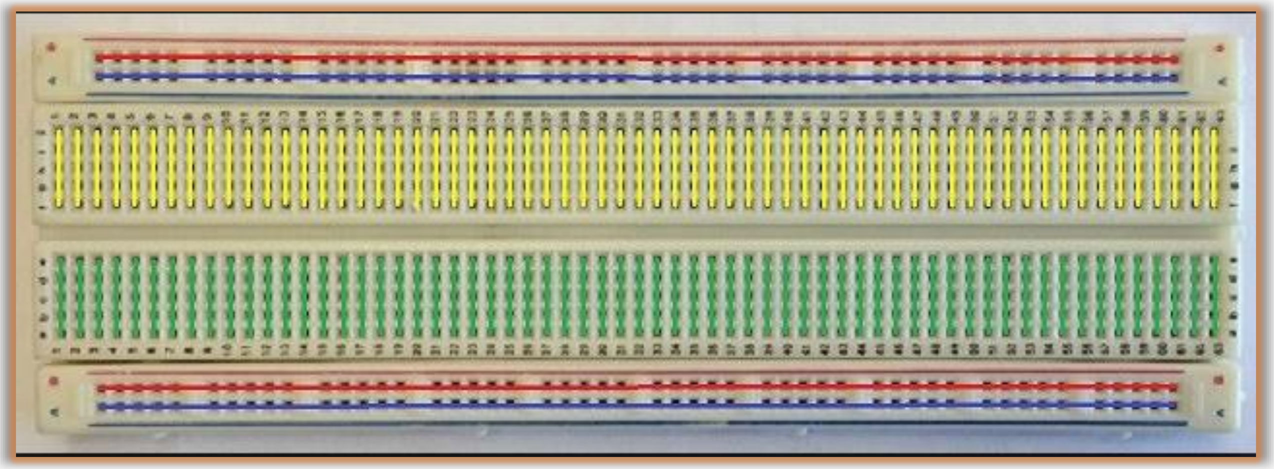


Figure 3b, back of breadboard

b. Jumpers, do not order too few, they are also relatively cheap, fig.4.

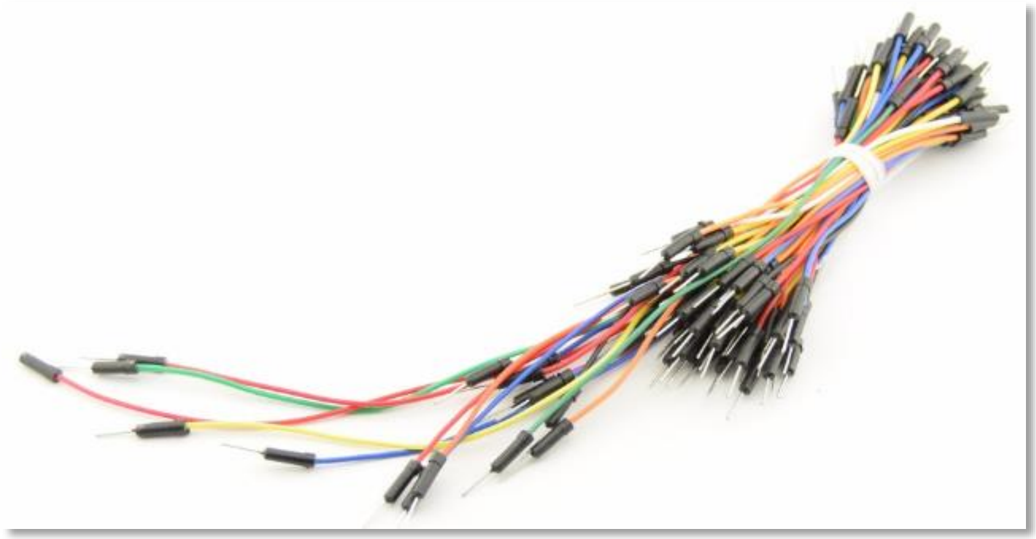


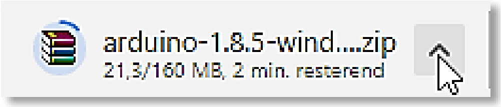
Figure 4, connection wires

c. In addition, of course, all common components that are included in the circuits, see chapter 11, components list.

2 Installing the Arduino IDE 1.8.5

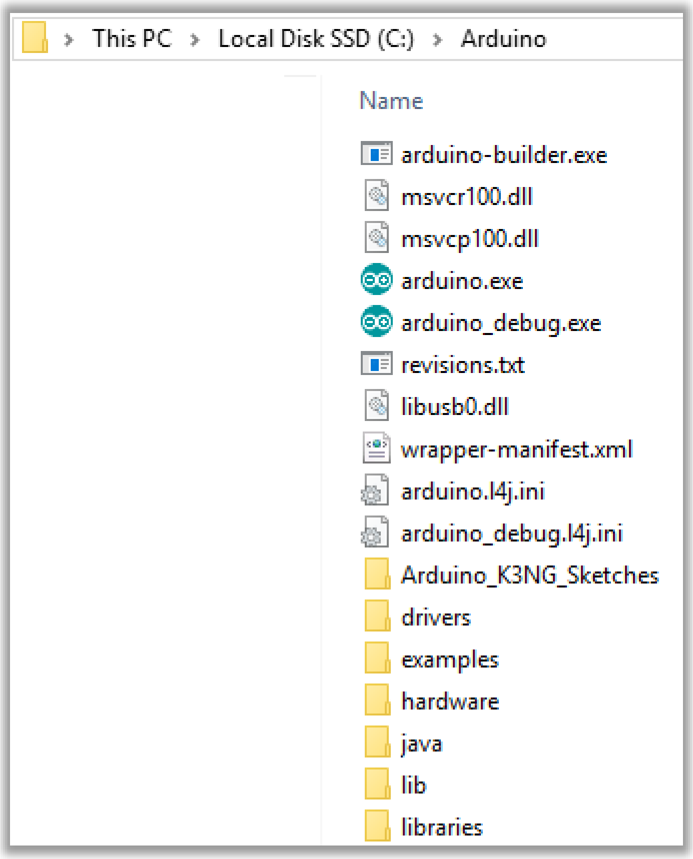
Arduino IDE is a program that allows you to communicate between your PC and the micro-controller and upload the necessary software to the microprocessor. So you download this program first .
Create a file somewhere on your Hard Drive, fig.5, with e.g. the name: 'Temporary download folder K3NG and download the following zip file:

https://www.arduino.cc/en/Main/Software



arduino-1.8.5-wind....zip
21,3/160 MB, 2 min. resterend

Then create a new folder named: *C:/Arduino* and copy the contents of file: *arduino-1.8.5* from the temporary download folder to your Arduino folder, fig.6.



This PC › Local Disk SSD (C:) › Arduino

Name

arduino-builder.exe
msvcr100.dll
msvcp100.dll
arduino.exe
arduino_debug.exe
revisions.txt
libusb0.dll
wrapper-manifest.xml
arduino.l4j.ini
arduino_debug.l4j.ini
Arduino_K3NG_Sketches
drivers
examples
hardware
java
lib
libraries

Figure 6, The contents of arduino-1.8.5

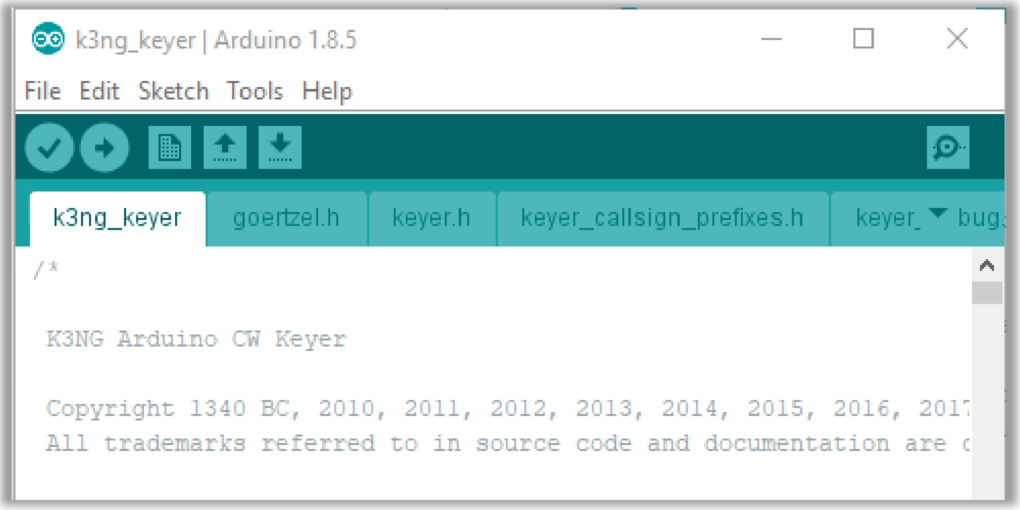
Now start the program with the Arduino.exe file. So far the installation of Arduino IDE.

3 Operating the Arduino IDE program.

It is now time to connect the USB port of your Arduino MEGA 2560 to the USB port of your computer.

Start Arduino IDE and for example a similar Arduino window appears, fig.7.

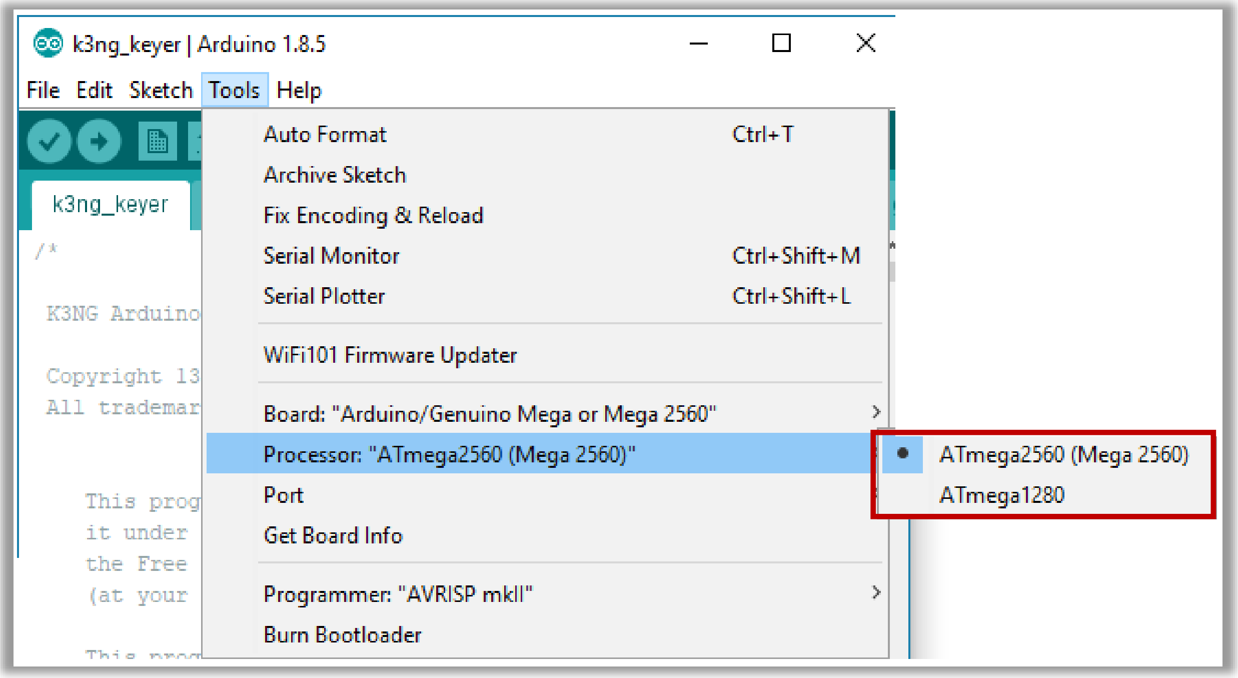
3.1 Arduino IDE configuration


k3ng_keyer | Arduino 1.8.5
File Edit Sketch Tools Help
k3ng_keyer
goertzel.h
keyer.h
keyer_callsign_prefixes.h
keyer_
bug:

```
/*

 K3NG Arduino CW Keyer

 Copyright 1340 BC, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017
 All trademarks referred to in source code and documentation are c
```

Figure 7, Arduino, a possible main window.

First choose the following options, figs. 8 and 9 from ***Tools.*** From Tools, select the Arduino Mega, fig.8



Board: "Arduino/Genuino Mega or Mega 2560"
Processor: "ATmega2560 (Mega 2560)"
Programmer: "AVRISP mkII"

```
/*

 K3NG Arduino

 Copyright 13
 All trademar

    This prog
    it under
    the Free
    (at your

    This prog
```

Figure 8, ATmega2560

The port is usually determined by Windows 10, fig.9. If this is not the case, look under *Device Management*, whether the Arduino is connected to the right USB port, fig.10. You do this via: *Control Panel > Ports,* and setting the correct port number. Of course, in your case, a different port can be assigned as in the example below.

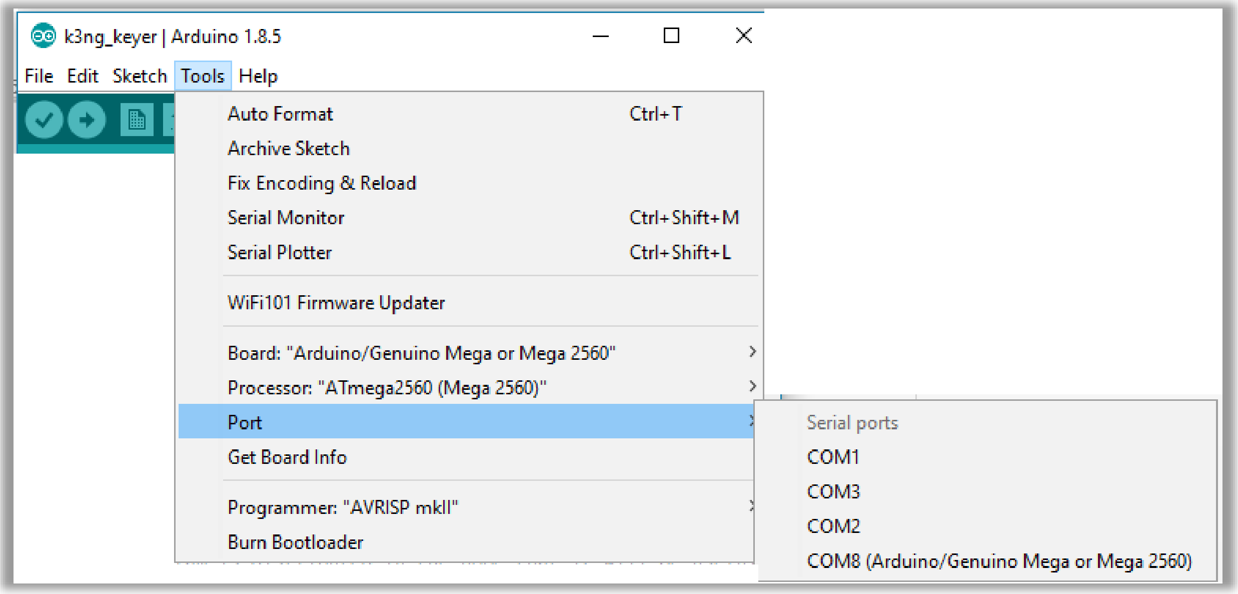

k3ng_keyer | Arduino 1.8.5
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Arduino/Genuino Mega or Mega 2560"
Processor: "ATmega2560 (Mega 2560)"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM1
COM3
COM2
COM8 (Arduino/Genuino Mega or Mega 2560)

Figure 9, assigning a port number

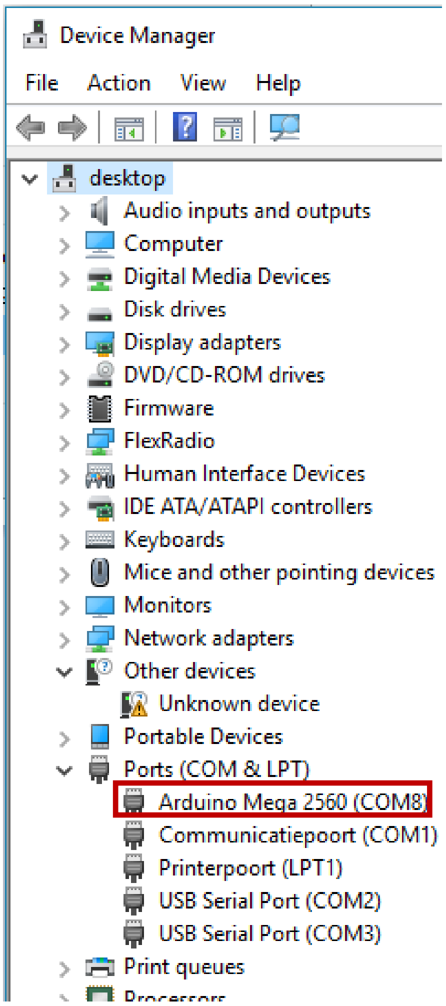


Figure 10, device management

4 Blink sketch.

Before you start with the K3NG sketch (program) it is advisable to start loading the sketch *Blink*. Arduino has provided a number of useful programs for this purpose listed under: *Examples.* With *Blink* you can then check whether the Arduino MEGA functions properly, and it may only take you 10 minutes.
Now disconnect from your PC.

NB. It is not directly required to use the external power supply (7-12 DC Volts), but it offers an advantage with larger circuits and the use of a display, so that the work becomes more stable. The power via your USB connection is then switched off automatically, but the internal power supply of the Arduino remains at 5 Volts.

4.1 Setup.

The figure 11 below shows the simple setup of the Arduino Mega with your breadboard.
Take a good look at how you position the parts. With most LEDs the longer connection wire is the +.

