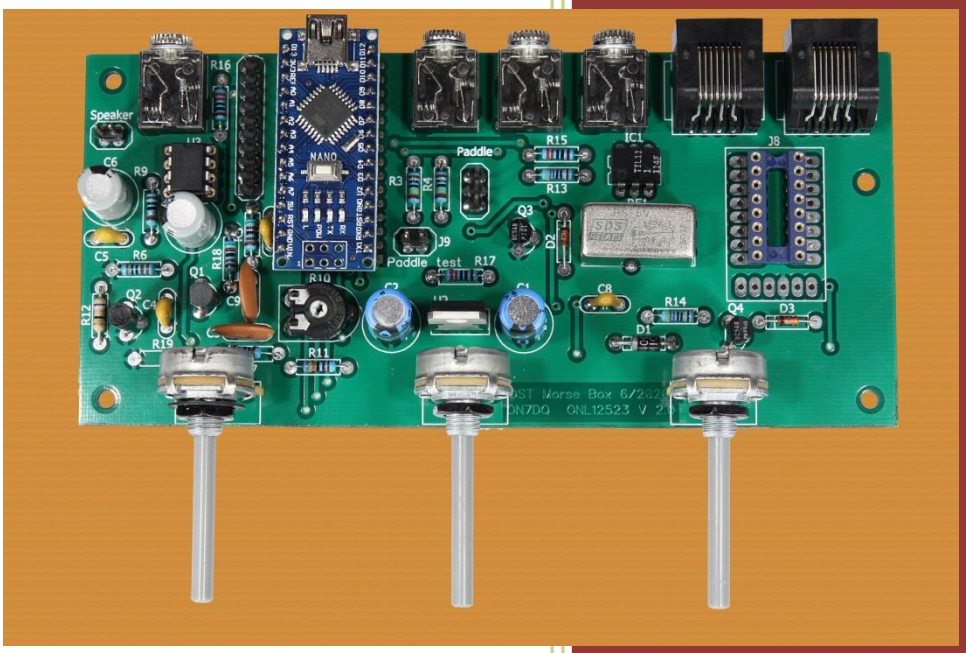
The OST Morse Box



Ostend Radio Club UBA - OST

Save some paper and view this manual on your computer!
Print only these three: the circuit diagram + the placement diagram, and the component list.

Contents

THE OST MORSEBOX - V2.0

Description - Purpose

OST is the local UBA amateur radio club in the city of Ostend, Belgium. The OST Morse Box is a versatile circuit that you can fit between the microphone and the microphone input of any transceiver. It was mainly designed for use with a VHF/UHF transceiver, but can be used for many other purposes.

The main goal is to be able to hold CW practice sessions on the 2m or 70cm band, where the students themselves can reply to the teacher in FM-modulated CW. Another use is to make a few practice QSO's in a safe environment, before "throwing yourself" on the HF bands ...

In addition, the circuit contains a number of extra features, making this a very interesting club-building project.See the block diagram on the next page.

By using an inexpensive Arduino Nano the following functions are available:

- DDS Tone-Generator in software, for a perfectly pure sine wave
- Microphone is switched off while transmitting CW to avoid disturbing background noise
- Automatic activation of the transceiver PTT
- Adjustable DELAY for the PTT, from 0.5 ... 10 seconds
- Variable CW speed of 10 ... 35 words per minute (WPM)
- OLED display to read the set parameters and texts
- Keying with Straight Key, Paddle or a built-in Touch Paddle!
- Paddle polarity settable to NORMAL / REVERSE
- Built-in "Keep Alive" circuit for use with a Power Bank
- Adaptable to all existing transceivers, using the appropriate microphone plugs
  (The basic model is based on the widely distributed RJ-45 connectors)
- Power via the USB connection of the Arduino, via the microphone jack or from an external power supply
- Use as a separate electronic keyer for transceivers without a built-in keyer
- Random CW generator, display characters in the serial monitor and on the OLED display.
- Beacon function, can also be used as a memory keyer (1 memory of 80 characters).
- Additional functions can be set via AT commands via the serial monitor
- Entering and transmitting text via the serial monitor
- Windows program for controlling the OST Morse Box (then no need for the Arduino IDE)
- Built-in TEST function for touch paddle

All files needed for making the PCB are on github*. Make it yourself, order from a local manufacturer, or order them cheaply in China (send a mail to ON7DQ for more info).

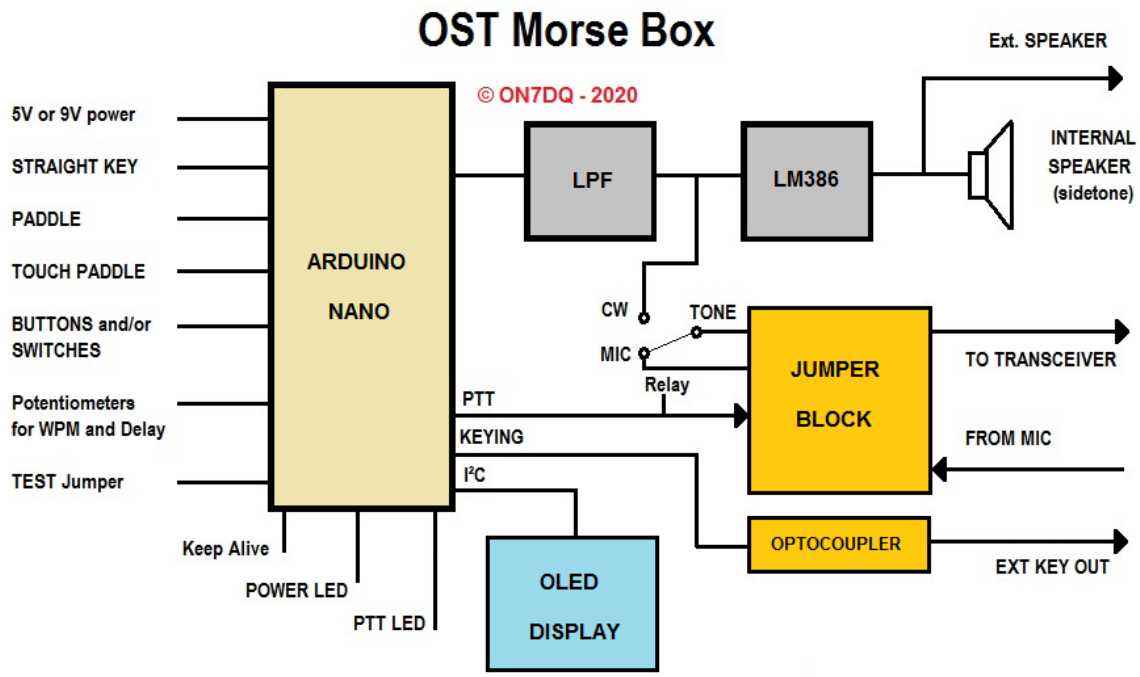
Versatility: not all options are mandatory, those who wish to do so, can partially fill the PCB to obtain only the desired functions. This is clearly indicated in the construction manual.

The software and the development of the PCB layout were done by OST club member Gilbert, ONL12523. Great work and thanks Gilbert!
I just came up with the idea for the Morse Box ... built the first prototype, and wrote this manual.

Good luck in building, and have fun using the OST Morse Box!

* https://github.com/on7dq/OST-Morse-Box **Luc, ON7DQ - June 2020**

Block Diagram – Operation



The Arduino Nano takes on most of the functions:

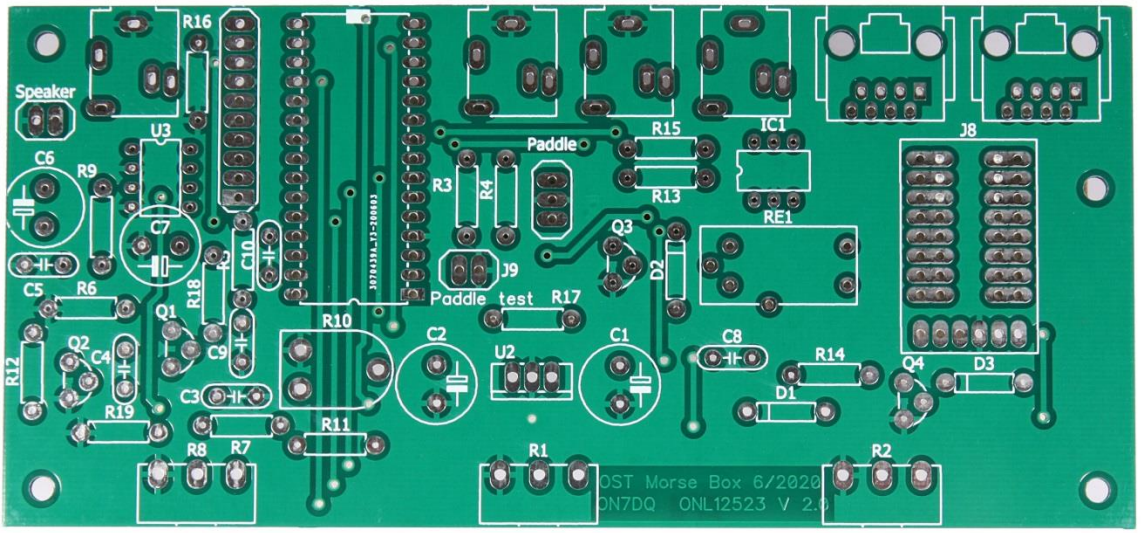
To the left are the inputs: power, key, paddles, buttons and/or switches, potentiometers, test jumper.

To the right all the outputs: the generated sinewave, followed by a Low Pass Filter and LF amplifier, an internal or external speaker, CW/MIC relay, PTT and Keying, OLED display.

At the bottom the “keep alive circuit” for use with a powerbank, and connections for 2 LEDs.

The jumper block ensures the correct interconnection of microphone and transceiver.
Via the optocoupler, an external transmitter can safely be keyed (your own QRP project, beacon transmitter, etc.).

This is the professionally made printed circuit board, before we start assembling!



Building Instructions

For circuit and placement diagrams, see last two pages.

Minimal version

For a first test we build the Arduino and the audio part.

With this you can already make CW tones off-air, and you have built a so called 'Code Practice Oscillator' (CPO). First mount the Arduino Nano, it is the heart of the circuit!

NOTE : it must be a **ATmega328P** CPU, the older 168 won't work.
It may be best to use a socket (two rows of female headers), or if not available, solder the Nano directly with its pins into the PCB, but don't push it down completely, just enough that you can solder them. This way, you don't have to clip off any pins, and If the Arduino would fail, you can still cut all the pins and lash it out one by one so as not to damage the PCB.

We assume you use a pre-programmed Arduino Nano. In the case of an empty Arduino Nano, see Appendix 1 how to program it yourself.

For the audio part (LM386 and peripheral components), we need:
R5 and R18 = 2 x 5.6 kΩ
R6 = 1 kΩ
R9 = 10 Ω

C3 and C9 = 2 x 47 nF
C4, C5 and C10 = 3 x 100 nF
C6 and C7 = Elco 100 µF/10V (or more)

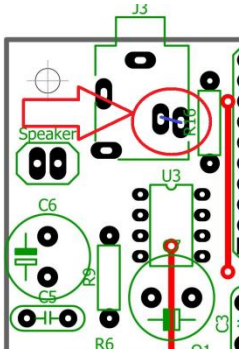
Q1 = NPN transistor BC547B (or equivalent, a 2N2222 would be fine, but note the different mounting!)