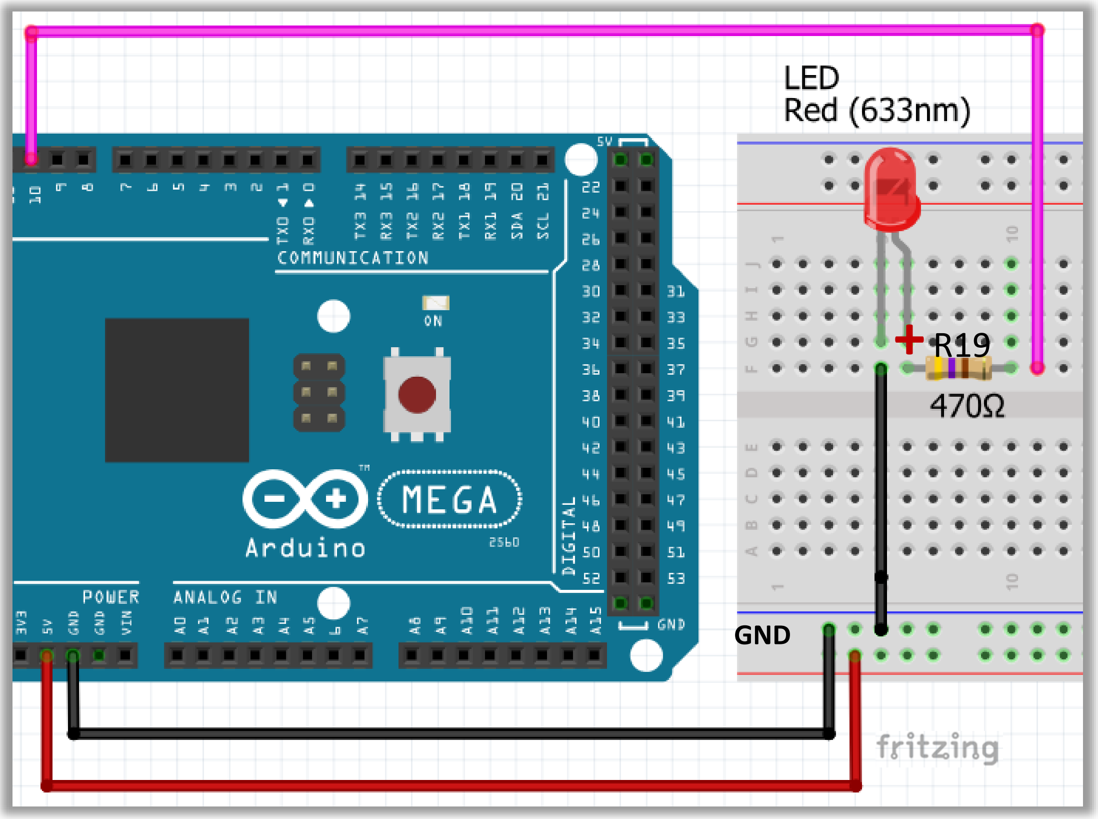


Figure11, Blink setup

☞ *Work too hurriedly.*
You can easily make mistakes when connecting to your breadboard, try concentrating and not to make a mistake. If you do not proceed with caution then you may have to buy your next Arduino MEGA 2560, as happened to me.

4.2 Creating and starting the program.

First of all, this manual is not meant to give you an explanation about the programming language, but with a little bit of effort there is enough info available on the internet to globally understand the programs. In addition, some commands are commented with an explanation.

Start Arduino IDE and load the sketch Blink, fig.12.

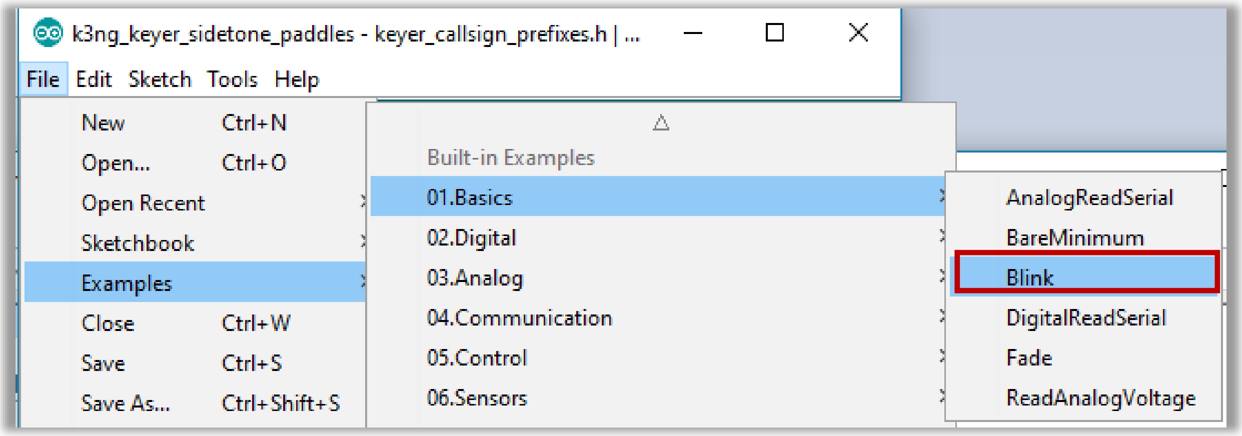


Figure 12, select Blink: Examples > 01.Basics > Blink

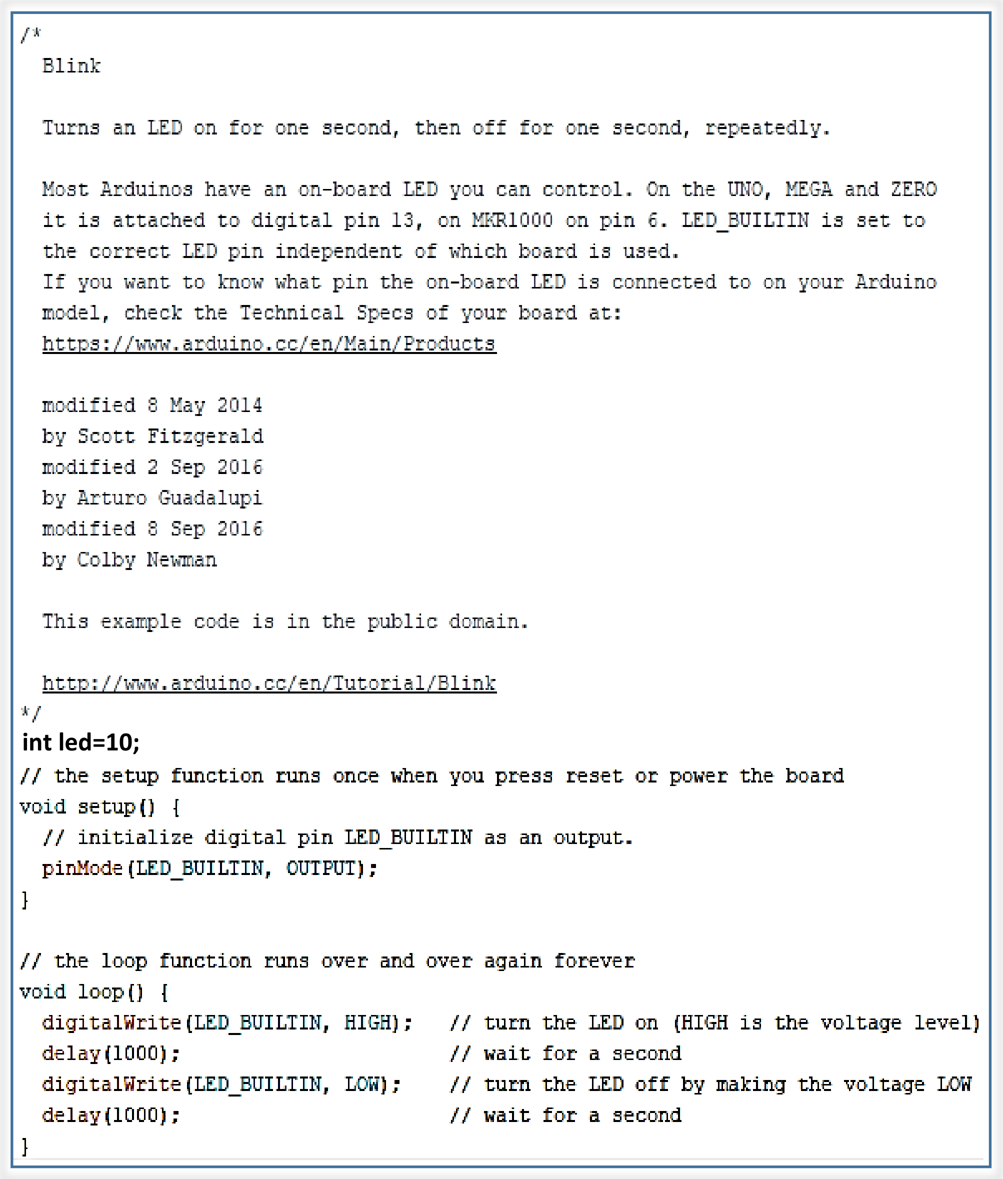
A clear explanation of this program can be found on:

https://www.arduino.cc/en/Tutorial-0007/BlinkingLED

The Blink program now appears in the Arduino IDE window. In the example below, fig. 13, the pin variable is set at 10, according to the arrangement, fig.10.

Everything between: /* …….. */ is taken as text and not compiled. Lines prior to: // are also not compiled.

Add the line: ***int LedPin = 10;***



```
/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/
int led=10;
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Figure 13, the Blink sketch

If everything is right on your breadboard, connect the USB port of your breadboard to the PC.

Click on the **"V"** icon, fig.14, to compile the program. If you have an error message you have created some error in the sketch.

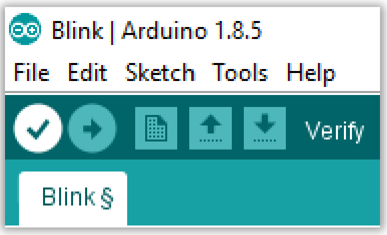


Figure 14, compile.

Then click on the "arrow" icon to upload the sketch to your Arduino, fig.15.

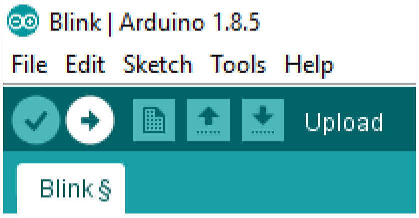


Figure15, upload.

If you do not receive an error message when uploading, the LED will be switched on and off with the time intervals programmed.
You have successfully completed your first experiment. !!

After this you start the "real" project. To reduce the chance of errors, we are using a step-by-step procedure. To save your design programs: go to File > Preferences fig.16 and 16a and save your sketches in a self-named folder.
Do not forget to click **OK**.

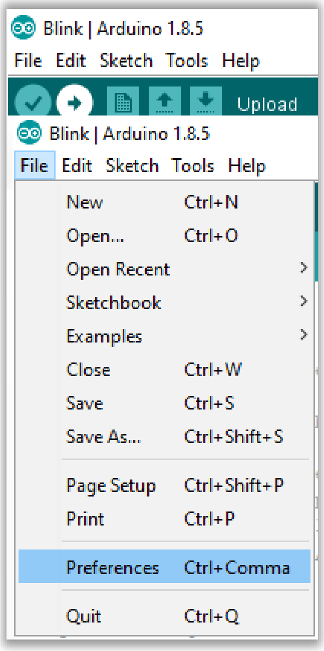
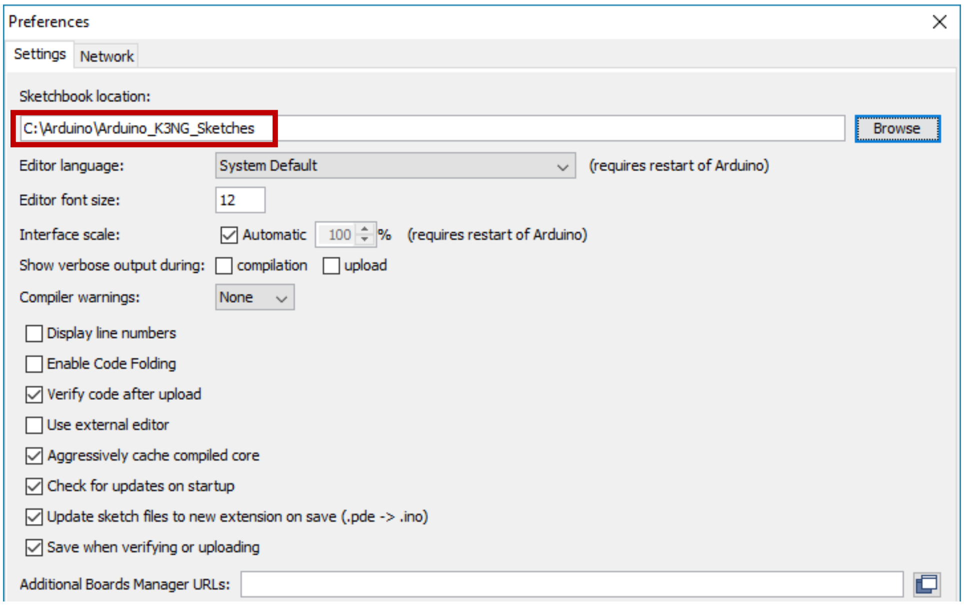


Figure 16, sketch book location.



Sketchbook location: C:\Arduino\Arduino_K3NG_Sketches

Editor language: System Default (requires restart of Arduino)

Editor font size: 12

Interface scale: ☑ Automatic 100 % (requires restart of Arduino)

Show verbose output during: □ compilation □ upload

Compiler warnings: None

□ Display line numbers
□ Enable Code Folding
☑ Verify code after upload
□ Use external editor
☑ Aggressively cache compiled core
☑ Check for updates on startup
☑ Update sketch files to new extension on save (.pde -> .ino)
☑ Save when verifying or uploading

Additional Boards Manager URLs:

Figure 16a, preferences

5 K3NG CW Keyer, UNO

Below is the original diagram, fig.17, of the Arduino K3NG CW keyer based on UNO which can be found under:

https://www.arduino.cc/nl/uploads/Main/arduino-mega2560_R3-sch.pdf

In general terms, this schedule is used in this manual, but with a modified port occupation for the benefit of the MEGA 2560 and a supplement for a display and a PS2 keyboard connection.

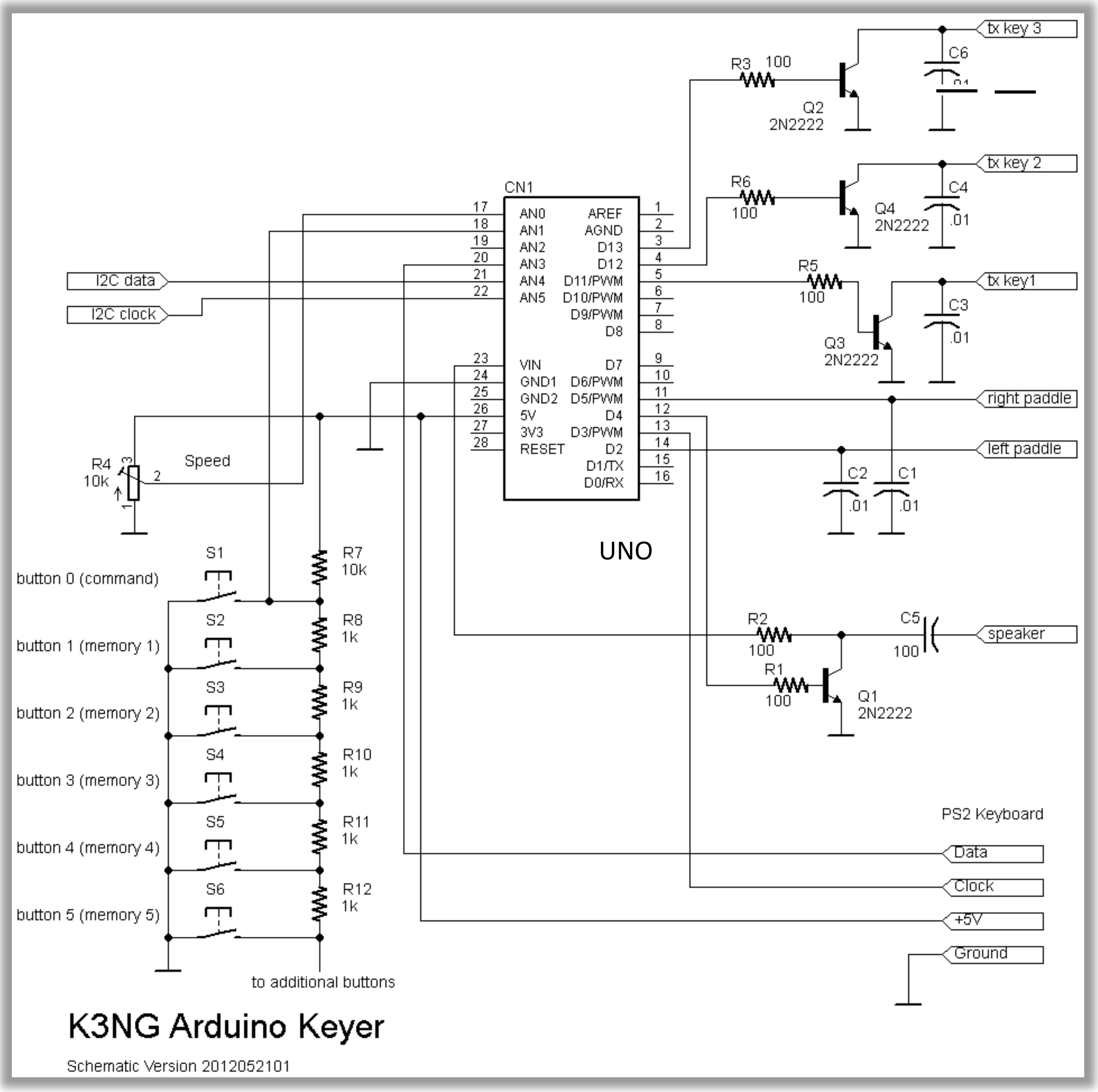


Figure 17, K3NG basic circuit

ısed C's, can be ceramic disk capacitors , so without polarity

6. Organize the K3NG CW keyer files.

The folder structure is very important for the proper functioning of the project and it is a pity that we can find very little information on the internet.
That's why I use an organigram. Fig.18 shows how to set up the Arduino structure. The yellow rectangles show the folders and the blue colored rectangles the corresponding files.
On the next page it describes how to do the folder organization.

Arduino K3NG Keyer organization chart

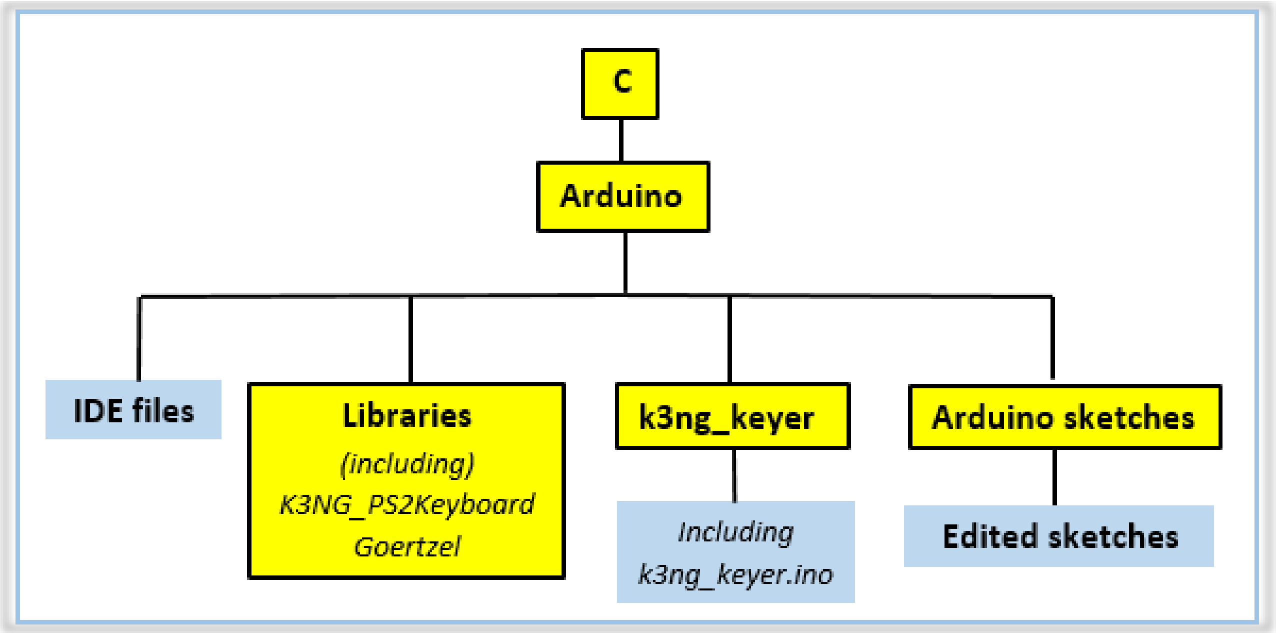
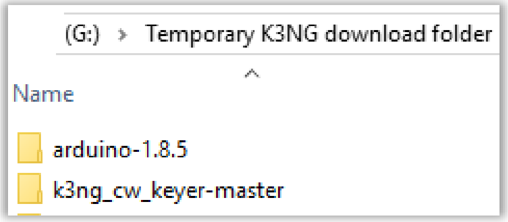


Figure18, Arduino K3NG organization chart

For the K3NG keyer you have to download the zip file below into your *Temporary download folder*. The intermediate step to put it in a temporary folder looks a bit overdone at first, but it offers the advantage of being able to quickly recover all the basic files if necessary

https://github.com/k3ng/k3ng_cw_keyer

k3ng_cw_keyer-ma....zip



(G:) › Temporary K3NG download folder

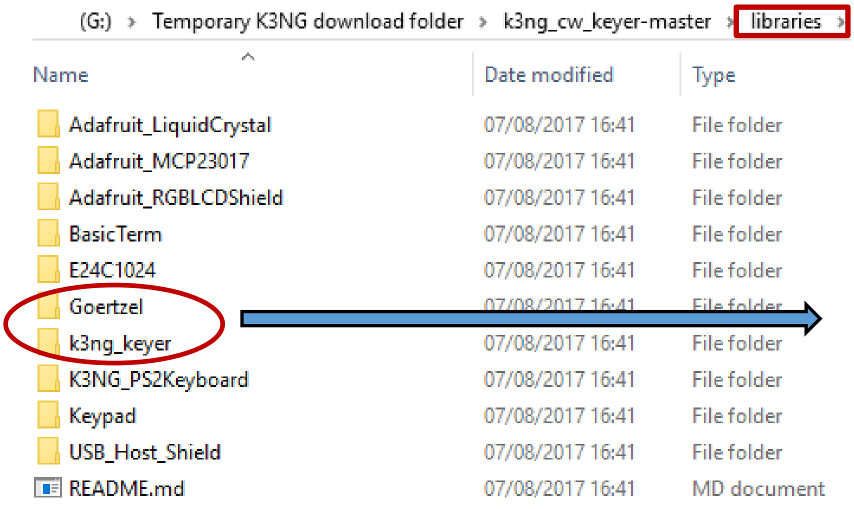
Name
arduino-1.8.5
k3ng_cw_keyer-master

Figure 19, temporary download folder

6.1 Put the files in the right place.

Assuming the Arduino IDE from the Blink project is still in the same location, you do the following, according to the organization chart:
Go to the **libraries** folder via:
temporary download folder > k3ng_cw_keyer-master > libraries

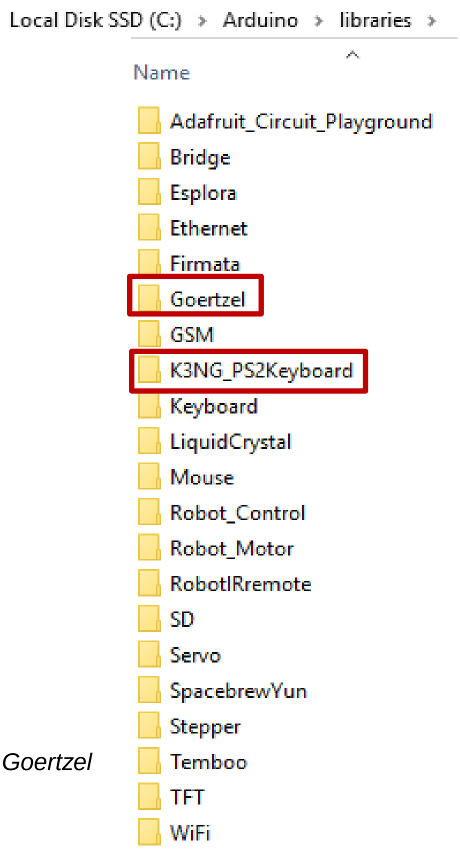
Copy the folder *K3NG_PS2Keyboard* and *Goertzel* from this library, fig.20 and paste it into the folder of the Arduino library, fig.20a.



(G:) › Temporary K3NG download folder › k3ng_cw_keyer-master › libraries ›

Name	Date modified	Type
Adafruit_LiquidCrystal	07/08/2017 16:41	File folder
Adafruit_MCP23017	07/08/2017 16:41	File folder
Adafruit_RGBLCDShield	07/08/2017 16:41	File folder
BasicTerm	07/08/2017 16:41	File folder
E24C1024	07/08/2017 16:41	File folder
Goertzel	07/08/2017 16:41	File folder
k3ng_keyer	07/08/2017 16:41	File folder
K3NG_PS2Keyboard	07/08/2017 16:41	File folder
Keypad	07/08/2017 16:41	File folder
USB_Host_Shield	07/08/2017 16:41	File folder
README.md	07/08/2017 16:41	MD document

Figure 20, libraries folder



Local Disk SSD (C:) › Arduino › libraries ›

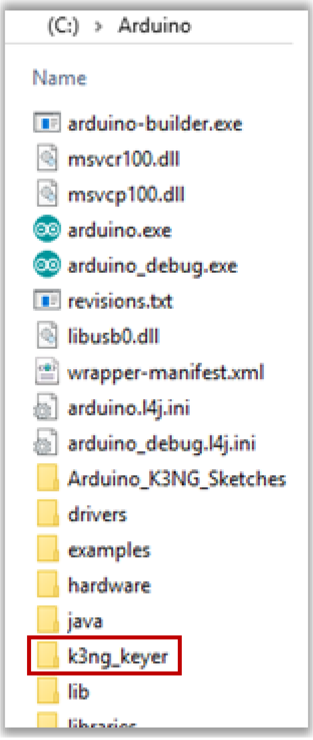
Name
Adafruit_Circuit_Playground
Bridge
Esplora
Ethernet
Firmata
Goertzel
GSM
K3NG_PS2Keyboard
Keyboard
LiquidCrystal
Mouse
Robot_Control
Robot_Motor
RobotIRremote
SD
Servo
SpacebrewYun
Stepper
Temboo
TFT
WiFi

Figure 20a, the folders of K3NG_PS2Keyboard en Goertzel

To clarify, the PS2 keyboard folder is added to the library because you will at a later stage connect your keyboard and Goertzel to decode Morse on the display. This way Arduino IDE can find and process the necessary file. (Thanks for the help of Fred, VK2EFL).

Create a new folder with the name: ***k3ng_keyer*** under *C: > Arduino*

If you start building the K3NG keyer program, in the next chapter, you will need the file *k3ng_keyer.*
Go to: *Tijdelijke K3NG map > k3ng_cw_keyer-master > libraries >* ***k3ng_keyer***
Copy the contents from **k3ng_keyer** naar *C: > Arduino >* ***k3ng_keyer,*** fig.21.
The folder and file layout is now complete and you can now finally start with the Arduino K3NG Keyer project itself.



(C:) › Arduino

Name

arduino-builder.exe
msvcr100.dll
msvcp100.dll
arduino.exe
arduino_debug.exe
revisions.txt
libusb0.dll
wrapper-manifest.xml
arduino.l4j.ini
arduino_debug.l4j.ini
Arduino_K3NG_Sketches
drivers
examples
hardware
java
k3ng_keyer
lib
libraries

Figure 21, k3ng_keyer

7. The K3NG Arduino MEGA 2560 hardware.

7.1 Start with the components of the Arduino K3NGkeyer

Always disconnect the USB connection from your PC before working on your breadboard.

You can finally start with the physical construction of the keyer. The breadboard I use corresponds to the one shown on page 7, fig.3.

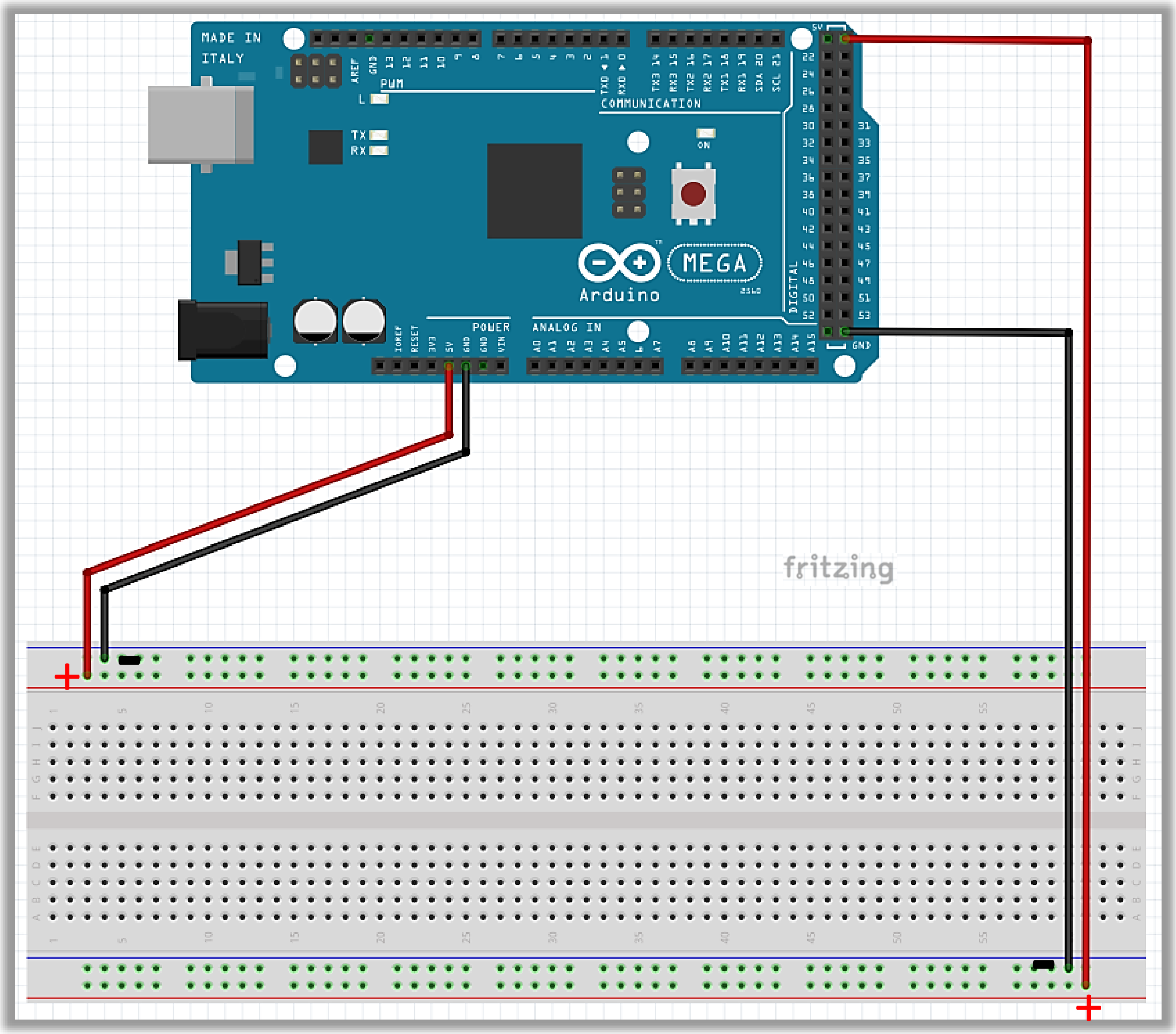


Figure 22, basic breadboard parts with power lines from the Arduino MEGA

8. The Roadmap.

Step 8.1: Adding the buttons.

As a first step in this project you place the push buttons as much as possible on the left side of the board, fig. 24. There are six of them, the most left is the ***Command*** button for programming.
The other push buttons are ***Memory*** buttons. later on you can attach multiple memory buttons if necessary.
The buttons are simple on-off switches, connected according to the diagram, fig.23.
.

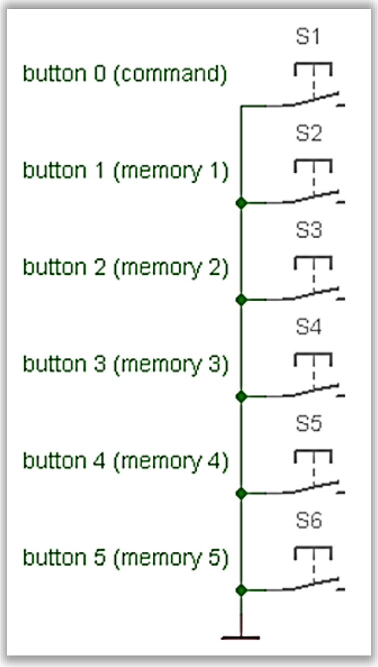


Figure 23, diagram with buttons.

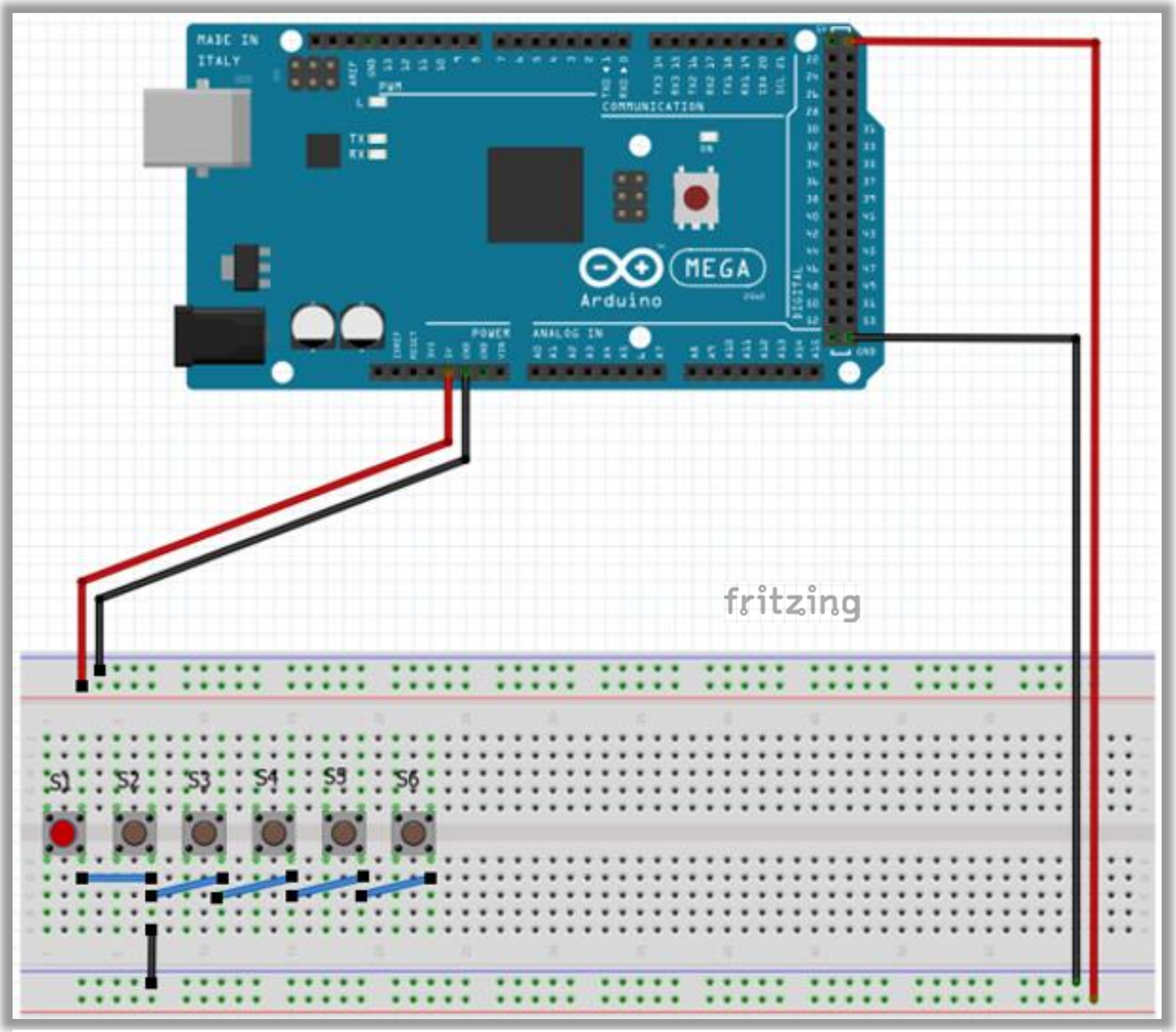


Figure 24, the push buttons

Step 8.2: Adding the resistors.

Place the 6 resistors on the board, fig. 26 and pay attention to the position of the resistors, it is easy to make a mistake. Fig. 25 shows the circuit again .

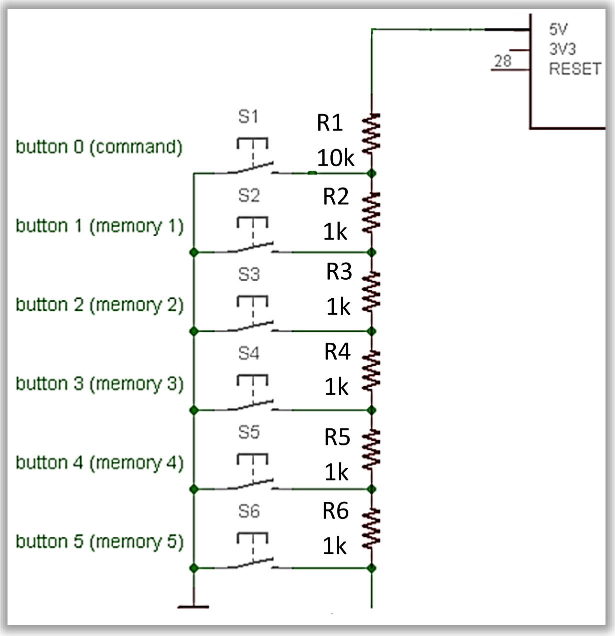


Figure 25, six resistors.

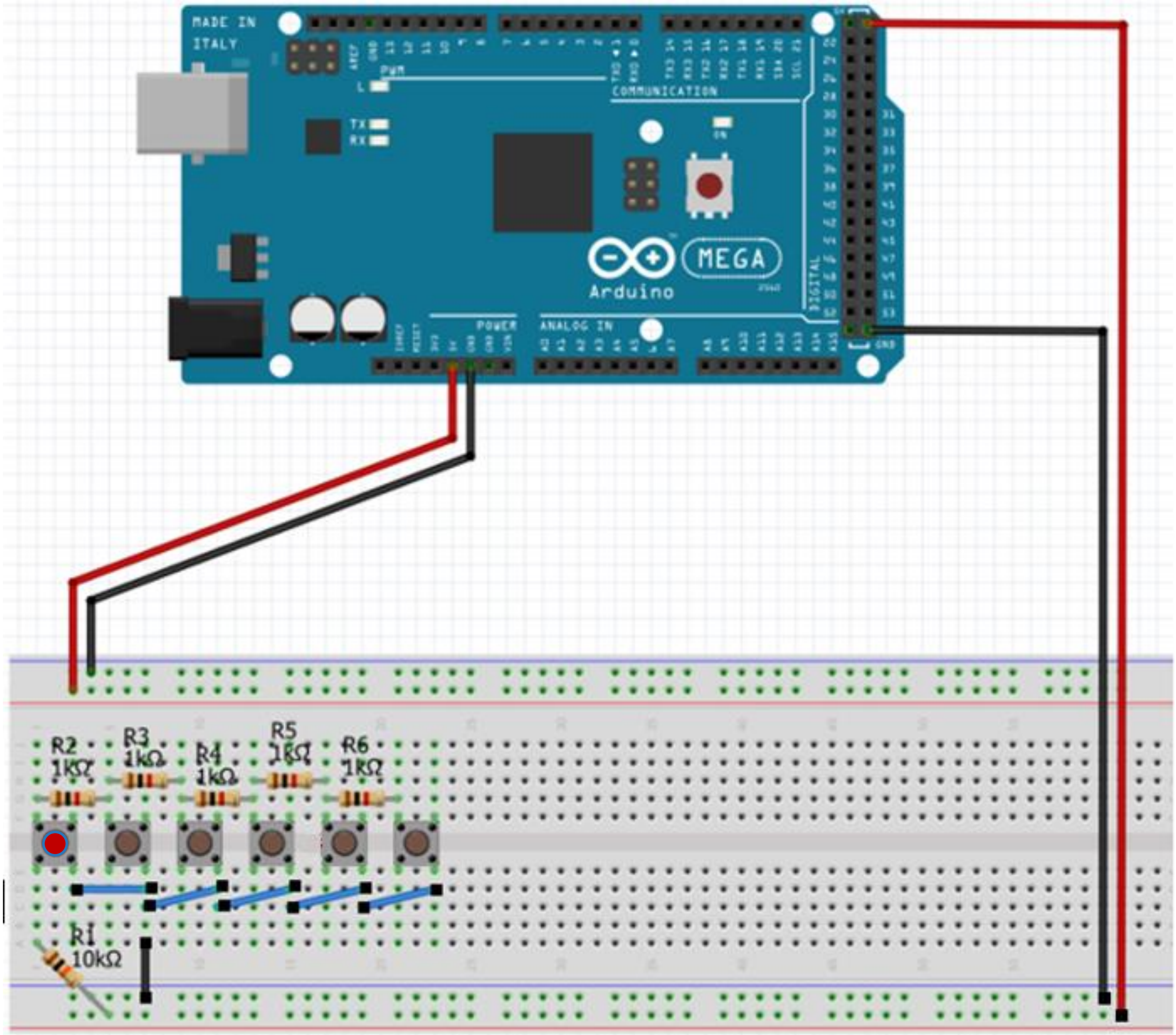


Figure 26, buttons with resistors.

Step 8.3: Adding a ledLED.

This step can also be found on the K3NG website, and provides an opportunity for checking if the Command button has been pressed, taken into account in the following steps, figs. 27 and 28.

To check whether the Command button has been pressed, a LED is added. Briefly press the Command button (digital port D28 becomes "high", approx. 5 Volts) the light comes on, briefly pressing again the LED goes off. Pay attention to how the LED is connected, the longest connection wire is the + and is connected to the 470 Ohm resistor.

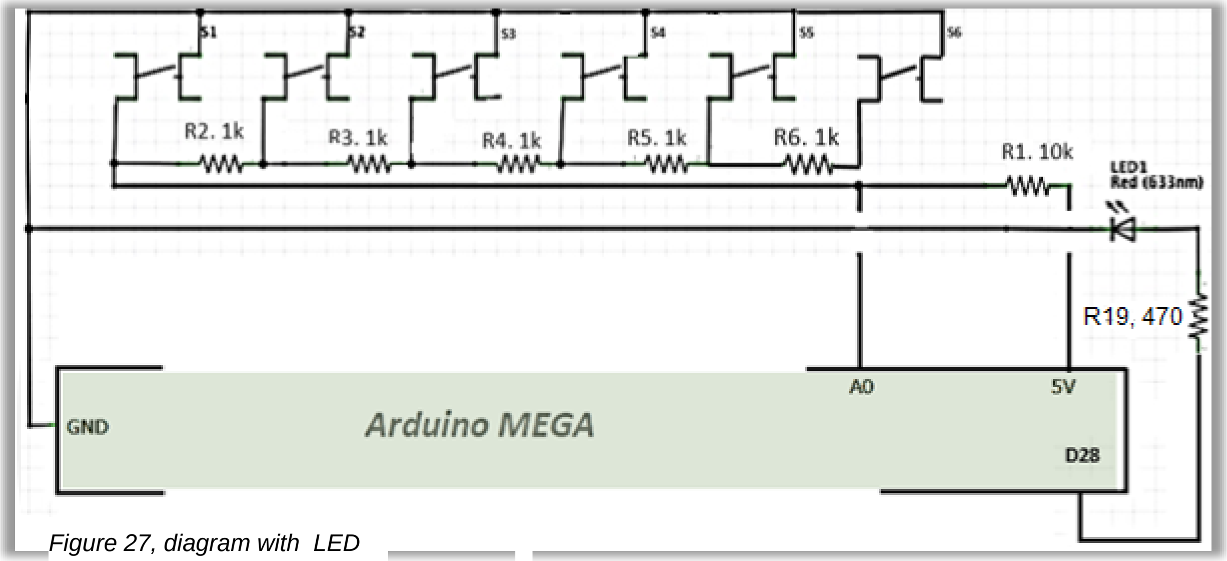


Figure 27, diagram with LED

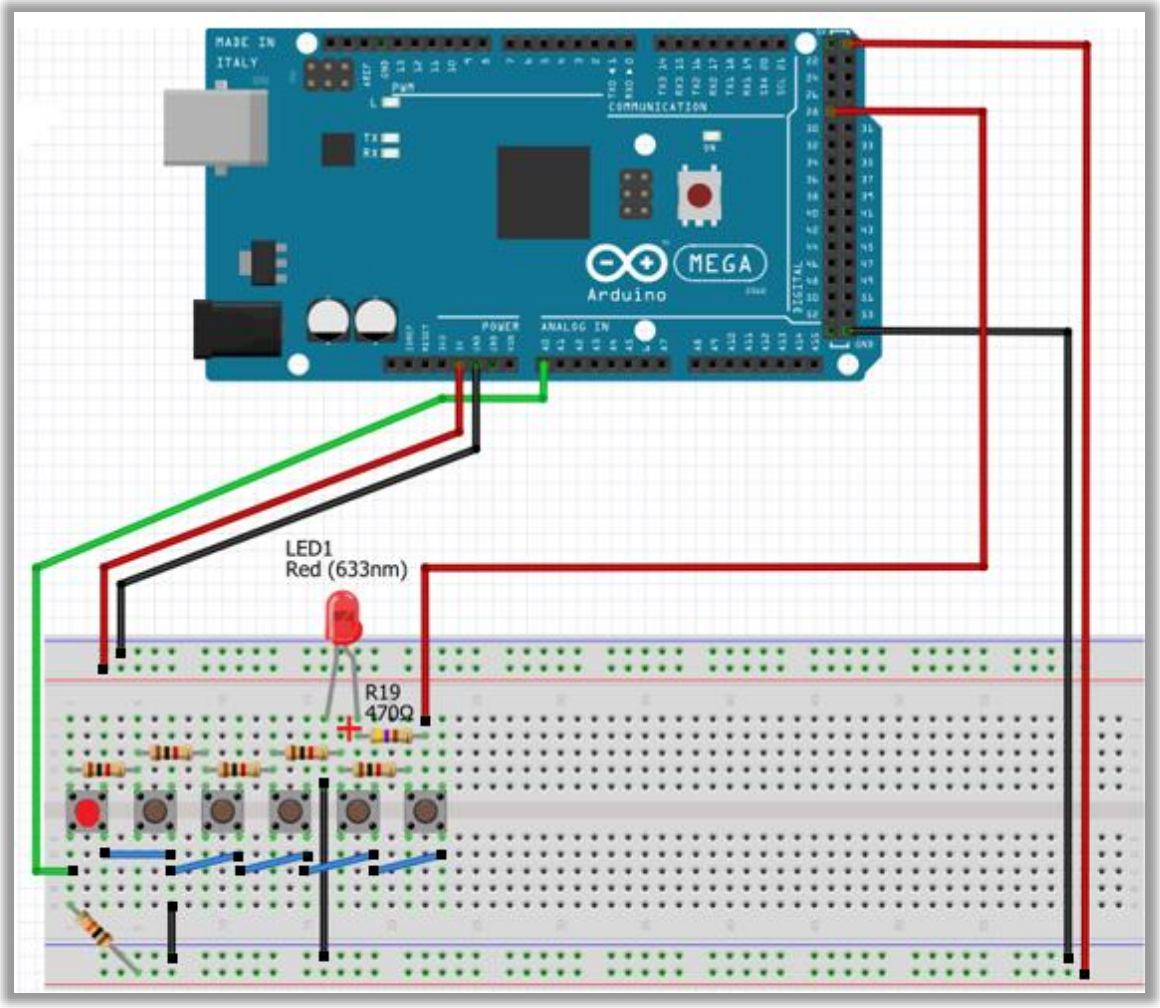
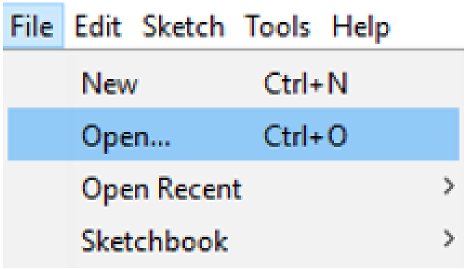


Figure 28, arrrangement

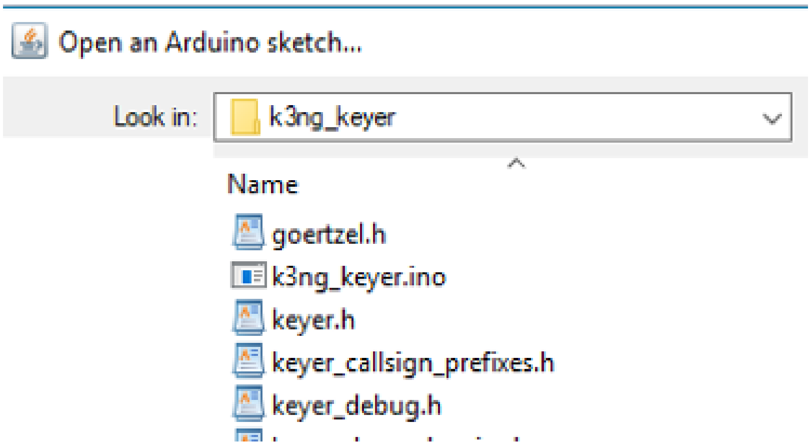
Stap 8.4: Load the data from the K3NG keyer into Arduino IDE

Open het Arduino IDE programma.

To open the basic K3NG program in Arduino IDE go to ***Open*** and click on the ***k3ng_keyer.ino*** file in the folder: ***SSD (C:)* > *Arduino* > *k3ngkeyer*,** *fig. 29 en 29a.*
In there you will find all the necessary ***h.*** files needed to build the keyer.



Figuur 29, open k3ng_keyer.ino

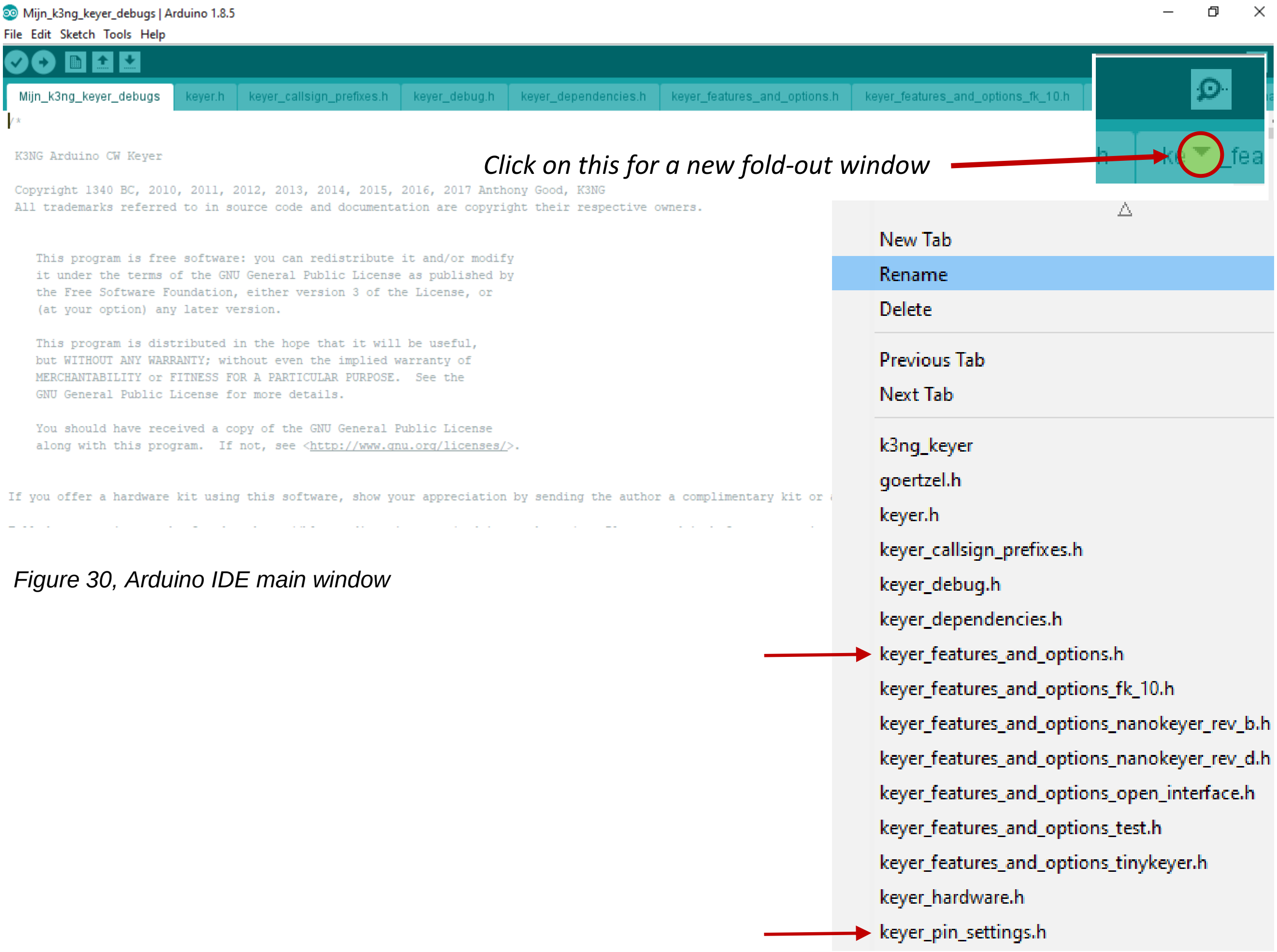


Look in: k3ng_keyer

Figure 29a, basic k3ng_keyer.ino file

The required horizontal green bar of the main window now shows all required **.h** positions, which you can see more clearly when you open the hidden fold-out window, fig.30.

You now start from the basic K3NG configuration and then make the necessary modifications. The function of the so-called .h files will be clearly explained in the remainder of the manual.


Mijn_k3ng_keyer_debugs | Arduino 1.8.5
File Edit Sketch Tools Help
Mijn_k3ng_keyer_debugs
keyer.h
keyer_callsign_prefixes.h
keyer_debug.h
keyer_dependencies.h
keyer_features_and_options.h
keyer_features_and_options_fk_10.h
New Tab
Rename
Delete
Previous Tab
Next Tab
k3ng_keyer
goertzel.h
keyer.h
keyer_callsign_prefixes.h
keyer_debug.h
keyer_dependencies.h
keyer_features_and_options.h
keyer_features_and_options_fk_10.h
keyer_features_and_options_nanokeyer_rev_b.h
keyer_features_and_options_nanokeyer_rev_d.h
keyer_features_and_options_open_interface.h
keyer_features_and_options_test.h
keyer_features_and_options_tinykeyer.h
keyer_hardware.h
keyer_pin_settings.h