R8 = logarithmic potentiometer 10 kΩ, this is the volume control.
If you can't source a log potentiometer, use a linear one, and fit **optional resistor R19** = 2.2 kΩ (or one fifth of your potentiometer value).
Read this article to see why this works: https://www.qrp-labs.com/qcx/qcxmods/qcxvolume

U3 = LM386N (best mounted in an 8 pin socket)

J3 = 3.5 mm stereo jack, pcb mounted (for external speaker)*

* If you don't want to use an external speaker or headphones, you need to bridge the following pads of J3 with a wire (see figure).

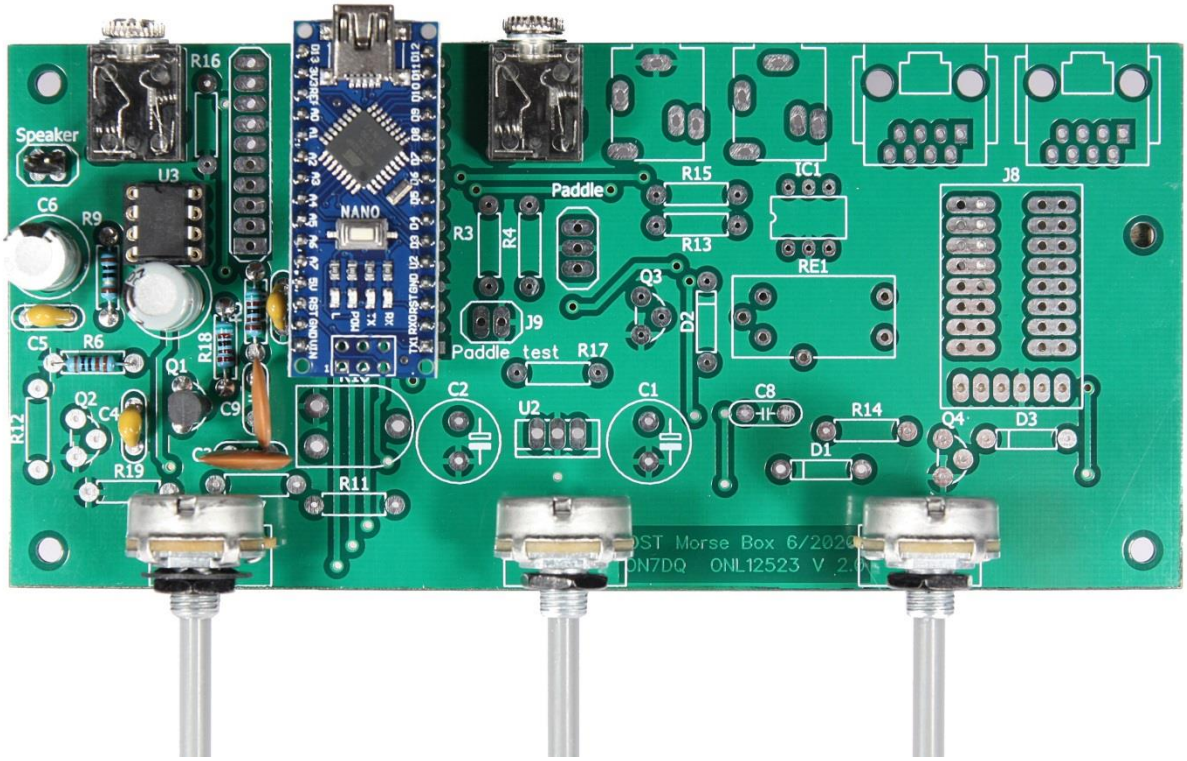


Speaker = a 2 pin header for speaker connection (or solder 2 wires directly to the speaker).
You can use any small speaker of 8 Ohm / 0.2 to 2W.

The R1 (DELAY) and R2 (WPM) potentiometers are optional, but strongly recommended.
Mount the two linear potentiometers R1 and R2, both = 10 kΩ.

If you don't use them, you have to connect the wipers of both to ground.
The software will then choose "default" values when starting up.
Via the AT commands (see Appendix 2) you can still adjust these values. The custom values are stored in the Arduino's flash memory, so you'll start up with these values next time. The potentiometers always take precedence over the values in flash, unless you turn them fully down.

J1 = 3.5 mm stereo jack, PCB mounted (KEY IN), for keying with a Straight Key.
The Paddle and the Touch Paddle are options (see later). When you're ready, you'll have this result:



Now is the time for a first **TEST**!

Depending on your knowledge of the Arduino IDE you have two choices:

With this knowledge: connect the Arduino to the PC, and start the IDE. Make the correct settings for Board and COM port, and open the Serial Monitor. Set it for 115.200 Baud.

Without any Arduino knowledge: connect the Arduino to a power bank, a 5V phone charger or the USB port of a PC (we will only use the 5V supply).

Connect a straight key to J1 (KEY), and set the volume control R8 halfway.

When keying, you should hear a tone, and if using the Serial Monitor, you can read some settings.

NO? → This problem will have to be solved before continuing.
See Appendix 3, or ask some wise person for help.

YES? → Congrats! Continue with the next steps.

Wire the connection to the transceiver

Place the following parts.
R7 should be determined in test, but take 100 kΩ as a starting value. Mount it a bit above the PCB.
If the value needs to be changed significantly, you can cut out the resistor and solder another one on the remaining wires, or remove the wires.
Tip: In the first tests I found these values for the following rigs:
Kenwood TM-733E and Yaesu FT-857D : 100 kΩ
Icom IC-706 MkIIG : 10 kΩ

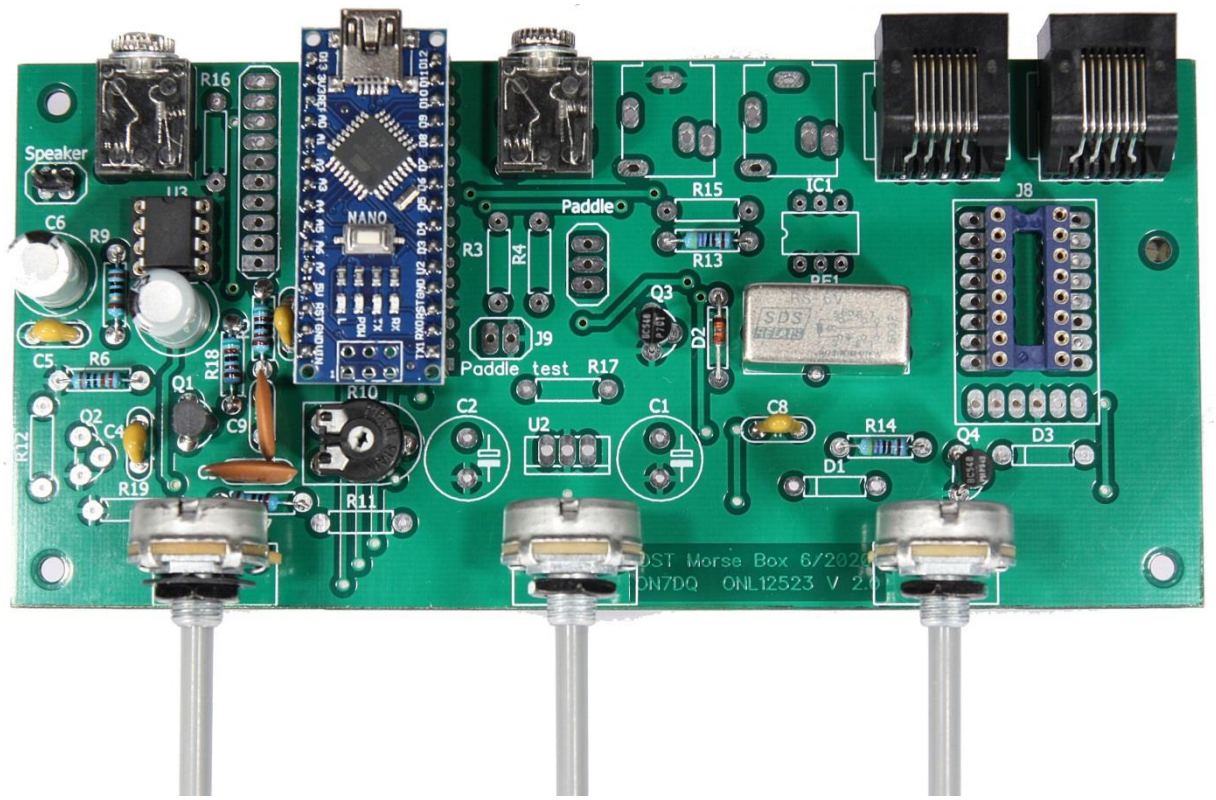
R10 = a 10 kΩ trimmer, with which we will set the modulation (the deviation of the FM transceiver). You can do this by ear, ask a friend for a modulation report, or more correctly by the so-called Bessel-zero method. See Appendix 4.

R13 and R14 = 2 x 4.7 kΩ
C8 = 100 nF

Q3 = NPN transistor BC547B (or equivalent)
Q4 = NPN transistor BC547B (or equivalent)
D2 = diode 1N4148

Relay RE1.1 en 1.2 (= actuator coil + contacts), this is the MIC/CW switchover relay. When idle, this relay will connect the microphone, so that the transceiver can be used for "voice".
If you can't find the correct type of relay, try connecting another small SPDT relay with short wires to the corresponding holes on the PCB.

At this stage, this is what you should see:



And now comes the hardest part ... wire the connections to the transceiver.

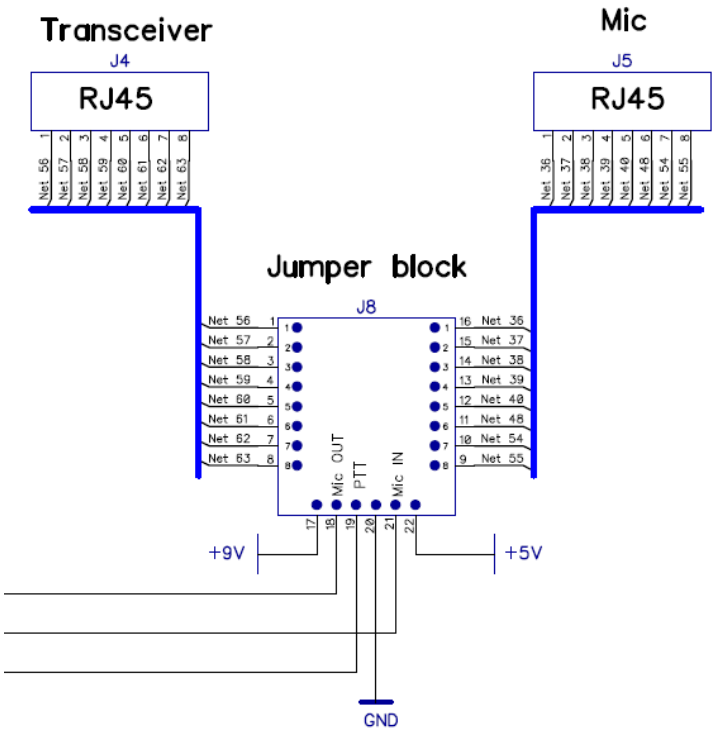
On the PCB is a spacious "switching panel", we called it the **Jumper Block**, with which you can connect basically any transceiver. Carefully study the diagram below.

On the left are the connections to the transceiver, on the right the connections coming from the microphone. At the bottom are the connections to the Morse Box.

Four of these are essential: **Mic In, Mic Out, PTT and GND**.
The others are optional:

+9V is to take power from the transceiver or from an external power supply

+5V is to power any extra homemade circuit, connected at the microphone input, e.g. a desktop microphone, a DTMF keyboard, a Voice Keyer ....



Have a look at all the possibilities, and make your choice!
If you have a transceiver with an RJ-45 microphone jack, it's best to put two RJ-45 connectors (J4 and J5) on the PCB. If you have a transceiver with a classic round microphone plug, you don't have to use those RJ-45's, but instead buy a chassis connector that fits the microphone, and make the wiring directly to the pads on the PCB. To connect the transceiver, make a cord a with microphone plug of the same type as used on the microphone, and from that cord, also make all the connections directly to the PCB. Make sure to put some form of strain relief for the cord.

Alternatively, you could also cut a short Ethernet cable in half, and mount a round microphone plug. On the Morse Box you can then use the RJ-45 connector J4-Transceiver.

J8, the JUMPER BLOCK can be equipped with a 16-pin IC-socket, but it is not a requirement.

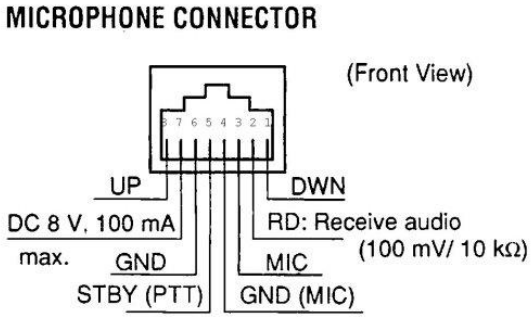
NOTE: look up in the manual of your transceiver which pins have to be looped through, even if not used in the Morse Box.

Typical examples are: power supply for the electret microphone (sometimes the DC is on the MIC line, sometimes it's a separate wire), the UP/DOWN keys, the PF keys (programmable function keys), a CALL button, a DTMF keyboard, ...
We can't possibly discuss this for all devices here, but we can add solutions found by other users in the next edition of this manual. Keep them coming !

This website can help to find the right connections: https://www.qsl.net/g4wpw/date.html
In the next section, please find an example for a transceiver of each of the three major brands: Kenwood, Icom and Yaesu.

Kenwood

Connection with RJ-45 connectors for a **Kenwood TM-733E.**
From the user manual we get the following microphone connections.
Note that the connector is upside down compared to the PCB mounted connector.



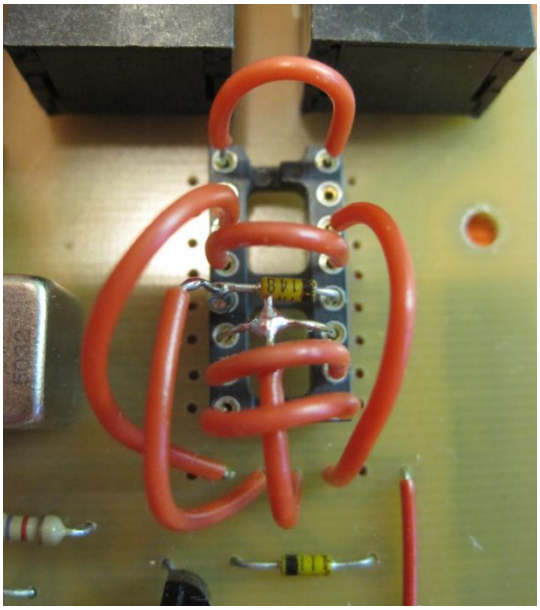
So we make the following table:

Pin 1	DWN
Pin 2	(RX audio)
Pin 3	MIC
Pin 4	MIC GND
Pin 5	PTT
Pin 6	PTT GND
Pin 7	DC 8V
Pin 8	UP

In the Jumper Block, the following connections have to be made, this can be done with small pieces of wire, if you have used an IC socket, or you can also solder all wires directly to the pads on the PCB.
The numbers between parentheses refer to the numbering on the Jumper Block.

- DWN (1) – (16)
- nothing (2)
- MIC (3) Transceiver side, to MIC OUT (18)
- MIC (14) Microphone side, to MIC IN (21)
- MIC GND (4) – (13)
- PTT (5) – (12), via diode D3 = 1N4148, cathode to MIC side (12), and also PTT (5) Transceiver side to PTT (19)
- GND (6) – (11), and also to GND of the Morse Box PCB (20)
- DC 8V (7) – (10)
- UP (8) – (9)

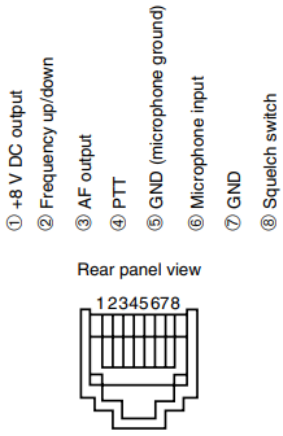
It probably looks a lot more complicated than it really is. If in doubt, ask the designers for assistance! For R7 I took 100 kΩ. When everything is ready, it looks like this:



Kenwood Jumper Block

Icom

Wiring for an **Icom IC-706 MkIIG**. From the manual:

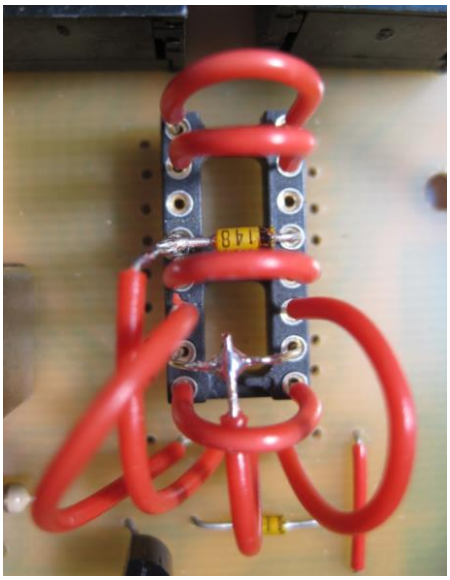


So we need to make the following connections:

- +8V (1) - (16)
- UP/DWN (2) - (15)
- nothing (3)
- PTT (4) - (13), via diode D3 = 1N4148, cathode to MIC side (13), and also PTT (4) Transceiver side to PTT (19)
- MIC GND (5) – (12)
- MIC (6) Transceiver side, to MIC OUT (18)
- MIC (11) Microphone side, to MIC IN (21)
- GND (7) – (10), and also to GND of the Morse Box PCB (20)
- nothing (8)

In my tests, I had to reduce R7 to 10 kΩ, to get an acceptable modulation.

This is how it looks for Icom:

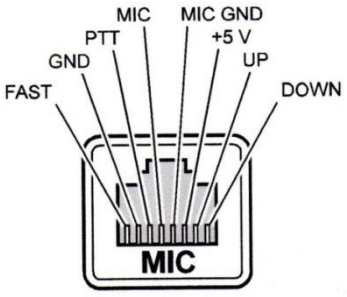


Icom Jumper Block

Yaesu

Here we take a **Yaesu FT-857D** as an example. MIC connections are as follows
Again, the connector is upside down from normal
convention, so we make a table:

Pin 1	DWN
Pin 2	UP
Pin 3	+5 V
Pin 4	MIC GND
Pin 5	MIC
Pin 6	PTT
Pin 7	PTT GND
Pin 8	FAST

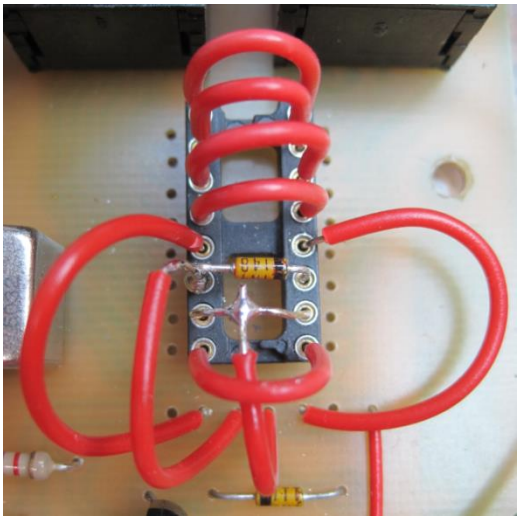


BEWARE: Yaesu specifies a maximum current of only 10 mA fort he +5V line, this is **NOT OK** for the Morse Box. Check carefully if you have another Yaesu model.

Connections are:

- DWN (1) - (16)
- UP (2) - (15)
- +5 V (3) - (14)
- MIC GND (4) – (13)
- MIC (5) Transceiver side, to MIC OUT (18)
- MIC (12) Microphone side, to MIC IN (21)
- PTT (6) – (11), via diode D3 = 1N4148, cathode to MIC side (11), and also PTT (6) Transceiver side to PTT (19)
- GND (7) – (10), and also to GND of the Morse Box PCB (20)
- FAST (8) – (9)

R7 was left at 100 kΩ.
And this is how ik looks for Yaesu:



Yaesu Jumper Block

Options

Each of the following options can be used independently, pick your choice!

OLED Display

A small OLED display of 128x64 pixels is controlled via the I²C bus. You can read the set DELAY and WPM, as well as the random characters of the practice generator.
To save memory, an ASCII library is used, so we can only place text on the display.

The connection is very simple, you only need four wires: **GND, Vcc, SDA** and **SCL**.

[Note: Vcc for the display is 3.3V, but can also tolerate 5V. The I²C logic of the Arduino is also 5V, but considering we only pull the I²C lines "low" or let them "float", there will not be any 5V on the I²C lines of the display, so there is no problem.]

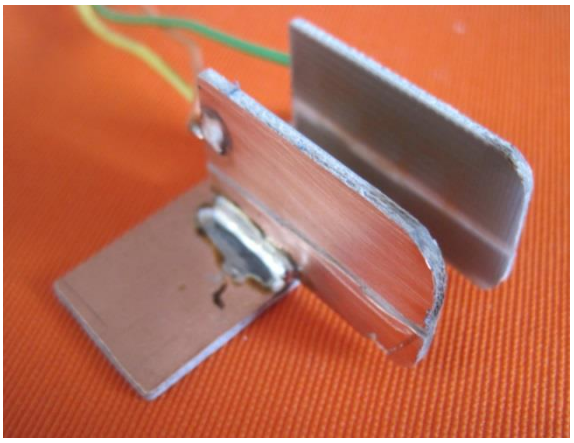
You can solder the wires directly to the display, or use male and female headers of your choice. Connect the four wires to the J6 header, which is located between the Arduino Nano and the EXT Speaker jack J3. PIN 10 is closest to the back edge of the PCB.

PIN 1 to SCL
PIN 2 to SDA
PIN 6 to Vcc
PIN 7 to GND

Normal paddle

Mount a 3,5 mm stereo jack at J2, that's all you need to start "paddling" away …

Touch paddle



Make a touch paddle like in the picture.
It consists of three pieces of circuit board. Two of those form the contacts, isolated from ground by removing a 2 mm strip of copper. Make it into a personal design, but make sure there is some ‘ground’ near both contacts, this forms a small capacitor which is needed to make a ‘capacitive touch sensor’ work.