```
/*

 K3NG Arduino CW Keyer

 Copyright 1340 BC, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017 Anthony Good, K3NG
 All trademarks referred to in source code and documentation are copyright their respective owners.


    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.


If you offer a hardware kit using this software, show your appreciation by sending the author a complimentary kit or
```

Click on this for a new fold-out window

Figure 30, Arduino IDE main window

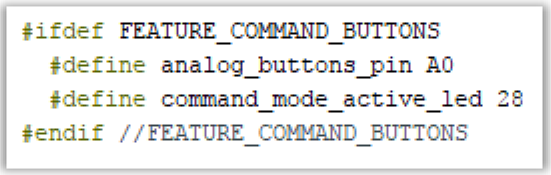
You are going to use the two .h files below to adjust things in the sketch.

a. ***keyer_pin_settings.h***
   To fix the pin occupation.

b. ***keyer_features_and_options.h***
   To switch certain options from the K3NG program on or off.

Step 8.5: Adjust the basic K3NG sketch.

keyer_pin_settings.h, `where you define the pin setting`



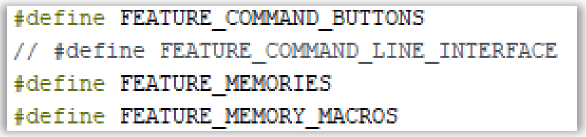
```
#ifdef FEATURE_COMMAND_BUTTONS
  #define analog_buttons_pin A0
  #define command_mode_active_led 28
#endif //FEATURE_COMMAND_BUTTONS
```

Figure 31, pin settings

The analogue pin has been set to: A0. (Currently *Command_mode_active_led* is set to an in-active state). The digital pin is for port 28, fig.31, which switches the LED.

At this stage you have to activate three commands, fig.32 and you do that by removing the **//**.
Do not forget that, otherwise the program will not work .

keyer_features_and_options.h.