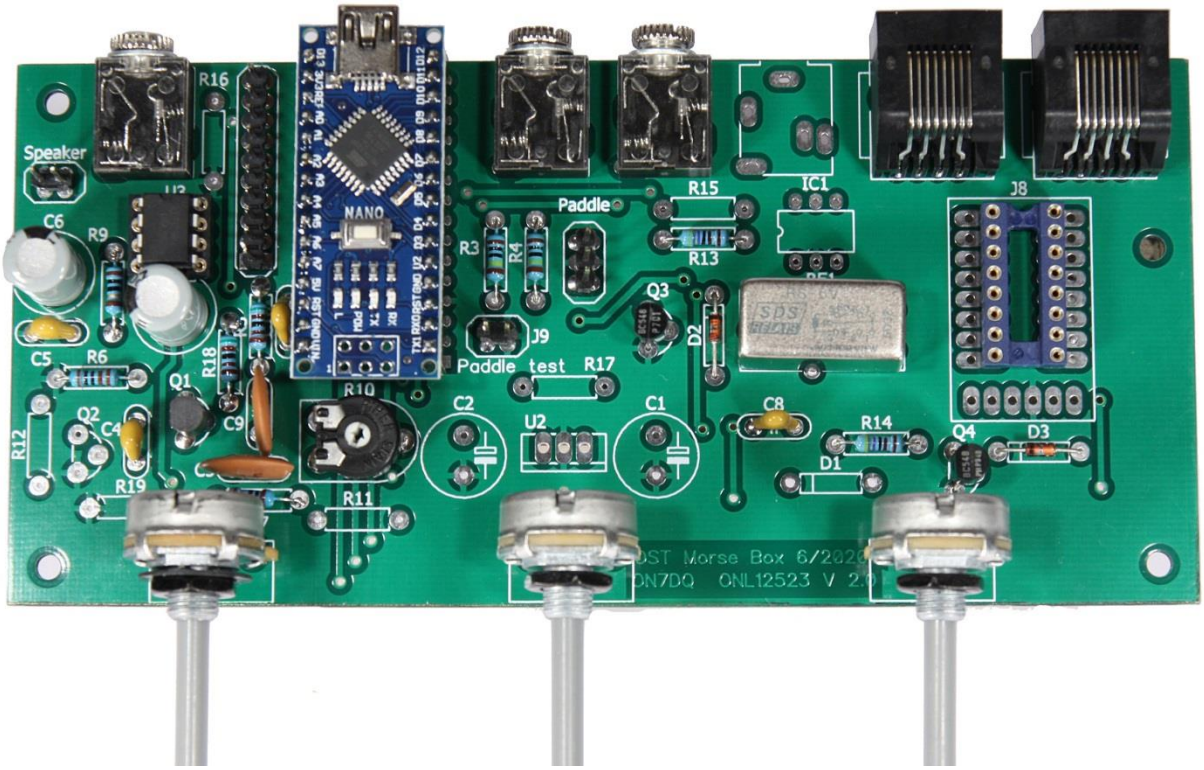
Mount resistances R3 and R4, both = 1MΩ.

How does it work? : the Arduino puts +5V on the output pin D9, and via the two resistors the small capacitors will be charged. Time for charging is measured, and when not touching the paddle, this time will be very short. When touching the paddle, this time will be longer, so then there is ‘contact’.

Connect the touch paddle with a 3-pin header, labelled “Paddle” on the PCB (it’s next to R3/R4), or solder wires directly into the PCB. The middle pin is GND. You’ll have to find out for yourself where the DIT and DAH contacts are, depending on whether you are left- or right-handed (you can also change this in software via an AT+ command, see Appendix 2).

For verification, you can also activate the test function for the touch paddle. See the ”Operating Manual” on page 21.

After these options you have the following result, external parts are not shown:



DC input and voltage regulator

WARNING: WHAT FOLLOWS CAN BE HARMFUL TO YOUR TRANSCEIVER.
USE THIS OPTION AT YOUR OWN RISK!

If you want to power the circuit from the transceiver's microphone line, you first need to look at two things:

- Is there a suitable voltage available on this connector, and if so, what voltage is it ?
- What is the maximum current you can get from this connection? You need up to 70 mA.
-

E.g.: in the Kenwood TM-733E 8V is available with a maximum of 100 mA.

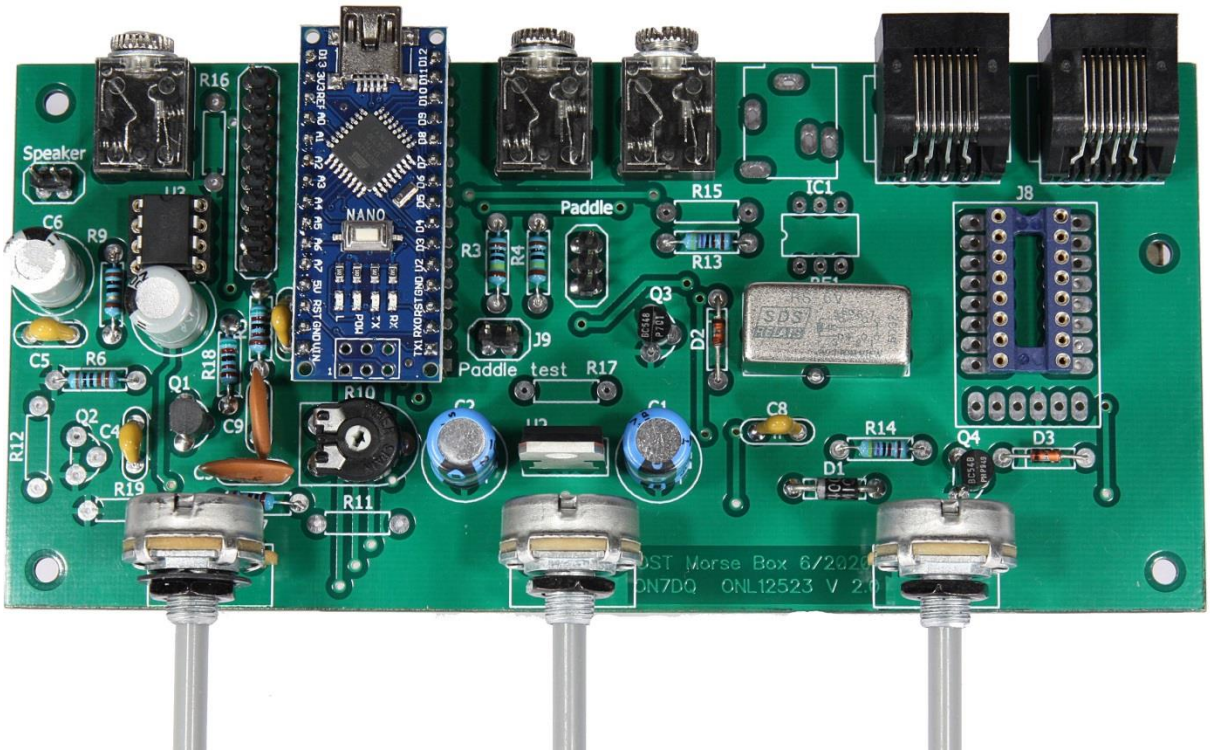
If you think it is safe, and the voltage from the transceiver is e.g. **+13.8V**, you can mount the voltage regulator U2 = LM7808. A heatsink is not necessary. When the voltage is already **8V** , you may omit the regulator. Replace it by a jumper wire from the input to the output pad of the LM7808, or even better, put a small 100 mA fuse there (normal fuse or polyfuse).

In all cases you will need diode D1 = 1N4007, and the elco's C1 = 100 µF/25V (or more) and C2 = 10 µV/16V (or more).

Make a connection in the jumper block from the +9V line (pin 17) to the appropriate pin on J4 – Transceiver side.

Note: This option can also be used to connect a separate mains power supply, for example a 12V wall wart or similar, or possibly a direct connection to the 13.8V power supply for the transceiver.
In this case, make sure that the power cord is secured. Put the voltage on pin +9V (17) in the Jumper Block, but do not connect it to the transceiver or microphone pins!

With the regulator mounted, this is what it looks like:



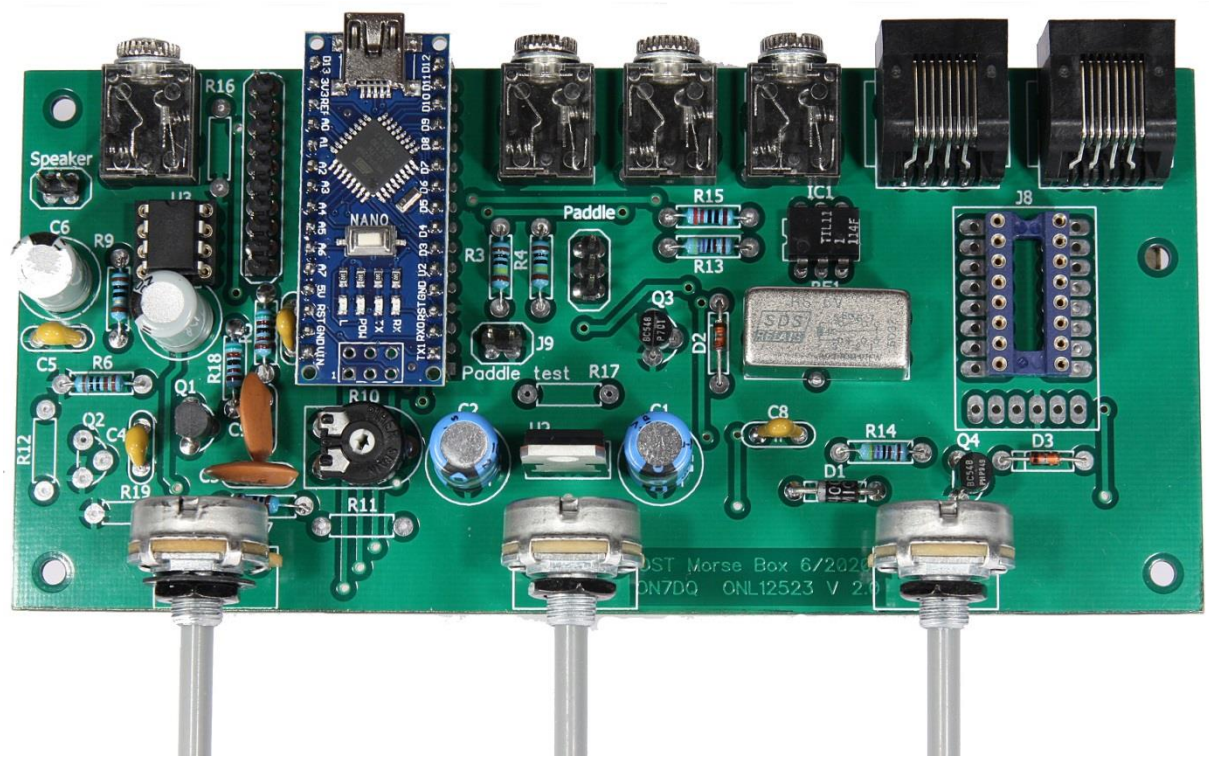
Transceiver keying output

The Morse Box can also be used as a basic electronic CW keyer, e.g. to drive a home built QRP transceiver, beacon transmitter, etc.

For this we need : R15 = 220Ω, IC1 = optocoupler TIL111, CNY17, or equivalent (in a 6-pin socket if you like).

Also mount J7 = the keying jack, a 3.5 mm stereo model, with KEY OUT = TIP.

Look how it grows, we're almost there!



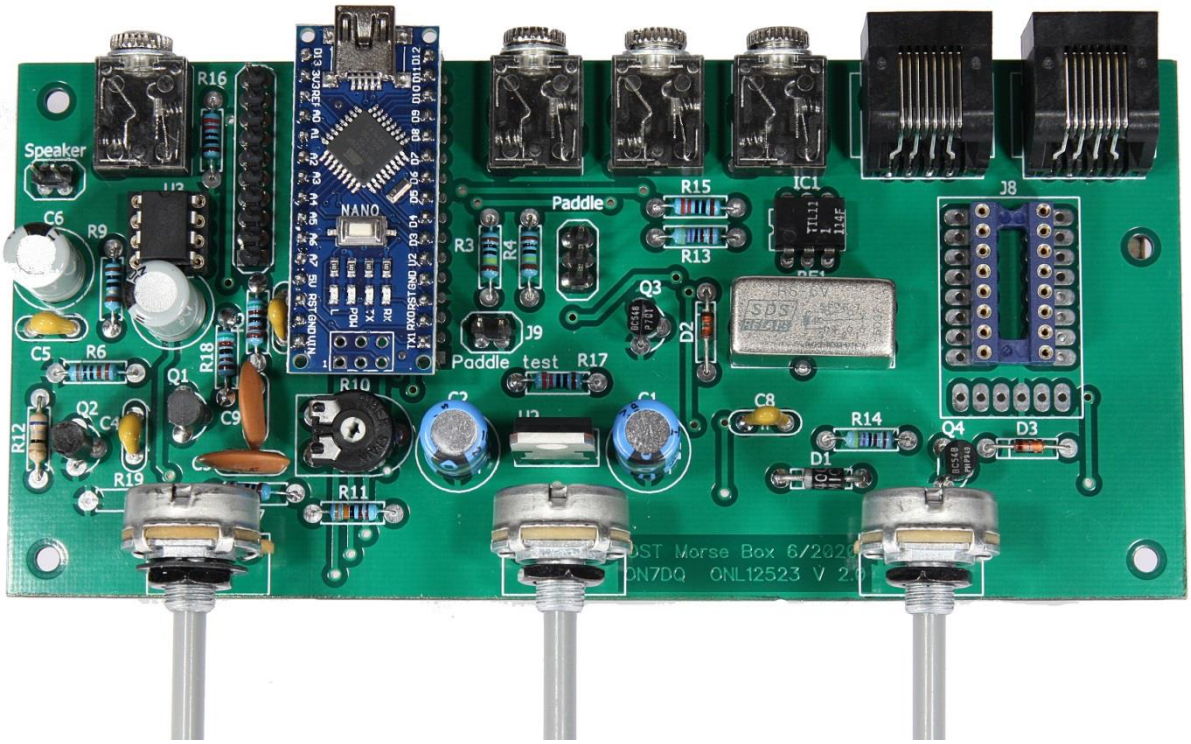
Keep Alive circuit

When powering the circuit from a so-called power bank, it can happen that the power bank shuts itself down if the current is too low. The Morse Box uses only some 10 mA when idle.

The Keep Alive circuit draws a current of 80 mA every 10 seconds for a duration of 300ms, this should be enough for most power banks to keep them 'alive'.
For this purpose, mount R11 = 3,3kΩ, Q2 = BC547B (or equivalent) and R12 = 68Ω.
Note: Use this option only when necessary, the circuit may cause a light tick in the speaker.

Better decoupling of the LM386 power line (series resistor and extra cap) could probably remove the ticking, but then, it's also an indicator that your box is still powered if you forget to unplug the power bank.

And with this last option, the board is completely finished!



Extra “bells and whistles”!

Buttons and switches

Inputs **A1, A2** and **A3** on the Arduino where still available, so we added a few handy tricks.
All connections are made on pin header J6.
Every function needs a pushbutton to ground for a “one shot” operation, or a toggle switch for continuous operation. See diagram further below, and the Operating Manual on page 21.

A1. Random Morse generator (J6 pin 5)

With a push button from line A1 to GND, a group of CW characters is generated at every push of the button. These can be letters, numbers and punctuation marks or so-called "prosigns".
The number of characters is also variable, to keep it a bit exciting.
If you wish to have more groups, you can hold down the push button, or you can mount a toggle switch in parallel.over the push button contacts.

HINT: There are handy switches that combine a push button with a toggle switch. In one direction the lever returns (= push button), in the other direction the lever remains in position, e.g. this model we found at https://reichelt.de/ :

item-no.: MS 500D
Toggle switch, 1-pin, 6 A - 125 V AC, on-off-(on)

Type:	
Miniature tumbler switch	
Design:	ON - OFF - (ON)
Pole number:	1
Connection:	
Soldered/plug connection	
Load limit:	125 V AC 6 A
Mounting:	M6x0.75
Boring aperture:	6.2 mm
Material:	Fine silver contacts
Length:	13 mm

You can check the characters in the Serial Monitor, and on the OLED display.
NOTE: During the exercise, the PTT is also activated! So turn off the transceiver, or provide a switch to interrupt the PTT if you only want to practice locally, but still want to make a phone QSO with the transceiver. With the switch you have to interrupt line (19) in the Jumper Block.

A2. Beacon or Memory Keyer (J6 pin 4)

Same operation as A1.
Place a push button for one-time transmission of a beacon text of up to 80 characters.
Place a switch (in parallel to the push button) to transmit the text continuously, with a repetition time set with BTIME via the Serial Monitor. The text can only be "programmed" via the Serial Monitor, so you can't "key" it in. However, you don't need the Serial Monitor to transmit the text. See Appendix 2 for the corresponding AT commands. There is also a Windows program to do the same.

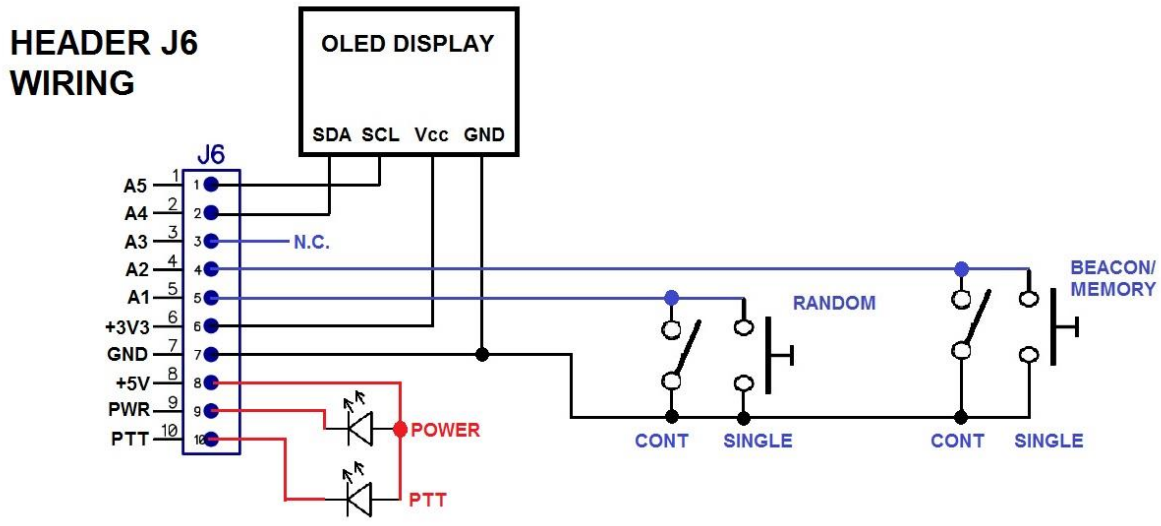
A3. (J6 pin 3)

Not assigned for now. If you have a good idea for its' use, let us know!

LED's

There are two more connections on J6, for 2 LED's.
Mount the necessary resistors of 220 Ω (or larger) on the PCB at R16 and R17.
The LED's connect with the anode to the +5V on **J6 – pin 8**
A **POWER LED** connects with the cathode to **J6 – pin 9**
A **TX LED** (PTT) connects with the cathode to **J6 – pin 10**

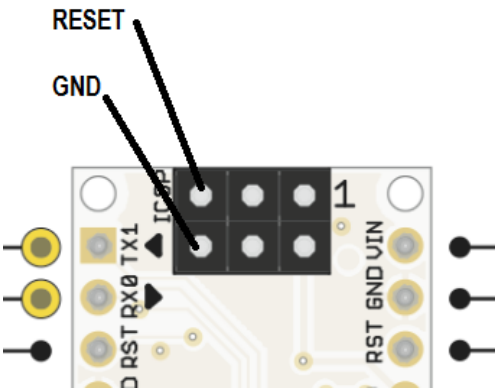
The following diagram shows all connections that can be made at J6.



Reset button

You can reset the Arduino with the button on the Arduino itself, but after fitting it in an enclosure, the button is no longer accessible.

If you think you need it, you can provide an external reset button for the Arduino. The reset line is available on pin 5 of the ICPS header of the Arduino Nano, see figure. GND is available on pin 6.



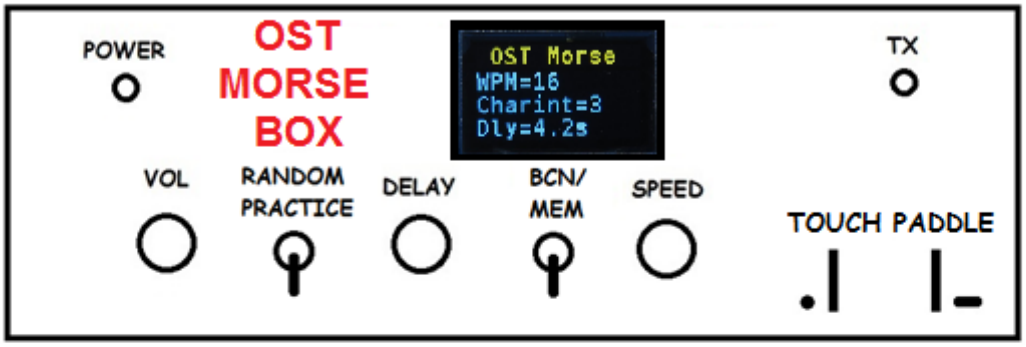
The Arduino can also be reset by interrupting the power supply, or by restarting the Serial Monitor, and also with a Reset button in the Windows program.

Boxing the Morse Box ...

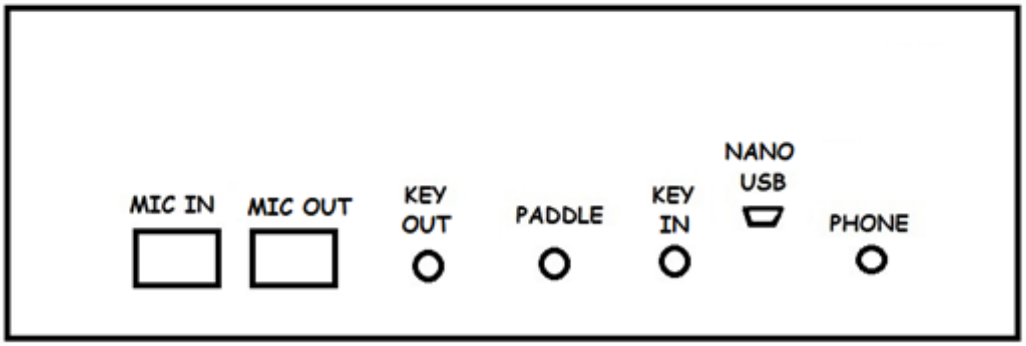
We leave it up to the you to put the Morse Box into a nice cabinet. Find a commercial one, or make one yourself.