```
#define FEATURE_COMMAND_BUTTONS
// #define FEATURE_COMMAND_LINE_INTERFACE
#define FEATURE_MEMORIES
#define FEATURE_MEMORY_MACROS
```

Figure 32, activate functions.

☞ With “*Edit > Search…*” you can quickly find a command in the program.

Once you have made the changes to the codes, you can compile the sketch and, after connecting your Arduino MEGA, upload to the Arduino Mega for testing. Here also applies: are you sure that all components are correctly connected ?
Do not forget to compile and update the program. After pressing the Command button, the LED should light up and pressing this button again will switch it off.

Step 8.6: Expand with potentiometer.

Below is the diagram, fig.33 and breadboard setup, fig.34, with the added potentiometer, which you can use later on to set the signal speed from 1 - 999 words per minute (WPM).

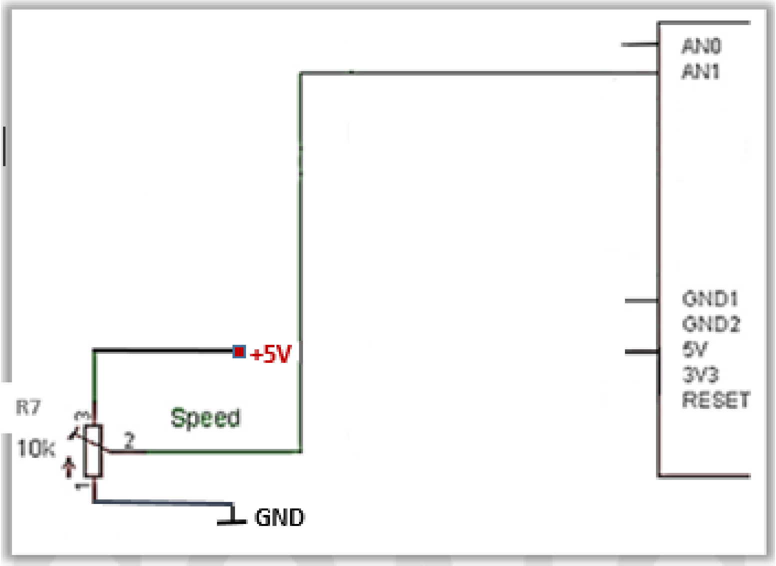


Figure33, diagram with potentiometer.

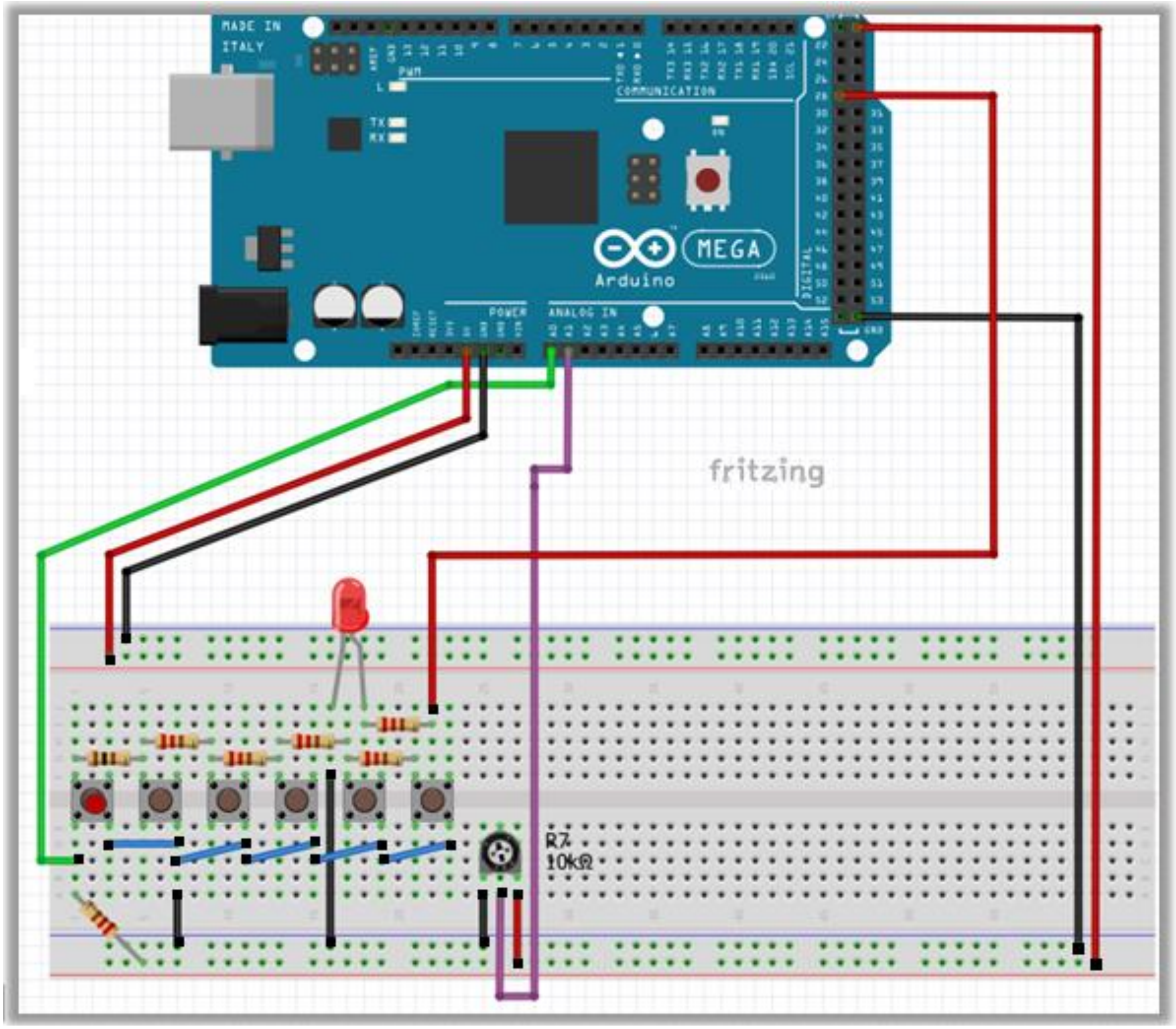
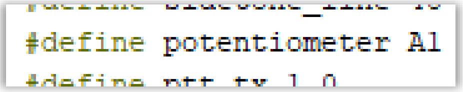


Figure 34, breadboard arrangement with potentiometer.

To make the potentiometer work, you have to do the following:

Select ***keyer_pin_settings.h*****,** which determines the pin setting, because the analog port A0 is already occupied, choose port A1, fig.35.

keyer_pin_settings.h

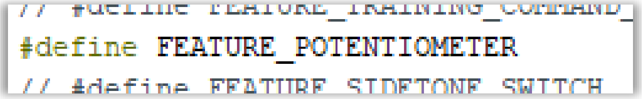


```
#define potentiometer A1
#define ptt_tx_1 0
```

Figure 35, Analog port A1

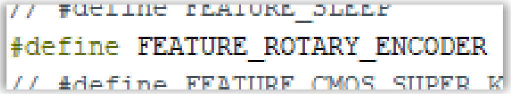
In the ***keyer_features_and_options.h*** file, activate the following command (remove the //) fig. 36/37:

keyer_features_and_options.h



```
// #define FEATURE_TRAINING_COMMAND_
#define FEATURE_POTENTIOMETER
// #define FEATURE_SIDETONE_SWITCH
```

Figure 36, Potentiometer option



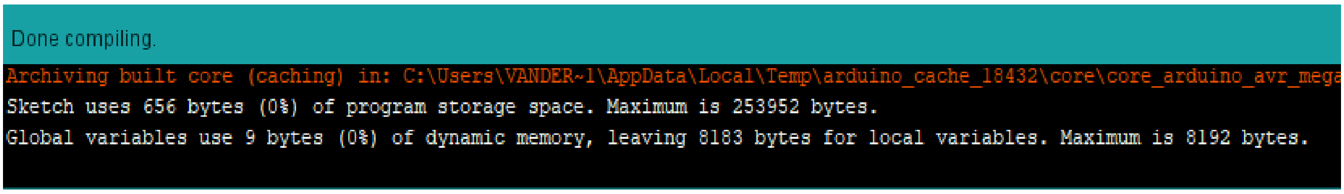
```
// #define FEATURE_SLEEP
#define FEATURE_ROTARY_ENCODER
// #define FEATURE_CMOS_SUPER_K
```

Figure 37, Rotary Encoder option

To conclusion this step compile the sketch. Uploading makes no sense at this stage.

Note:

If you are compiling for the first time you will probably receive the following message, fig. 38. The text in red does not mean alarm , the text below that indicates how many storage space is used of the total available amount.



```
Archiving built core (caching) in: C:\Users\VANDER~1\AppData\Local\Temp\arduino_cache_18432\core\core_arduino_avr_mega
Sketch uses 656 bytes (0%) of program storage space. Maximum is 253952 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 8183 bytes for local variables. Maximum is 8192 bytes.
```

Figure 38, used storage space.

Step 8.7: Signal key and loudspeaker extension.

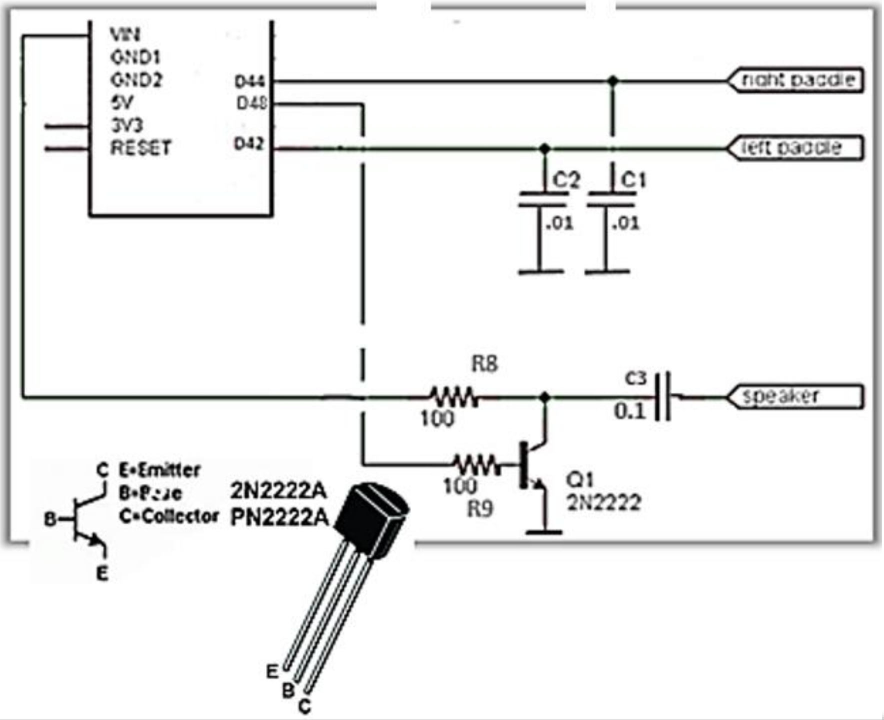


Figure 39, keyer and loudspeaker

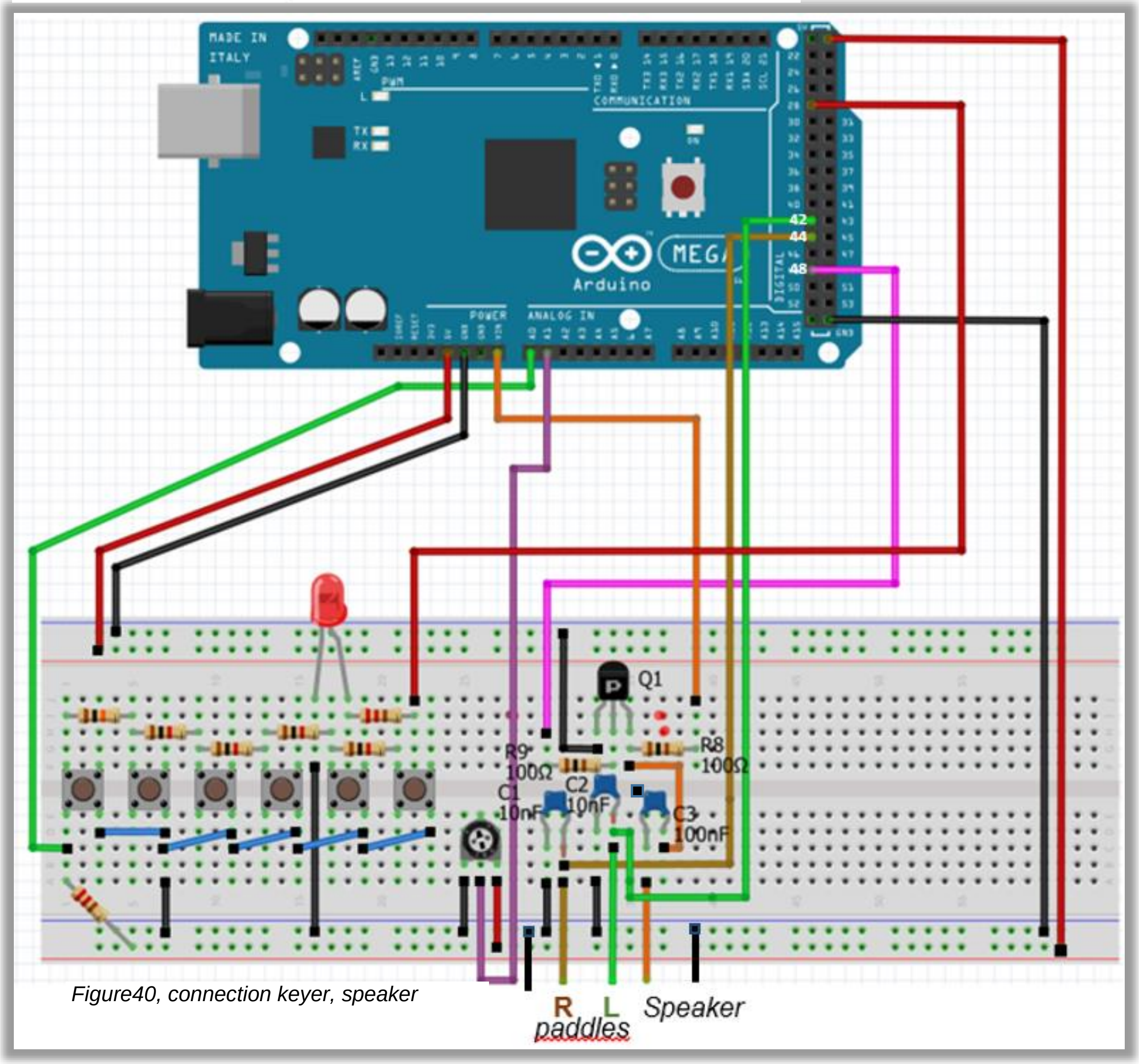


Figure40, connection keyer, speaker

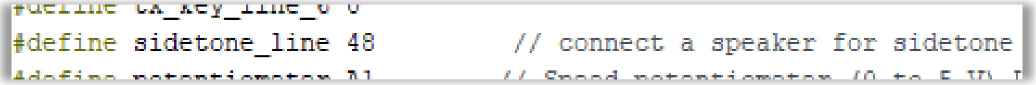
With this step, fig.39 and 40, you are able to convert the manual morse keying into sound using a loudspeaker, perhaps only interesting for those who have already mastered the CW. But beware: at step 8.7 you will add a keyboard with which audible CW characters can be produced.

And it gets even better when you add a display in step 8.8 where you can see which CW characters you produce.
For the loudspeaker and keyer paddles to work, there are no special commands necessary in the ***keyer_features_and_options.h*** file.
However, you must define in the ***keyer_pin_settings.h*** file which pin setting you want for the sidetone and paddles. The speaker has been chosen to be on pin 48,and pin 42 and 44 for the paddles, fig. 41 and 41a.

The 2N2222 transistor is a standard NPN switching transistor which must be connected according to the diagrams.

The key and sidetone pin settings, according to the Arduino pins, are:

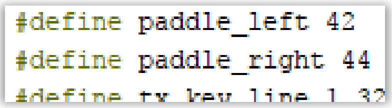
keyer_pin_settings.h



```
#define tx_key_line_6 0
#define sidetone_line 48         // connect a speaker for sidetone
#define potentiometer A1        // Speed potentiometer (0 to 5 V) I
```

Figure 41, side tone pin setting

keyer_pin_settings.h



```
#define paddle_left 42
#define paddle_right 44
#define tx_key_line_1 32
```

Figure 41a, paddles

Step 8.8: Connecting your keyboard.
First you are going to add a keyboard (connection), which allows you to produce audible morse signs without using the key. The keyboard I use is type PS/2, a whicht is easily available.
Fig. 42 shows the front view of the chassis part,and in fig. 43 you can see how the corresponding pins are connected. After checking, test the operation after compilation and uploading.

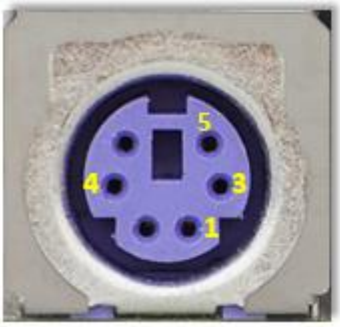
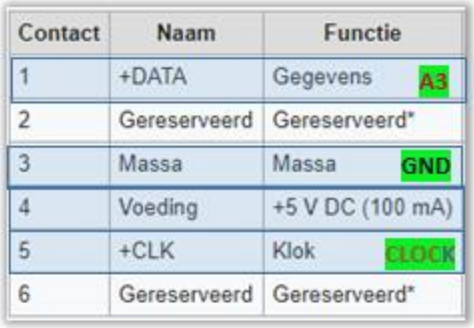


Figure42, front view chassis part

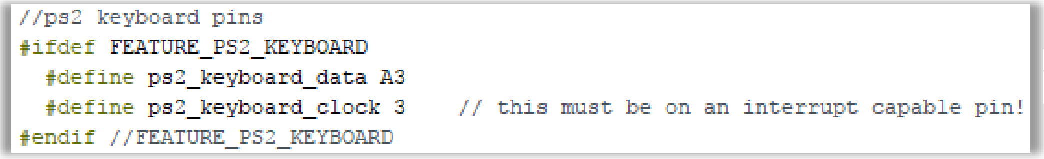


Contact	Naam	Functie
1	+DATA	Gegevens A3
2	Gereserveerd	Gereserveerd*
3	Massa	Massa GND
4	Voeding	+5 V DC (100 mA)
5	+CLK	Klok CLOCK
6	Gereserveerd	Gereserveerd*

Figure 43, pin names.

Connecting the keyboard is "a piece of cake". activate the following commands according to the figures 44 and 45 below. Fig. 46 shows the added wiring.

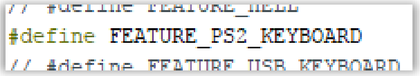
keyer_pin_settings.h bestand:



```
//ps2 keyboard pins
#ifdef FEATURE_PS2_KEYBOARD
  #define ps2_keyboard_data A3
  #define ps2_keyboard_clock 3    // this must be on an interrupt capable pin!
#endif //FEATURE_PS2_KEYBOARD
```

Figure 44, define pin settings.

keyer_features_and_options.h



```
// #define FEATURE_HELL
#define FEATURE_PS2_KEYBOARD
// #define FEATURE_USB_KEYBOARD
```

Figure 45, define feature_ps2_keyboard.

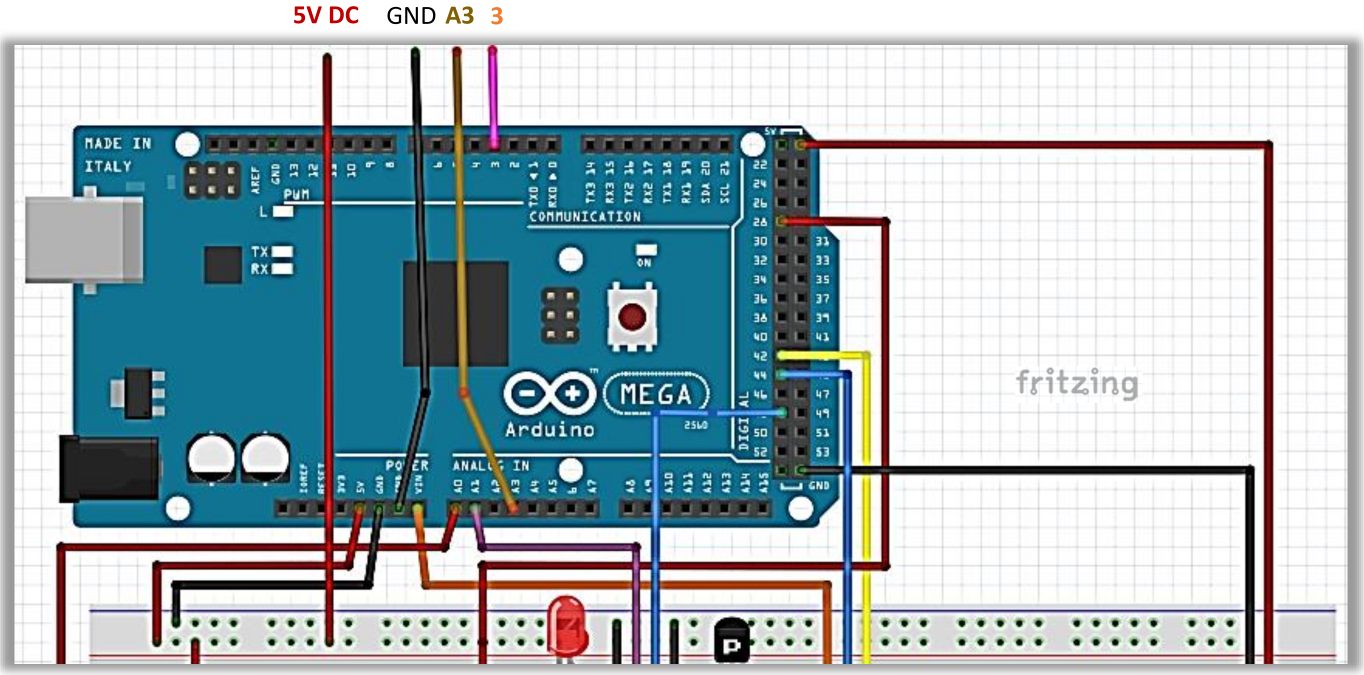
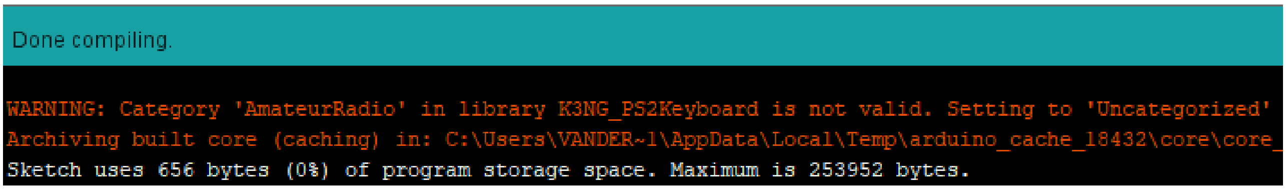


Figure 46, GND (black), A3 (bruin) en 3 (pink) and the 5V connection(red)

During the compilation you get the following warning, fig. 46a, regarding your keyboard. You may ignore this warning and proceed to upload.



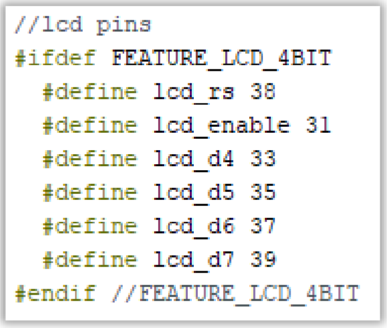
```
Done compiling.

WARNING: Category 'AmateurRadio' in library K3NG_PS2Keyboard is not valid. Setting to 'Uncategorized'
Archiving built core (caching) in: C:\Users\VANDER~1\AppData\Local\Temp\arduino_cache_18432\core\core_
Sketch uses 656 bytes (0%) of program storage space. Maximum is 253952 bytes.
```

Figure 46a, the warning message

Step 8.9: Connecting your display

By applying a display, the project is getting really fun, because you can now see what morse code text you produce with your keyboard, and possibly your key.
To make things easier to, I used a second breadboard which also has to be provided with 5V and GND connections. The display is a 4-bit 2 row LCD.
Later on, the decoding setup can be added to the 2nd breadboard. Both breadboards are used efficiently. Change in the *keyer_pin_settings.h* en *keyer_features_and_options.h* according to figs. 47 and 48. Figs. 50 and 51 show the board setup.

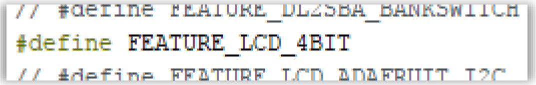
keyer_pin_settings.h bestand:



```
//lcd pins
#ifdef FEATURE_LCD_4BIT
  #define lcd_rs 38
  #define lcd_enable 31
  #define lcd_d4 33
  #define lcd_d5 35
  #define lcd_d6 37
  #define lcd_d7 39
#endif //FEATURE_LCD_4BIT
```

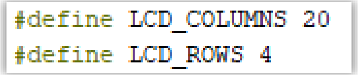
Figure 47, LCD setting

keyer_features_and_options.h bestand:



```
// #define FEATURE_DL2SBA_BANKSWITCH
#define FEATURE_LCD_4BIT
// #define FEATURE_LCD_ADAFRUIT_I2C
```

Figure 48, feature LCD



```
#define LCD_COLUMNS 20
#define LCD_ROWS 4
```

Figure 48a

For a 4-bit LCD display with for example 20 rows and 4 columns you will have to change the following items in ***keyer_settings.h,*** ***fig.48a.*** The LCD pinsettings can be left unchanged. If you follow the diagram below, fig. 49, the display should work properly. Set the potentiometer R10 so the characters become visible.

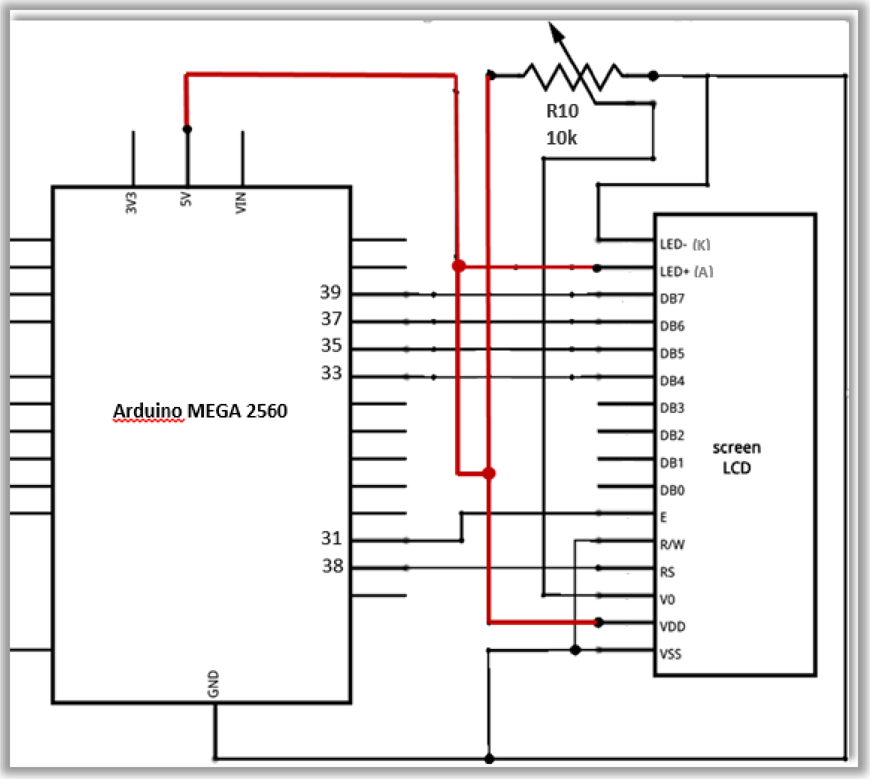


Figure 49, LCD circuit

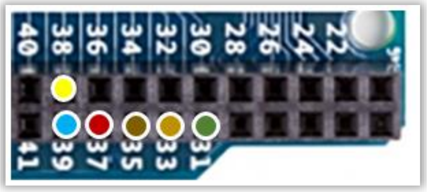


Figure 50, detail pin setting

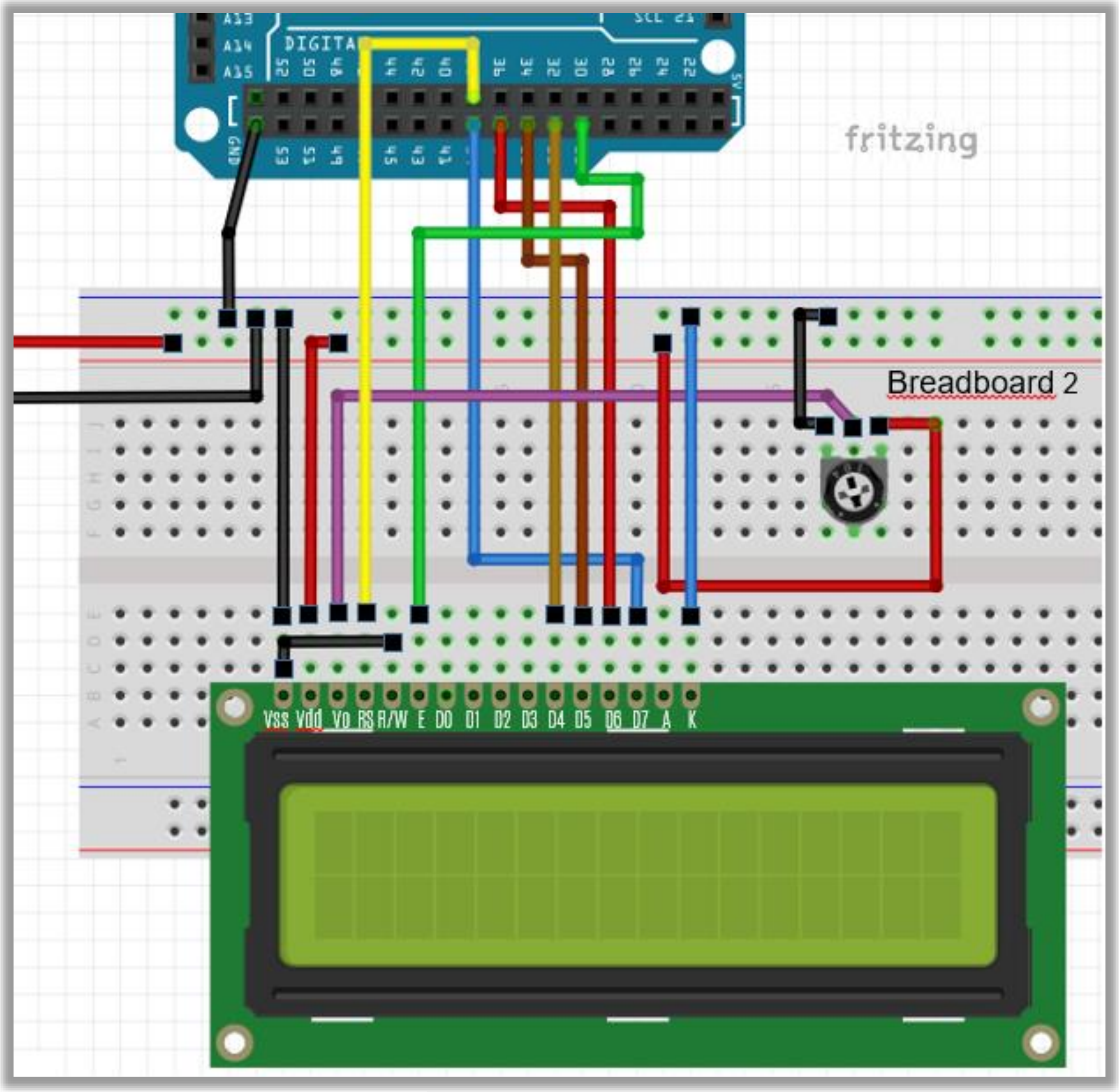
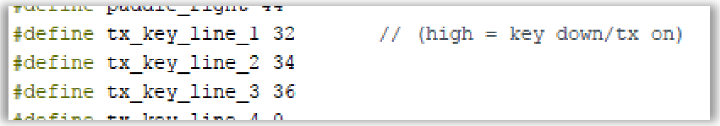


Figure 51, LCD mounting, 4-bits 2-line LCD

Step 8.10: Connection the Transceiver.

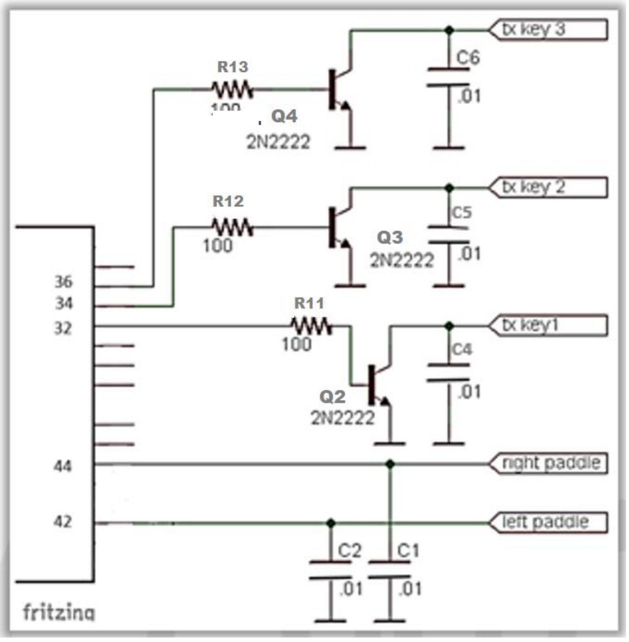
Now go on working on your 1st breadboard.
The K3NG design has 3 TX connections, which means that you can connect 3 transceivers, for example a Kenwood, Icom and a Yaesu. I imagine that the three transceivers are set so that, if one band is "closed", you can continue a contest on another bands. Maybe a bit far ferched, but the possibility is available.
To activate this option, you must change the ports for the *tx_key_line_1, tx_key_line_2* and *tx_key_line_3* that appear in the ***keyer_pin_settings.h*** file. I personally use digital pins 32, 34 and 36 for that, see fig.52.