Because everyone will have a different purpose for this project, there is no prescribed way of doing this, we only give a few basic ideas here.
Here's an 'artists impression' of what it could look like, front and back...

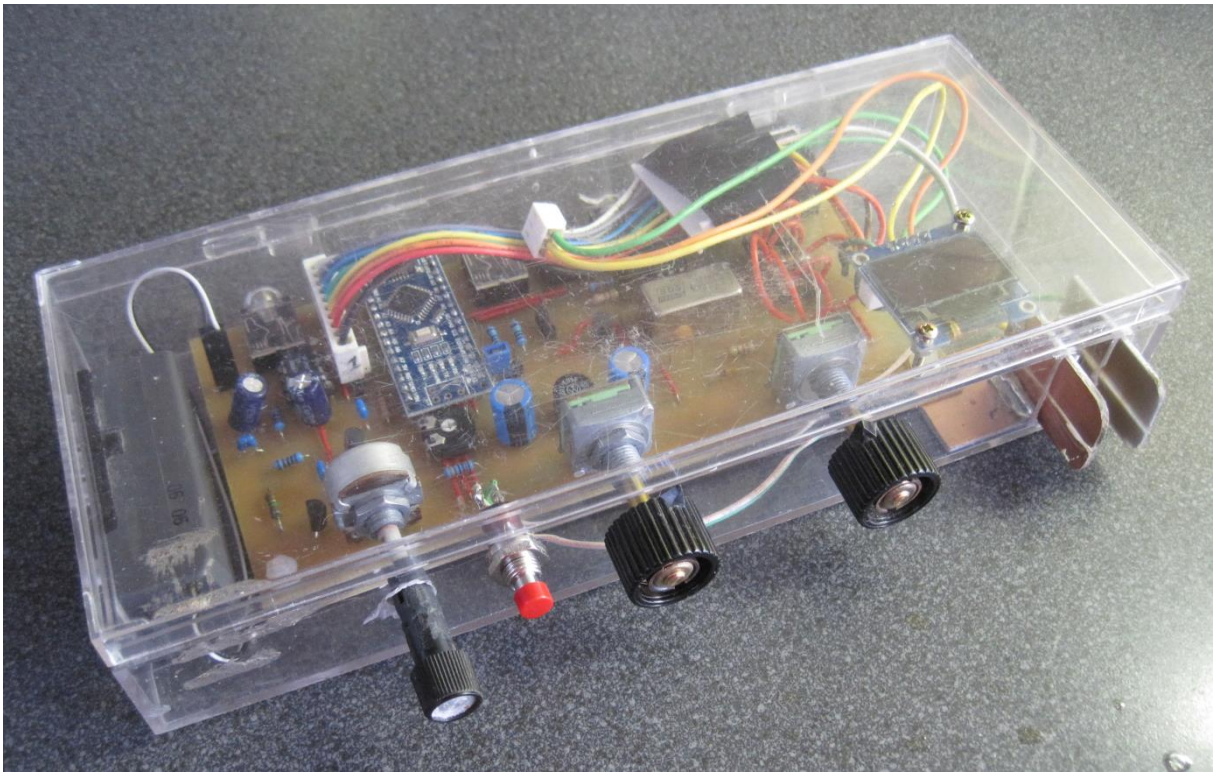
FRONT



BACK

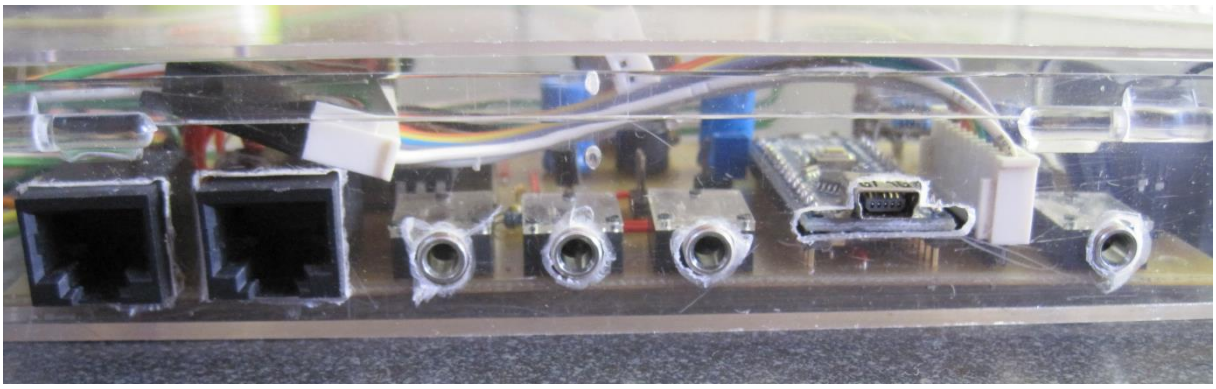


This is how I built the prototype into a plastic cigar box.



It doesn't look nice or professional, but it is very 'didactic', it's easy to view the circuit from all sides … WYSIWYG!

The connectors at the back:



Challenge: did you make a nice cabinet?

Send me (ON7DQ) a picture, I will post it on my blog, and the best one may make it into this manual

Parts list

* Some parts need to be determined in test, or chosen according to the option, see building manual. Not in this list : various pins or pinheaders + wires to connect external things, materials for the touch paddle, etc. In the empty left column you can tick off what you have already assembled.

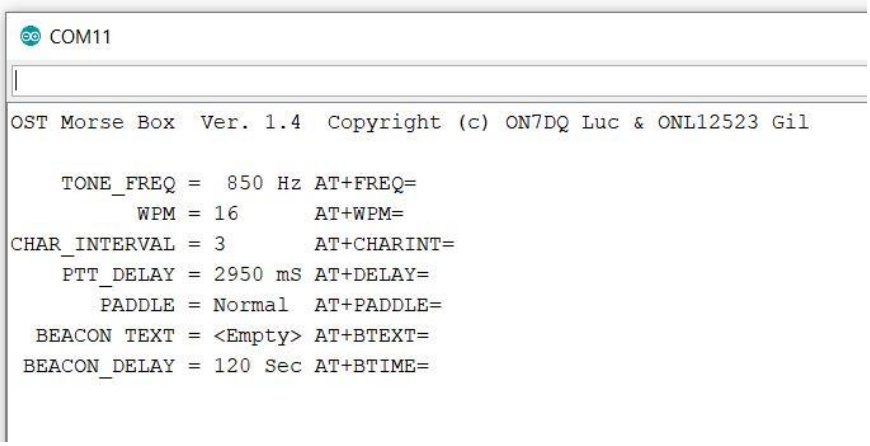
	Ref	Value/Type	Notes		Ref	Value/Type	Notes
	C1	100uF			R1	Potm 10k lin	Delay
	C2	10uF			R2	Potm 10k lin	WPM
	C3	47n			R3	1M	
	C4	100n			R4	1M	
	C5	100n			R5	5k6	
	C6	100uF			R6	1k	
	C7	100uF			R7 *	10k of 100k	See text
	C8	100n			R8 *	Pot 10k log, or 10k lin	See text
	C9	47n			R9	10	
	C10	100n			R10	Trimmer 10k	Flat type
					R11	3k3	
	D1	1N4007			R12	68	
	D2	1N4148			R13	4k7	
	D3 *	1N4148	PTT diode		R14	4k7	
					R15	220	
	IC1	TIL111			R16	220	
					R17	220	
	J1	Jack 3,5 mm	Key		R18	5k6	
	J2	Jack 3,5 mm	Paddle		(R19 *)	(2k2)	See text
	J3	Jack 3,5 mm	Speaker ext				
	J4	RJ45 plug	Transceiver		RE1	SDS-Relay 6V	
	J5	RJ45 plug	Mic		SPKR	Header 2 pins	Speaker
	J6	Header 10p	OLED, Buttons, LEDs		U1	Arduino Nano	
	J7	Jack 3,5 mm	Keying OUT		U2 *	LM7808, or jumper, or fuse	See text
	J8	Jumper block	(16p IC socket)		U3	LM386N	
	J9	Header 2 pin	Paddle TEST		OLED	Display 0.96”	128X64
					LED Green	Power ON	
	Paddle	Header 3 pin	Touchpaddle		LED Red	PTT ON	
					Misc.		
	Q1	BC547B	Or		USB-A naar	Mini-USB	Cable
	Q2	BC547B	equivalent		RJ-45	Ethernet cable	50cm à 1m
	Q3	BC547B	NPN		5V Supply	Powerbank or	other supply
	Q4	BC547B	transistor		Buttons,	Switches	See text

Operating Manual

In this manual, we assume that you have installed all options, and know how to use the Serial Monitor.

At start-up, by connecting the power supply or after pressing the reset button on the Arduino, the OLED screen will briefly show the version number for 1 second, then the set parameters.

The Serial Monitor displays the following settings.



The meaning of these values (see also Appendix 2):

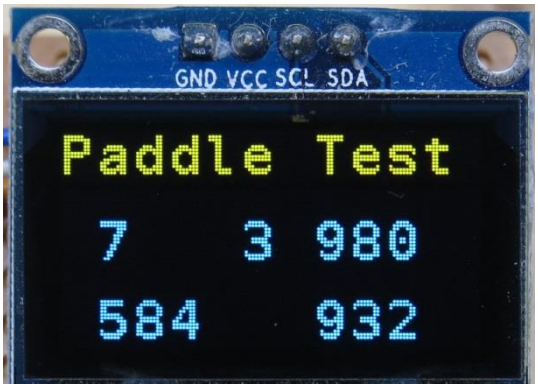
- TONE_FREQ — The sidetone frequency as stored in the EEPROM.
- WPM — The setting of potentiometer R2. When no potentiometer is connected, the value stored in EEPROM is shown.
- CHAR_INTERVAL — The character spacing stored in the EEPROM.
- PTT_DELAY — PTT delay in milliseconds, as set by potentiometer R1. When no potentiometer is connected, the value stored in EEPROM is shown.
- PADDLE — The setting stored in the EEPROM.
- BEACON_TEXT — The beacon text stored in the EEPROM (max. 80 characters)
- BEACON_DELAY — The beacon repeat time in seconds, as stored in the EEPROM.

On the OLED display, only the following parameters are shown:

- WPM Setting of potentiometer R2. When no potentiometer is connected, the value from the EEPROM is shown.
- Charint The character spacing, as stored in the EEPROM
- Dly PTT delay in seconds

Paddle Test Mode, after restart with a jumper at J9 (= grounding pin D2)

This is a test routine to check the operation of the capacitive sensors.



The second line shows:

- First number Time in milliseconds from the capacitive test routines
- Second number Value when touching the left paddle
- Third number Value when touching the right paddle

The third line shows:

- First number Setting of WPM potentiometer R2, value = 0 to 1010
- Second number Setting of DELAY potentiometer R1, value = 0 to 1010

When using the Serial Monitor, it will show the same values.

Don’t forget to remove the jumper after the test, and reset the Arduino.

Straight key and Paddle operation

When pressing the straight key, the audio relay is turned on, the PTT line is drawn low and the audio tone starts. The PTT timer is also loaded here with the PTT_DELAY value.
This also means that if the key remains pressed (by you … or by your cat!), the transmitter will only be switched off after the PTT_DELAY has timed out.