```
#define paddle_right 44
#define tx_key_line_1 32       // (high = key down/tx on)
#define tx_key_line_2 34
#define tx_key_line_3 36
#define tx_key_line_4 0
```

Figure 52, the three drie TX three outputs activated.

The CW ops can now key the channel, and will be able to "hear" the opposite station.
With the keyboard you can also control the TX in CW mode, but returning Morse signals can not be deciphered for the "non CW-ers", but the next chapter will tell you what to do about that. Figures 53 and 54 and 55 show the schematic and additional wiring on the breadboard.



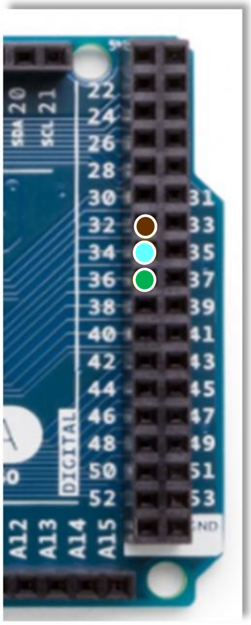


Figure 53, TX connection diagram.

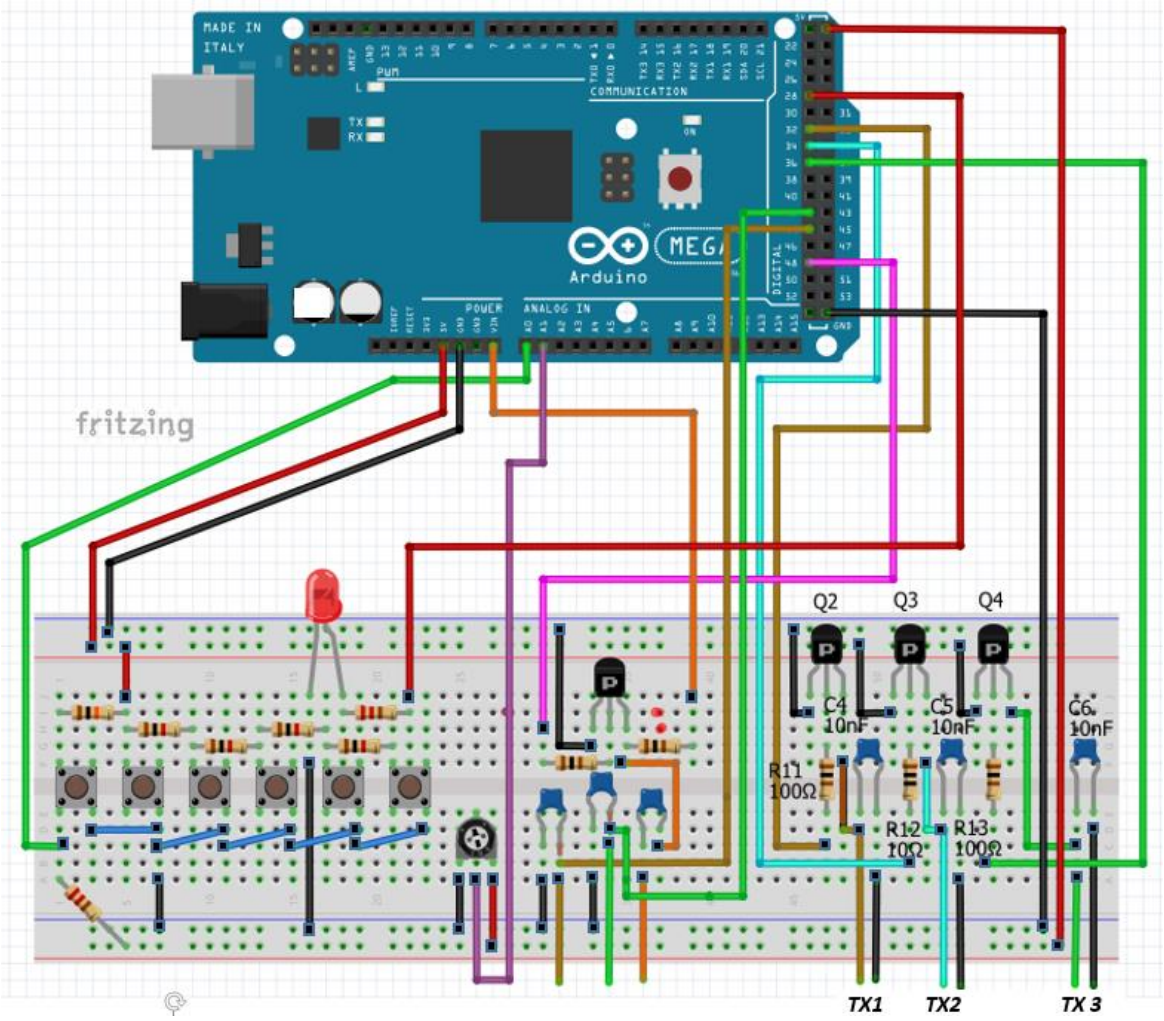


Figure 55, TX outputs.

Stap 8.11: GEORTZ DSP CW DECODER.

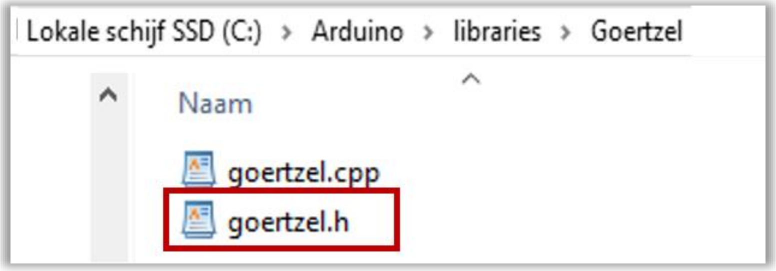
The "keying" chapter is now complete and it is time for you to look at the decode section which converts morse code to readable text on your display. This part is largely taken over from the KF4BZT site, in which he also describes that a conversion must take place from the UNO to the MEGA 2560 controller. He makes use of the basic code of OZ1JHM which is on this website:

http://www.skovholm.com/cwdecoder

Decoding is based on the Goertzel algorithm and you can find the information at:

https://en.wikipedia.org/wiki/Goertzel_algorithm

The whole decoding issue is such a complex mathematical item that I have not tried to dig any deeper into it. A file for the Arduino has been included in the library ***goertzel.h.***
This provides all decoding algorithms that are necessary for this to work properly. To that end a number of settings must be adjusted so that the Arduino Mega can handle this code. This file can be found in the Goertzel folder, as shown in fig.56.



Lokale schijf SSD (C:) › Arduino › libraries › Goertzel

Naam

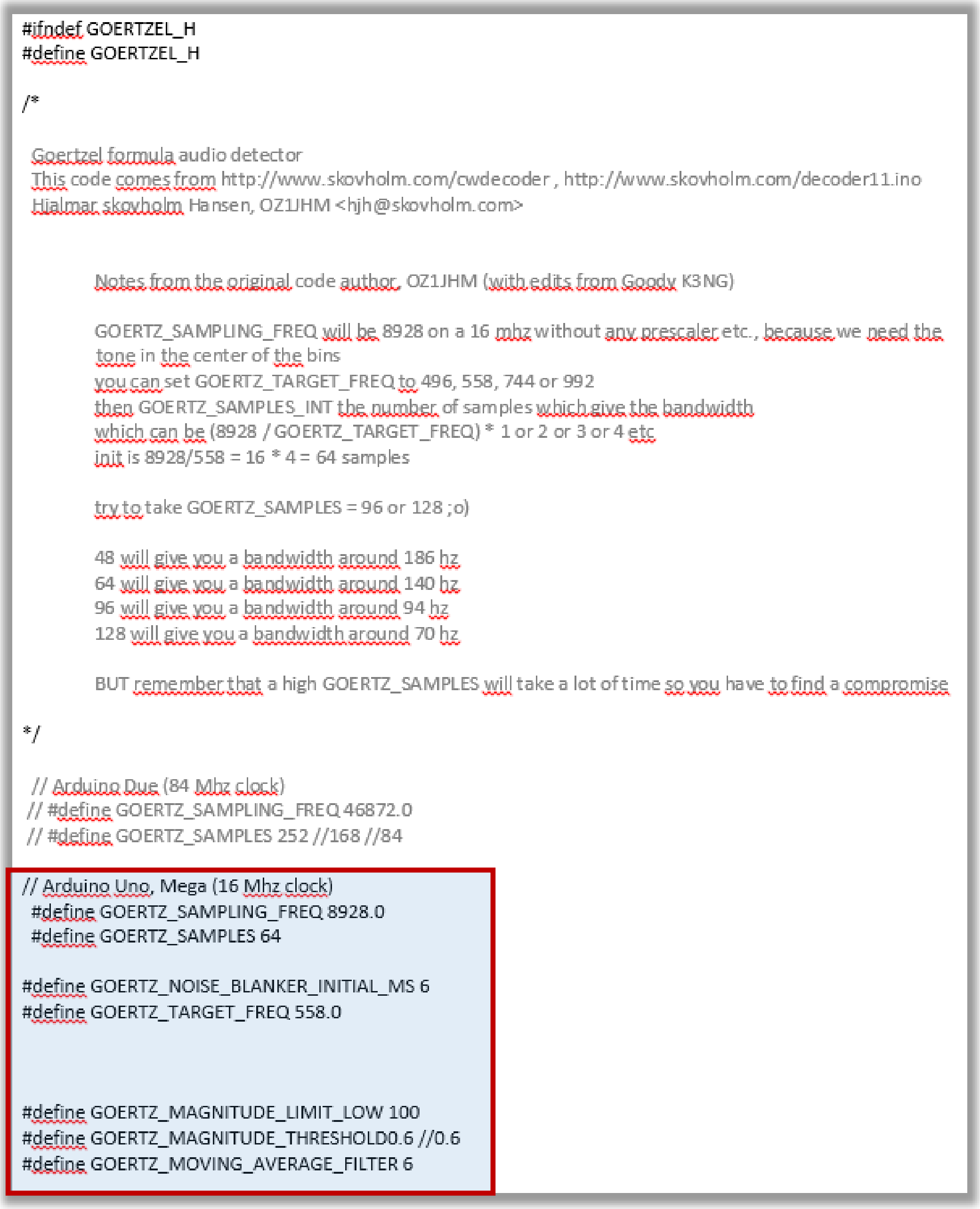
goertzel.cpp
goertzel.h

Figure 56, folder Goertzel

Open the ***goertzel.h*** file with e.g. WordPad where you can find some interesting information about the sampling frequency and bandwidth.

In the first part of the file, printed in gray, you will find further explanation of OZ1JHM's settings. There are settings that need attention in case the decoding does not work well. According to OZ1JHM, the Target Frequency should work with 558 Hz or maybe 744 Hz.

Change it in the red box, fig.57 to make the codes suitable for the Arduino Mega. But remember that a high GOERTZ_SAMPLES value takes a lot of CPU time, so you have to make a compromise.



```
#ifndef GOERTZEL_H
#define GOERTZEL_H

/*

 Goertzel formula audio detector
 This code comes from http://www.skovholm.com/cwdecoder , http://www.skovholm.com/decoder11.ino
 Hjalmar skovholm Hansen, OZ1JHM <hjh@skovholm.com>


        Notes from the original code author, OZ1JHM (with edits from Goody K3NG)

        GOERTZ_SAMPLING_FREQ will be 8928 on a 16 mhz without any prescaler etc., because we need the
        tone in the center of the bins
        you can set GOERTZ_TARGET_FREQ to 496, 558, 744 or 992
        then GOERTZ_SAMPLES_INT the number of samples which give the bandwidth
        which can be (8928 / GOERTZ_TARGET_FREQ) * 1 or 2 or 3 or 4 etc
        init is 8928/558 = 16 * 4 = 64 samples

        try to take GOERTZ_SAMPLES = 96 or 128 ;o)

        48 will give you a bandwidth around 186 hz
        64 will give you a bandwidth around 140 hz
        96 will give you a bandwidth around 94 hz
        128 will give you a bandwidth around 70 hz

        BUT remember that a high GOERTZ_SAMPLES will take a lot of time so you have to find a compromise

*/

 // Arduino Due (84 Mhz clock)
// #define GOERTZ_SAMPLING_FREQ 46872.0
// #define GOERTZ_SAMPLES 252 //168 //84

// Arduino Uno, Mega (16 Mhz clock)
 #define GOERTZ_SAMPLING_FREQ 8928.0
 #define GOERTZ_SAMPLES 64

#define GOERTZ_NOISE_BLANKER_INITIAL_MS 6
#define GOERTZ_TARGET_FREQ 558.0



#define GOERTZ_MAGNITUDE_LIMIT_LOW 100
#define GOERTZ_MAGNITUDE_THRESHOLD0.6 //0.6
#define GOERTZ_MOVING_AVERAGE_FILTER 6
```

Figure57, Goertz settings.

In the *keyer_pin_settings.h* file the pinsetting needs to be adjusted. You choose pin A11, and for *cw decoder indicato,r* pin 24, fig.58, 59 en 60.

Finally in the *keyer_features_and_options.h* the option for audio detector and cw_decoder must be activated, fig.61.

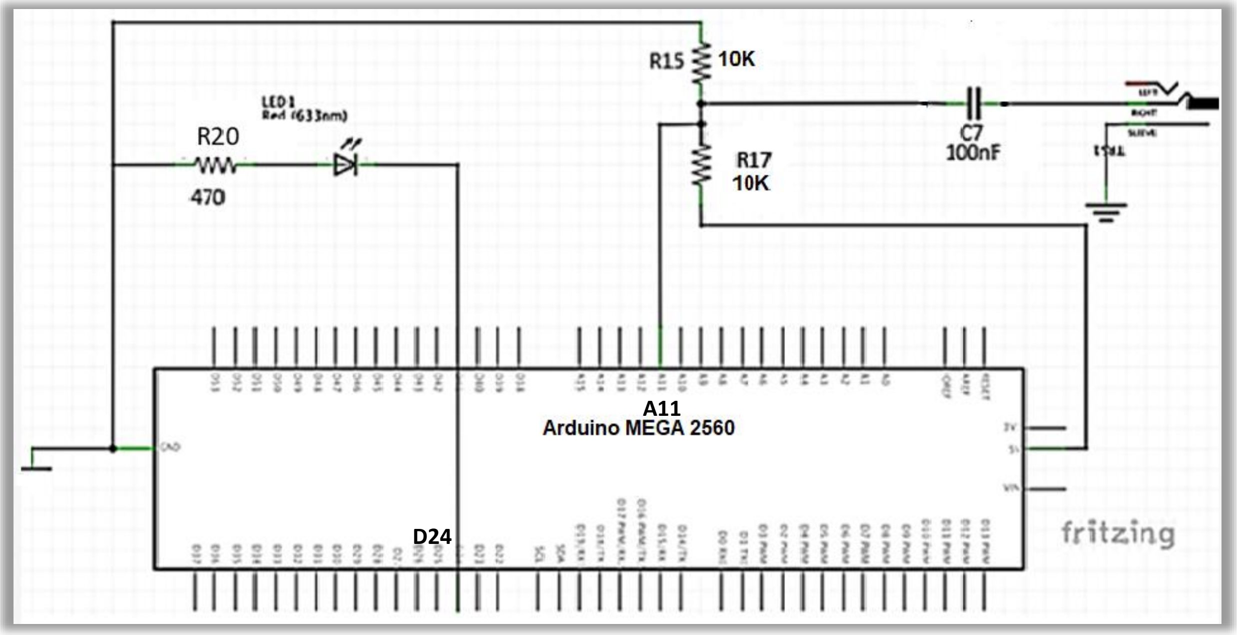
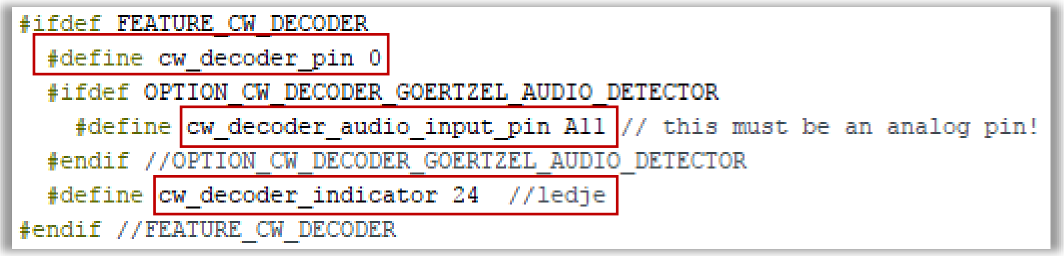


Figure 58, adjusting fort he MEGA

Choose the number "zero" for ***define cw_decoder_pin.***

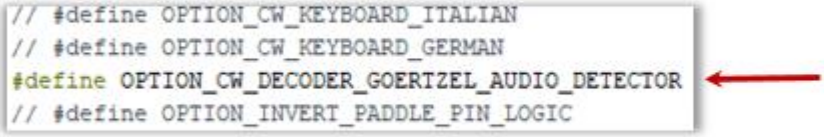
keyer_pin_settings.h



```
#ifdef FEATURE_CW_DECODER
  #define cw_decoder_pin 0
  #ifdef OPTION_CW_DECODER_GOERTZEL_AUDIO_DETECTOR
    #define cw_decoder_audio_input_pin A11 // this must be an analog pin!
  #endif //OPTION_CW_DECODER_GOERTZEL_AUDIO_DETECTOR
  #define cw_decoder_indicator 24  //ledje
#endif //FEATURE_CW_DECODER
```

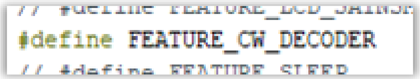
Figure 59, pinsetting for the decoder.

keyer_features_and_options.h



```
// #define OPTION_CW_KEYBOARD_ITALIAN
// #define OPTION_CW_KEYBOARD_GERMAN
#define OPTION_CW_DECODER_GOERTZEL_AUDIO_DETECTOR
// #define OPTION_INVERT_PADDLE_PIN_LOGIC
```

Figure 60, Goertzel Audio detector



```
// #define FEATURE_LCD_SAINSM
#define FEATURE_CW_DECODER
// #define FEATURE_SLEEP
```

Figure 61, feature_cw_decoder

NB. You can position this circuit on your 2nd breadboard, next to the display circuit. The LED serves to make the alignment to the Morse signal easier, and you connect it on A11, fig. 58 and 62.

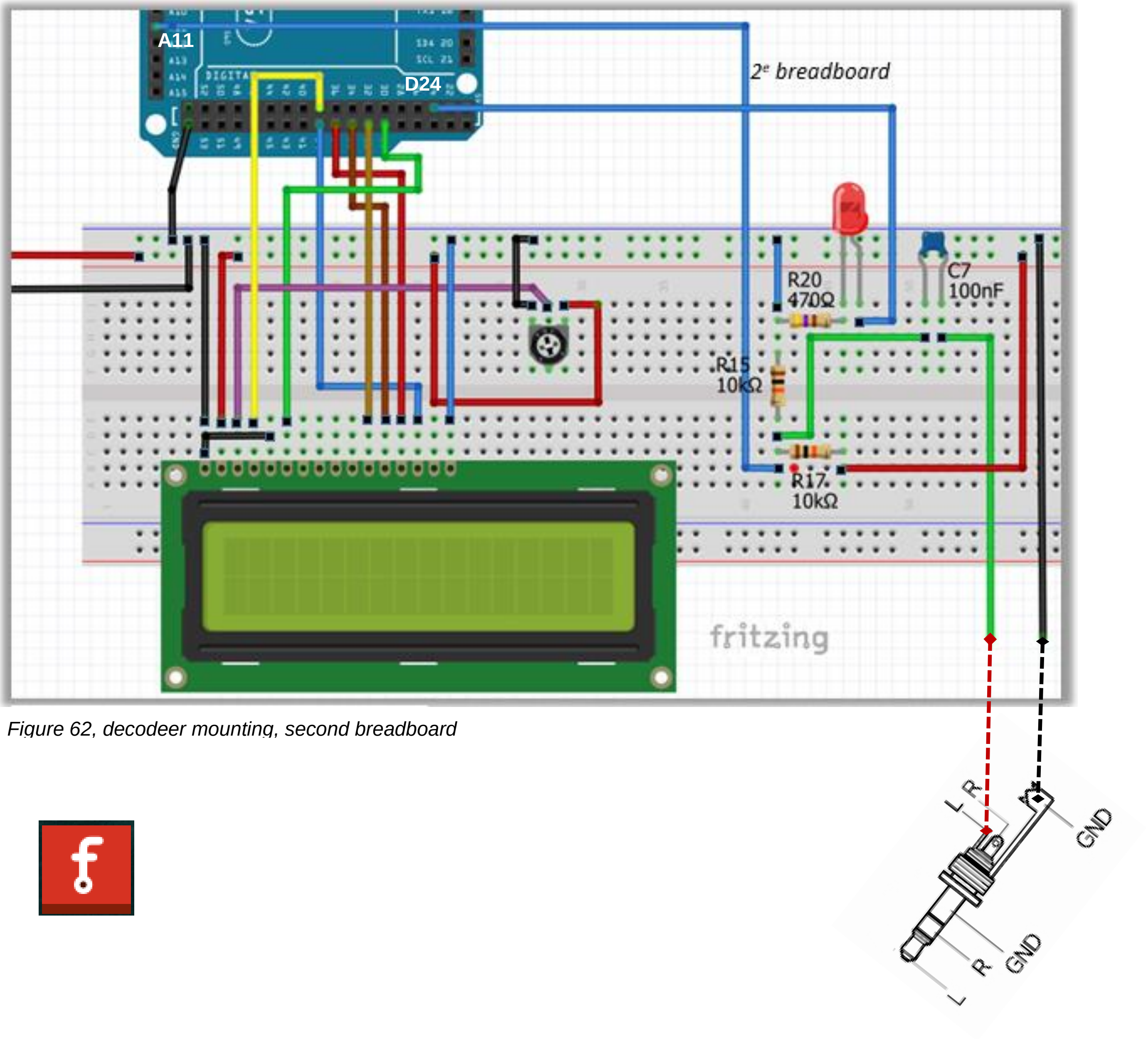


Figure 62, decodeer mounting, second breadboard

NB:
In many of the figures you will come across the name Fritzing and you may wonder what this means.
Fritzing is the name of a program with which you design circuits and breadboard setups.
If you want to know more about it, you can download the (freeware) program from the website

http://fritzing.org/download/

9. How does the detector work?

Tune your receiver to the CW part of the HF band. Make sure you have the LED connected between the mass and the pin you have chosen. Tune to a station until the LED starts to flash when receiving CW. You should use fine-tuning to make it readable, so that the LED shows that the right data is being sent.

Play around with the above settings in the goertzel.h file to see if you can get a better decoding. With the bandwidth setting of the transceiver (CW mode) at 1000Hz, the decoding is optimal in my case.

NB.
To be honest, I must say that you should not expect too much from decoding. The decoding method is still in an experimental phase. For the novice CW operator, the above mentioned decoding on the display is still very nice to follow and then mainly as there is a "proper" regular keying not disturbed by nearby CW signals.
You might also try to tune in beacon stations, which feature a nice keying characteristic

Further information about the operation and additional information from the CW-keyer can be obtained from the following web site:

https://blog.radioartisan.com/arduino-cw-keyer/

The group below is very active and helpful, you can ask all your questions about the K3NG-CW keyer and other K3NG projects **.**

https://groups.io/g/radioartisan ***(NEW)***

Useful additional sites:
https://www.arduino.cc/en/Main/Software
https://github.com/k3ng/k3ng_cw_keyer
https://learn.sparkfun.com/tutorials/how-to-read-a-schematic
https://www.arduino.cc/en/Main/Software
http://fritzing.org/home/
https://makerzone.mathworks.com/resources/getting-started-with-arduino-mega-2560-hardware/
http://www.skovholm.com/cwdecoder
http://www.justradios.com/uFnFpF.html

10. Keyboard commands

Special PS2 Keybords commands

F1 through F12------	play memories 1 through 12
Up Arrow-------------	Increase CW Speed 1 WPM
Down Arrow---------	Decrease CW Speed 1 WPM
Page Up---------------	Increase sidetone frequency
Page Down-----------	Decrease sidetone frequency
Right Arrow----------	Dah to Dit Ratio increase
Left Arrow------------	Dah to Dit Ratio decrease
Home------------------	reset Dah to Dit Ratio to default
Tab---------------------	pause sending
Delete-----------------	delete the last character in the buffer
Esc---------------------	stop sending and clear the buffer
Scroll Lock ----------	Merge the next two characters to form a prosign
Shift-------------------	Scroll Lock – toggle PTT line
CTRL-A----------------	Iambic A Mode
CTRL-B----------------	Iambic B Mode
CTRL-C-----------------	Single Paddle Mode
CTRL-D-----------------	Ultimatic Mode
CTRL-E-----------------	Set Serial Number
CTRL-G----------------	Bug Mode
CTRL-H----------------	Hellschreiber Mode (requires FEATURE_HELL)
CTRL-I-----------------	TX Line Disable/Enable
CTRL-M---------------	Set Farnsworth Speed (requires FEATURE_FARNSWORTH)
CTRL-N----------------	Paddle Revers
CTRL-O----------------	Sidetone On/Off
CTRL-T -----------------	Tune
CTRL-U-----------------	PTT Manual On/Off
CTRL-W---------------	Set WPM
CTRL-Z----------------	Autospace On/Off
SHIFT-F1, F2, F3…	Program memory 1, 2, 3… (programmeren van de geheugen toetsen)
ALT-F1, F2, F3…	Repeat memory 1, 2, 3…
CTRL-F1, F2, F3…	Switch to transmitter 1, 2, 3…

11. List of components

Required components:

Part	Name, values	number
	Arduino MEGA, 2560	1x
	Breadboard	2 x
	Keyboard PS/2	1x
S 1,2,3,4,5,6	on / off push buttons, 6x6 mm.	6 x
	connecting wires	a lot of
Led 1, 2	Red (633nm)	2x
R1,11,15,17	10 kΩ	4x
R2,3,4,5,6	1 kΩ	5x
R19, 20	470 Ω	2x
R7,10	10 kΩ, pot. meter	2x
R 8,9,11,12,13,16	100 Ω	6x
C1,2,4,5,6	0.01 µF (10nF)	5x
C3,C7	0.1 µF (100nF	2x
Q 1,2,3,4	2N2222A (NPN)	4x

12. Appendices

Keyer_pin_settings.h

```
/* Pins - you must review these and configure ! */
#ifndef keyer_pin_settings_h
#define keyer_pin_settings_h

#define paddle_left 42
#define paddle_right 44
#define tx_key_line_1 11       // (high = key down/tx on)
#define tx_key_line_2 12
#define tx_key_line_3 0
#define tx_key_line_4 0
#define tx_key_line_5 0
#define tx_key_line_6 0
#define sidetone_line 48         // connect a speaker for sidetone
#define potentiometer A1        // Speed potentiometer (0 to 5 V) Use pot from 1k to 10k
#define ptt_tx_1 0             // PTT ("push to talk") lines
#define ptt_tx_2 0             //   Can be used for keying fox transmitter, T/R switch, or keying slow boatanchors
#define ptt_tx_3 0             //   These are optional - set to 0 if unused
#define ptt_tx_4 0
#define ptt_tx_5 0
#define ptt_tx_6 0
#define tx_key_dit 0           // if defined, goes active for dit (any transmitter) - customized with tx_key_dit_and_dah_pins_active_state and tx_key_dit_and_dah_pins_inactive_state
#define tx_key_dah 0           // if defined, goes active for dah (any transmitter) - customized with tx_key_dit_and_dah_pins_active_state and tx_key_dit_and_dah_pins_inactive_state

#ifdef FEATURE_COMMAND_BUTTONS
  #define analog_buttons_pin A0
  #define command_mode_active_led 28
#endif //FEATURE_COMMAND_BUTTONS

/*
FEATURE_SIDETONE_SWITCH
  Enabling this feature and an external toggle switch  adds switch control for playing cw sidetone.
  ST Switch status is displayed in the status command.  This feature will override the software control of the sidetone (\o).
  Arduino pin is assigned by SIDETONE_SWITCH
*/

#ifdef FEATURE_SIDETONE_SWITCH
  #define SIDETONE_SWITCH 8
#endif //FEATURE_SIDETONE_SWITCH


//lcd pins
#ifdef FEATURE_LCD_4BIT
  #define lcd_rs 38
  #define lcd_enable 31
  #define lcd_d4 33
  #define lcd_d5 35
  #define lcd_d6 37
  #define lcd_d7 39
#endif //FEATURE_LCD_4BIT
```

```
#ifdef FEATURE_LCD1602_N07DH
  #define lcd_rs 8
  #define lcd_enable 9
  #define lcd_d4 4
  #define lcd_d5 5
  #define lcd_d6 6
  #define lcd_d7 7
#endif //FEATURE_LCD1602_N07DH

//ps2 keyboard pins
#ifdef FEATURE_PS2_KEYBOARD
  #define ps2_keyboard_data A3
  #define ps2_keyboard_clock 3    // this must be on an interrupt capable pin!
#endif //FEATURE_PS2_KEYBOARD

// rotary encoder pins and options - rotary encoder code from Jim Balls M0CKE
#ifdef FEATURE_ROTARY_ENCODER
  #define OPTION_ENCODER_HALF_STEP_MODE     // Half-step mode?
  #define rotary_pin1 0                      // CW Encoder Pin
  #define rotary_pin2 0                    // CCW Encoder Pin
  #define OPTION_ENCODER_ENABLE_PULLUPS     // define to enable weak pullups.
#endif //FEATURE_ROTARY_ENCODER

#ifdef FEATURE_LED_RING
  #define led_ring_sdi    A10 //2    //Data
  #define led_ring_clk    A9 //3    //Clock
  #define led_ring_le     A8 //4    //Latch
#endif //FEATURE_LED_RING

#ifdef FEATURE_ALPHABET_SEND_PRACTICE
  #define correct_answer_led 0
  #define wrong_answer_led 0
#endif //FEATURE_ALPHABET_SEND_PRACTICE

#ifdef FEATURE_PTT_INTERLOCK
  #define ptt_interlock 0  // this pin disables PTT and TX KEY
#endif //FEATURE_PTT_INTERLOCK

#ifdef FEATURE_STRAIGHT_KEY
  #define pin_straight_key 52
#endif //FEATURE_STRAIGHT_KEY

#ifdef FEATURE_CW_DECODER
  #define cw_decoder_pin 0
  #ifdef OPTION_CW_DECODER_GOERTZEL_AUDIO_DETECTOR
    #define cw_decoder_audio_input_pin A11 // this must be an analog pin!
  #endif //OPTION_CW_DECODER_GOERTZEL_AUDIO_DETECTOR
  #define cw_decoder_indicator 24
#endif //FEATURE_CW_DECODER


#if defined(FEATURE_COMPETITION_COMPRESSION_DETECTION)
  #define compression_detection_pin 13
#endif //FEATURE_COMPETITION_COMPRESSION_DETECTION

#if defined(FEATURE_SLEEP)
```

```
  #define keyer_awake 0
#endif

#if defined(FEATURE_CAPACITIVE_PADDLE_PINS)
  #define capactive_paddle_pin_inhibit_pin 0     // if this pin is defined and is set high, the capacitive paddle pins will switch to
normal (non-capacitive) sensing mode
#endif

#ifdef FEATURE_4x4_KEYPAD
  #define Row3 33
  #define Row2 32
  #define Row1 31
  #define Row0 30
  #define Col3 37
  #define Col2 36
  #define Col1 35
  #define Col0 34
#endif

#ifdef FEATURE_3x4_KEYPAD
  #define Row3 33
  #define Row2 32
  #define Row1 31
  #define Row0 30
  #define Col2 36
  #define Col1 35
  #define Col0 34
#endif

#else

  #error "Multiple pin_settings.h files included somehow..."

#endif //keyer_pin_settings_h
```

keyer_features_and_options.h

```
// compile time features and options - comment or uncomment to add or delete features
// FEATURES add more bytes to the compiled binary, OPTIONS change code behavior


#define FEATURE_COMMAND_BUTTONS
// #define FEATURE_COMMAND_LINE_INTERFACE  // Command Line Interface functionality
#define FEATURE_MEMORIES            // on the Arduino Due, you must have FEATURE_EEPROM_E24C1024 and E24C1024 EEPROM hardware in order to compile this
#define FEATURE_MEMORY_MACROS
// #define FEATURE_WINKEY_EMULATION    // disabling Automatic Software Reset is highly recommended (see documentation)
// #define FEATURE_BEACON
// #define FEATURE_TRAINING_COMMAND_LINE_INTERFACE
#define FEATURE_POTENTIOMETER        // do not enable unless you have a potentiometer connected, otherwise noise will falsely trigger wpm changes
// #define FEATURE_SIDETONE_SWITCH   // adds switch control for the sidetone output. requires an external toggle switch (assigned to an arduino pin - see keyer_pin_settings.h).
// #define FEATURE_SERIAL_HELP
// #define FEATURE_HELL
#define FEATURE_PS2_KEYBOARD       // Use a PS2 keyboard to send code - Change keyboard layout (non-US) in K3NG_PS2Keyboard.h.  Additional options below.
// #define FEATURE_USB_KEYBOARD         // Use a USB keyboard to send code - Uncomment three lines in k3ng_keyer.ino (search for note_usb_uncomment_lines)
// #define FEATURE_CW_COMPUTER_KEYBOARD  // Have an Arduino Due or Leonardo act as a USB HID (Human Interface Device) keyboard and use the paddle to "type" characters on the computer -- uncomment this line in ino file: #include <Keyboard.h>
// #define FEATURE_DEAD_OP_WATCHDOG
// #define FEATURE_AUTOSPACE
// #define FEATURE_FARNSWORTH
// #define FEATURE_DL2SBA_BANKSWITCH       // Switch memory banks feature as described here: http://dl2sba.com/index.php?option=com_content&view=article&id=131:nanokeyer&catid=15:shack&Itemid=27#english
#define FEATURE_LCD_4BIT              // classic LCD disidefplay using 4 I/O lines
// #define FEATURE_LCD_ADAFRUIT_I2C         // Adafruit I2C LCD display using MCP23017 at addr 0x20
// #define FEATURE_LCD_ADAFRUIT_BACKPACK    // Adafruit I2C LCD Backup using MCP23008 (courtesy Josiah Ritchie, KE0BLL)
// #define FEATURE_LCD_YDv1              // YourDuino I2C LCD display with old LCM 1602 V1 ic
// #define FEATURE_LCD1602_N07DH      // http://linksprite.com/wiki/index.php5?title=16_X_2_LCD_Keypad_Shield_for_Arduino
// #define FEATURE_LCD_SAINSMART_I2C
#define FEATURE_CW_DECODER
// #define FEATURE_SLEEP                 // go to sleep after x minutes to conserve battery power (not compatible with Arduino DUE, may have mixed results with Mega and Mega ADK)
#define FEATURE_ROTARY_ENCODER         // rotary encoder speed control
// #define FEATURE_CMOS_SUPER_KEYER_IAMBIC_B_TIMING
// #define FEATURE_HI_PRECISION_LOOP_TIMING
// #define FEATURE_USB_MOUSE             // Uncomment three lines in k3ng_keyer.ino (search for note_usb_uncomment_lines)
// #define FEATURE_CAPACITIVE_PADDLE_PINS  // remove the bypass capacitors on the paddle_left and paddle_right lines when using capactive paddles
// #define FEATURE_LED_RING              // Mayhew Labs Led Ring support
// #define FEATURE_ALPHABET_SEND_PRACTICE  // enables command mode S command - created by Ryan, KC2ZWM
// #define FEATURE_PTT_INTERLOCK
// #define FEATURE_QLF
// #define FEATURE_EEPROM_E24C1024
// #define FEATURE_STRAIGHT_KEY
// #define FEATURE_DYNAMIC_DAH_TO_DIT_RATIO
// #define FEATURE_PADDLE_ECHO
// #define FEATURE_STRAIGHT_KEY_ECHO
// #define FEATURE_AMERICAN_MORSE
// #define FEATURE_4x4_KEYPAD         // code contributed by Jack, W0XR - documentation:
https://github.com/k3ng/k3ng_cw_keyer/wiki/380-Feature:-Keypad
// #define FEATURE_3x4_KEYPAD         // code contributed by Jack, W0XR - documentation:
https://github.com/k3ng/k3ng_cw_keyer/wiki/380-Feature:-Keypad


// #define FEATURE_COMMAND_LINE_INTERFACE_ON_SECONDARY_PORT     // Activate the Command Line interface
on the secondary serial port
#define OPTION_PRIMARY_SERIAL_PORT_DEFAULT_WINKEY_EMULATION  // Use when activating both FEATURE_WINKEY_EMULATION and FEATURE_COMMAND_LINE_INTERFACE
                                                    //    simultaneously.  This will make Winkey emulation be the default at boot up;
                                                    //    hold command button down at boot up to activate CLI mode

// #define OPTION_SUPPRESS_SERIAL_BOOT_MSG
#define OPTION_INCLUDE_PTT_TAIL_FOR_MANUAL_SENDING
#define OPTION_EXCLUDE_PTT_HANG_TIME_FOR_MANUAL_SENDING
// #define OPTION_WINKEY_DISCARD_BYTES_AT_STARTUP     // if ASR is not disabled, you may need this to discard
errant serial port bytes at startup
// #define OPTION_WINKEY_STRICT_EEPROM_WRITES_MAY_WEAR_OUT_EEPROM // with this activated the unit will
write non-volatile settings to EEPROM when set by Winkey commands
// #define OPTION_WINKEY_SEND_WORDSPACE_AT_END_OF_BUFFER
#define OPTION_WINKEY_STRICT_HOST_OPEN              // require an admin host open Winkey command before doing
any other commands
#define OPTION_WINKEY_2_SUPPORT                  // comment out to revert to Winkey version 1 emulation
#define OPTION_WINKEY_INTERRUPTS_MEMORY_REPEAT
//#define OPTION_WINKEY_UCXLOG_9600_BAUD            // use this only with UCXLog configured for Winkey 9600 baud
mode
// #define OPTION_WINKEY_2_HOST_CLOSE_NO_SERIAL_PORT_RESET  // activate this when using Winkey 2 emulation and Win-Test
// #define OPTION_WINKEY_FREQUENT_STATUS_REPORT        // activate this to make Winkey emulation play better
with RUMlog and RUMped
#define OPTION_WINKEY_IGNORE_LOWERCASE             // Enable for typical K1EL Winkeyer behavior (use for
SkookumLogger version 1.10.14 and prior to workaround "r" bug)
// #define OPTION_REVERSE_BUTTON_ORDER              // This is mainly for the DJ0MY NanoKeyer
http://nanokeyer.wordpress.com/
#define OPTION_PROG_MEM_TRIM_TRAILING_SPACES        // trim trailing spaces from memory when programming in
command mode
#define OPTION_DIT_PADDLE_NO_SEND_ON_MEM_RPT        // this makes dit paddle memory interruption a little
smoother
// #define OPTION_MORE_DISPLAY_MSGS                  // additional optional display messages - comment out to save
memory
// #define OPTION_N1MM_WINKEY_TAB_BUG_WORKAROUND       // enable this to ignore the TAB key in the Send CW
window (this breaks SO2R functionality in N1MM)
// #define OPTION_WATCHDOG_TIMER                   // this enables a four second ATmega48/88/168/328 watchdog
timer; use for unattended/remote operation only
// #define OPTION_MOUSE_MOVEMENT_PADDLE             // experimental (just fooling around) - mouse movement will
act like a paddle
// #define OPTION_NON_ENGLISH_EXTENSIONS  // add support for additional CW characters (i.e. À, Å, Þ, etc.)
// #define OPTION_KEEP_PTT_KEYED_WHEN_CHARS_BUFFERED    // this option keeps PTT high if there are characters
buffered from the keyboard, the serial interface, or Winkey
```

```
// #define OPTION_DISPLAY_NON_ENGLISH_EXTENSIONS  // LCD display suport for non-English (NO/DK/DE) characters - Courtesy of OZ1JHM
// #define OPTION_UNKNOWN_CHARACTER_ERROR_TONE
// #define OPTION_DO_NOT_SAY_HI
// #define OPTION_PS2_NON_ENGLISH_CHAR_LCD_DISPLAY_SUPPORT // makes some non-English characters from the PS2 keyboard display correctly in the LCD display (donated by Marcin sp5iou)
// #define OPTION_PS2_KEYBOARD_RESET // reset the PS2 keyboard upon startup with 0xFF (contributed by Bill, W9BEL)
// #define OPTION_SAVE_MEMORY_NANOKEYER
#define OPTION_CW_KEYBOARD_CAPSLOCK_BEEP
// #define OPTION_CW_KEYBOARD_ITALIAN
// #define OPTION_CW_KEYBOARD_GERMAN
#define OPTION_CW_DECODER_GOERTZEL_AUDIO_DETECTOR
// #define OPTION_INVERT_PADDLE_PIN_LOGIC
// #define OPTION_ADVANCED_SPEED_DISPLAY //enables "nerd" speed visualization on display: wpm, cpm (char per min), duration of dit and dah in milliseconds and ratio (contributed by Giorgio, IZ2XBZ)
// #define OPTION_PROSIGN_SUPPORT    // additional prosign support for paddle and straight key echo on display, CLI, and in memory storage
// #define OPTION_RUSSIAN_LANGUAGE_SEND_CLI // Russian language CLI sending support (contributed by Павел Бирюков, UA1AQC)
#define OPTION_DO_NOT_SEND_UNKNOWN_CHAR_QUESTION
// #define OPTION_CMOS_SUPER_KEYER_IAMBIC_B_TIMING_ON_BY_DEFAULT
// #define OPTION_SIDETONE_DIGITAL_OUTPUT_NO_SQUARE_WAVE

// #define OPTION_WORDSWORTH_CZECH
// #define OPTION_WORDSWORTH_DEUTSCH
// #define OPTION_WORDSWORTH_NORSK
```

13. Attachments, *pictures of the Arduino K3NG CW Keyer*