The Paddle Keyer creates dots and dashes according to the set WPM value.
The sent dots and dashes are shown on the Serial Monitor.
Read more about the timing of the Morse code here: Morse World

Random code practice

The OST Morse Box also contains a handy CW trainer!

Connect A1 to GND with a push button or a switch. This pin is available on header J6.
Remember that during the exercises, the PTT is also activated, in order to be able to provide on-air practice sessions.

If that is not what you want, switch off your transceiver, or put an additional switch in the PTT line towards the transceiver (see construction manual).

A press on the **push button** generates one exercise at a time. It is best to turn the PTT_DELAY to a minimum, because the next exercise can only start when PTT_DELAY has timed out.

Using a **switch**, you can generate continuous random code.
A random string is created and transmitted with the set WPM values. The letters are immediately shown on the Serial Monitor. On the OLED display they only appear after the exercise is completed.
The PTT_DELAY setting determines the time between successive exercises.

Don't forget to disengage the switch if you've practiced enough, especially if you're transmitting.
You can always interrupt the exercises by touching the key or paddle.
After about 30 seconds, the info screen is displayed again on the OLED display.

Random code is composed as follows:

- Two letters, one digit, three letters, sometimes followed by /P
- Random characters, between 3 and 9 letters
- Random numbers, between 2 and 7 digits
- Three letters, a punctuation mark and a letter
- One of these three prosigns:
  # = <AR>
  $ = <BT>
  % = <SK>

Beacon mode / Memory keyer

Connect A2 to GND with a push button or a switch. This pin is available on header J6.

The beacon message of up to 80 characters is stored in EEPROM.
The default text is an empty string, and nothing will be transmitted.
If a text is programmed, It will be transmitted with the set WPM values.

With a **pushbutton** at A2, the beacon text is transmitted once.
So you could use this as a basic memory keyer with 1 memory, e.g. a CQ call.
Here again, it is best to turn the PTT_DELAY to minimum.

With a **switch** at A2, the beacon text will be transmitted repeatedly.
The beacon timer is loaded with the BTIME value from the EEPROM. This time in seconds counts from ‘start to start’ of the beacon text, so the pause between transmissions is:
BTIME – message duration.

There is no time limit or a limit on the number of transmissions, so you are responsible to comply with the regulations for unmanned stations in your country!

The OLED display and the Serial Monitor show the characters when the transmission is going on.
You can interrupt the transmission at any time, just by touching the key or paddle.

After about 10 seconds, the info screen is displayed again on the OLED display.

Serial mode

In the Serial Monitor, in addition to the AT commands, you can also type some regular text, or paste it from a text file. This text is then sent to the Morse Box via the USB Serial Line. End your text with a CR/LF (the <Enter> (key).
The characters are sent out immediately with the set WPM values.

The <UP> arrow key allows you to retrieve all previously typed text (as long as you don't close the Serial Monitor). This is a kind of memory keyer, with **unlimited memory**!

On the OLED display and in the Serial Monitor, the characters are shown when transmitting.
You can interrupt the transmission at any time, just by touching the key or paddle.

After about 10 seconds, the info screen will be restored to the OLED screen.

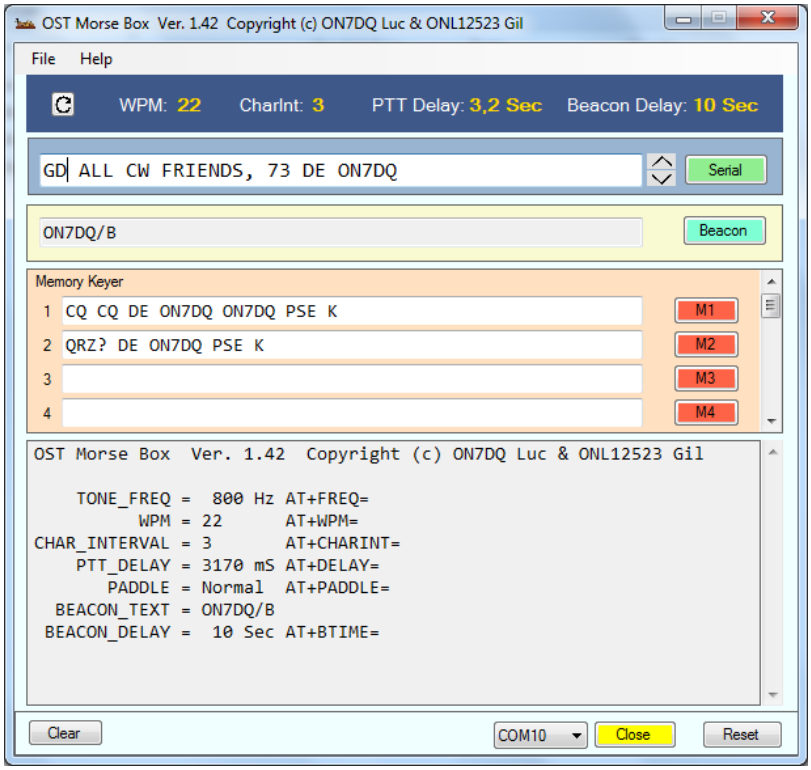
Audio/PTT control

In the Random, Beacon and Serial modes, the audio relay and the PTT line are switched ON at 250 ms before the sidetone is generated. The end of the tone is always according to the set PTT_DELAY.

Windows Program

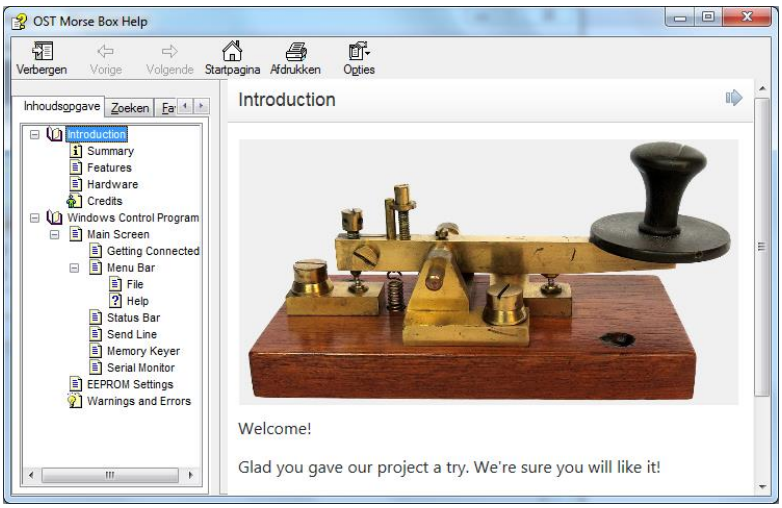
Gilbert has created a beautiful Windows program, which allows you to control the OST Morse Box. So you don't need to know anything about Arduino or programming.

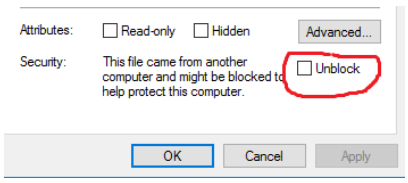
The program has it's own Serial Monitor, so anywhere in the manual where Serial Monitor is mentioned, you can use this program as well.
Connect the Morse Box to the PC before starting the program.



Operation of the program is mainly self-explanatory, see the built-in **Help** (**F1**). Right-click on the item **Introduction**, then **Open All**, to see the full Table of Contents.

Note for **Windows10**: if the contents of the Help remain empty, find the **Morsebox.chm** file, Right-click on it, go to Properties, and check the option "**Unblock**" under Security.





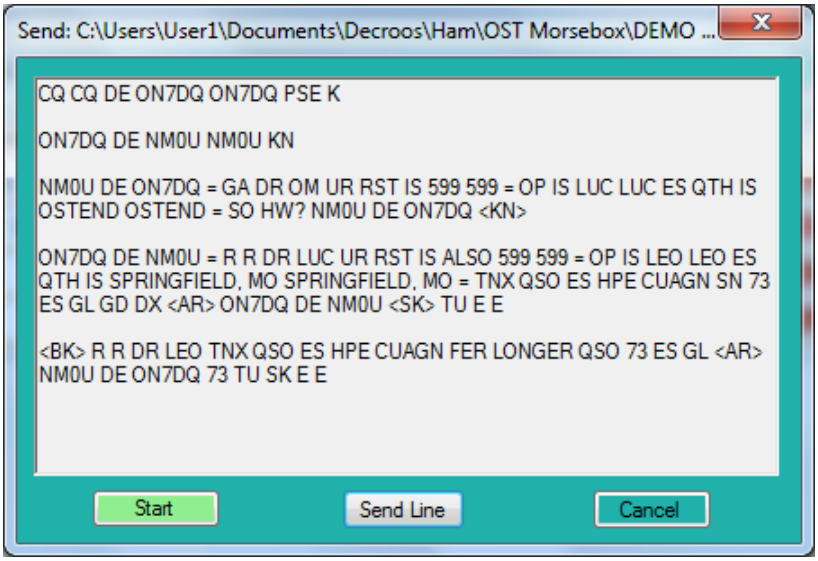
Here is a summary:

Select the right **COM port** at the bottom and click **Connect** (when connected the button shows **Close**)
At the top appears the status of the Morse Box: WPM, Charint, etc. Below, enter text or AT commands in the text box and hit **<Enter>** or the **Serial** button**.** Recall older text with the **<UP Arrow>.**

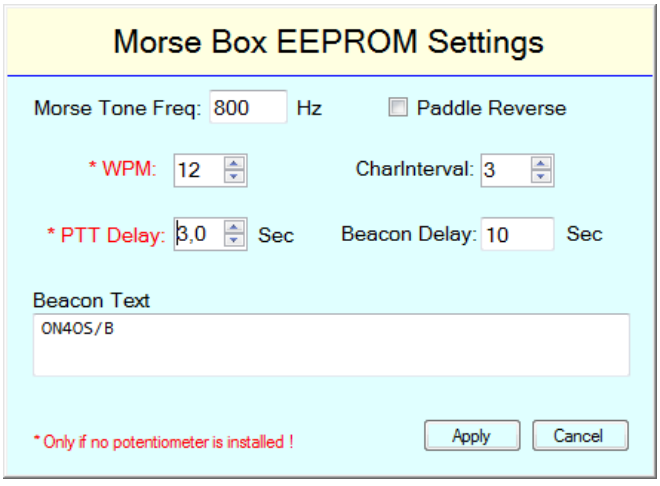
Then follows the fixed memory/beacon text, that you can transmit once with the **Beacon** button (changing the text is only possible via the EEPROM Settings).

Below that, you have a **Memory Keyer**, which can only be used from the Windows program.
Of the 20 available memories, 4 are shown. The memories in use are automatically saved when leaving the program. They can also be **Save**d and **Load**ed manually, via the **File** menu (in **.ini** files), to store different sets of memories for ragchewing, contesting, SOTA, etc …

Via the **File** menu you can load a text file to transmit (CW exercise, club bulletin, etc).
Start sends all the lines, or you can send it line per line.



File > EEPROM_Settings allows you to adjust the parameters.



The **Clear**, **Close** and **Reset** buttons do exactly as they say.
File > Exit allows you to exit the program.

Appendix 1: Programming the Arduino Nano

Here is a brief summary of what you need, a more detailed explanation may be given in your local radio club, as part of a “homebrew session”. Ask your local *geek / nerd / guru / wizard* ;-)

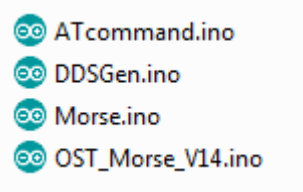
The program that allows you to program an Arduino board is called the **IDE: Integrated Development Environment**. A whole mouthful that simply means that you have all the tools in one program.

The Arduino IDE exists for Windows, Linux and MacOS but we will discuss only the Windows version. Download the most up-to-date version here: https://www.arduino.cc/en/Main/Software , and install the program. Let it install the Arduino Drivers as well.

Connect the Arduino Nano in the Morse Box to a USB port.

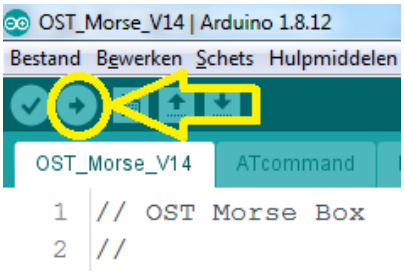
The Morse Box project uses some additional libraries. If you don't have them yet, you can install them with the Library Manager. Go to Tools > Manage Libraries ...
In the search field (top right), enter the name and have it installed. You need these libraries : CapacitiveSensor, SSD1306Ascii and EEPROM.

Make sure you have a folder with the four necessary files for the Morse Box project.
This could be your project folder: `\My Documents\Arduino\OST_Morse_V14`
In that folder you will put these 4 files:



In the IDE, you open only the file **OST_Morse_V14.ino**, the other files are automatically opened as well. Now you have to do some settings from the Tools menu:

Board	:	“Arduino Nano”	
Processor	:	“ATmega328P (Old Bootloader)”	(this is true for most Chinese Nanos)
Port	:	COM10	(Look up the port using Device Manager)



Finally, all you have to do is click on the UPLOAD button (arrow symbol, see picture above), and after a few seconds your Arduino Nano is programmed!

Appendix 2: Using the Serial Monitor and the AT Commands

Connect the Arduino to a PC which runs the Arduino IDE.
Open the Serial Monitor, and set the speed to 115200 bps. All the text you type in the textbox on top, will be sent to the Morse Box when you press ENTER.

The following commands allow you to make changes in the settings, which are then stored in the flash memory for a subsequent boot.

AT+FREQ=800	**set tone frequency**
AT+WPM=16	**set wpm speed**
AT+CHARINT=3	**set character interval**
AT+DELAY=2000	**set PTT delay**
AT+PADDLE=REVERSE	**set reverse paddle pins**
AT+BTEXT=xxxxxxxx	**set BEACON text**
AT+BTIME=100	**set BEACON delay**

The following values can be set

FREQ :	400 Hz to 1500 Hz
WPM * :	10 to 35
CHARINT :	3 to 10
DELAY * :	500 to 10000 ms (0.5 to 10 seconds)
PADDLE :	REVERSE or reverse, type anything else = normal
BTEXT :	Beacon/memory text, max. 80 characters (type without using quotes) The text can also be emptied by entering AT+BTEXT=
BTIME :	Delay between beacon transmissions, 5 to 30000 seconds (8.33 hours)

* Note that the values for WPM and DELAY are written to the flash memory, they also appear briefly on the OLED display. But if you use potentiometers, and these are not put at zero (which corresponds to a wire to GND), then the values of the potentiometers are used.

Appendix 3: Troubleshooting

The Morse Box does not start up

Did you mount potentiometers R1 and R2?
If not, did you connect the wipers of both potentiometers to GND?

The circuit draws too much power, or no sound comes out of the speaker

An obvious one … did you turn up the volume?
Have you perhaps put IC **U3 (LM386)** in reverse?
Is it really an LM386 and not an LM380? (it's easy to make this error, and the LM380 is also an audio amplifier, but with another pinout).
Are there no pins from the IC bent when inserted, so they don't make contact?
Have you mounted the jack J3 (EXT Speaker/Headphones)? If not, did you connect the two designated pads of J3 with a wire?
Without LM386 inserted, make sure you're measuring the 5V power supply on pin 6, on all other pins you should measure nothing (DC voltage).
Make sure C5 isn't short-circuited by an unwanted soldering bridge
Check that you've put the correct resistance values in the right places.

I send new values for WPM and/or DELAY via an AT command, but they have no effect

If potentiometers are used, and they are not set to zero, the WPM and DELAY potentiometers take precedence!
The values of the AT commands are stored in the flash memory for a next start (or a reset), IF by then the potentiometers are set at zero position.

Appendix 4: How to correctly adjust your FM deviation

For radio amateurs a maximum deviation of 3 kHz is allowed (at least in EU).
You have to make sure that you do not exceed this value, because you do not want to make QRM in a neighbouring channel. You can do this by ear, for example. a friend can observe your signal on the air, and check that you are not audible above and below the working frequency.

However, the correct way to adjust the deviation of an FM transceiver is not so difficult.
Most amateurs already possess a so-called RTL-SDR stick. Together with the program SDR# (or equivalent SDR software), you can judge the output spectrum of your transmitter.
In order not to over-drive the SDR, you have to connect the transceiver to a dummy load, or use the RTL stick without an antenna. Everything depends on the power used and the distance between the devices.

It's also possible to observe the carrier alone with a multimode VHF rig, with a narrow CW filter.

FM-modulation theory tells us that the carrier in an FM-modulated signal has a NULL at certain values of the **modulation-index β**.

This happens at β = **2.40** .. 5.52 .. 8.65 .. etc.

This modulation index is equal to the deviation divided by the modulating frequency:

$$\boldsymbol{\beta} = \frac{\Delta \boldsymbol{f}}{\boldsymbol{f_{LF}}}$$

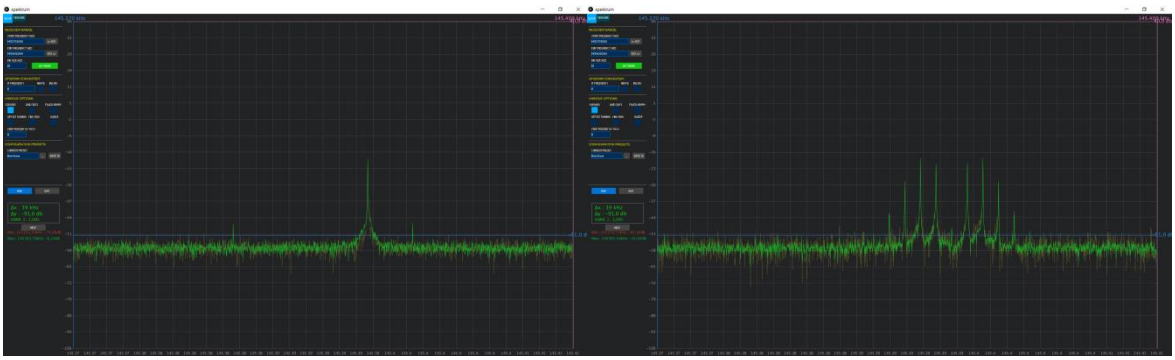
In order to obtain the first null of the carrier, β = 2.40

So we need to modulate with a tone of 3000 Hz / 2.40 = 1250 Hz.

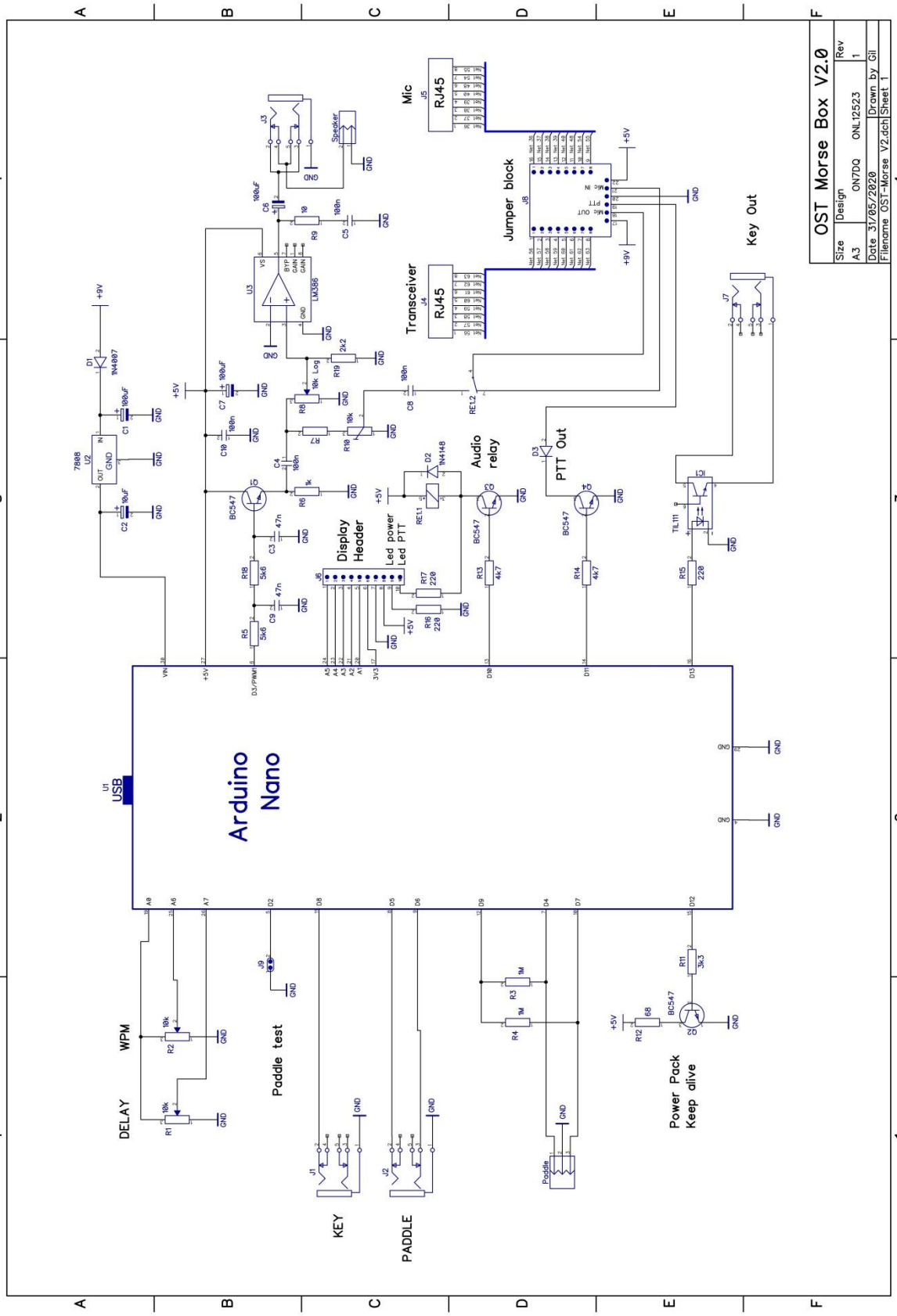
Via the command AT+FREQ=1250 you can set this up.

Transmit the tone, and adjust trimmer R10 from minimum position (no modulation) until you see or hear the carrier disappear for the first time. Then turn the trimmer back a little bit, and you have set the correct deviation.

On the left an image without modulation (note: the small peaks come from the RTL-SDR stick).
On the right an image with modulation, at the first carrier null.
(images recorded with the program "Spektrum").



Circuit diagram



Component placement