Introducing...

nanoKeyer

Professional contest keying for anyone

Kit Building Instructions

by Oscar, DJ0MY

Thanks for choosing to build this keyer.

I hope you will enjoy it and have a great building and operating experience with it.

Oscar, DJ0MY

Arduino Nano based K1EL Winkeyer compatible CW contest keyer

Open source hardware design by DJ0MY

Attention:

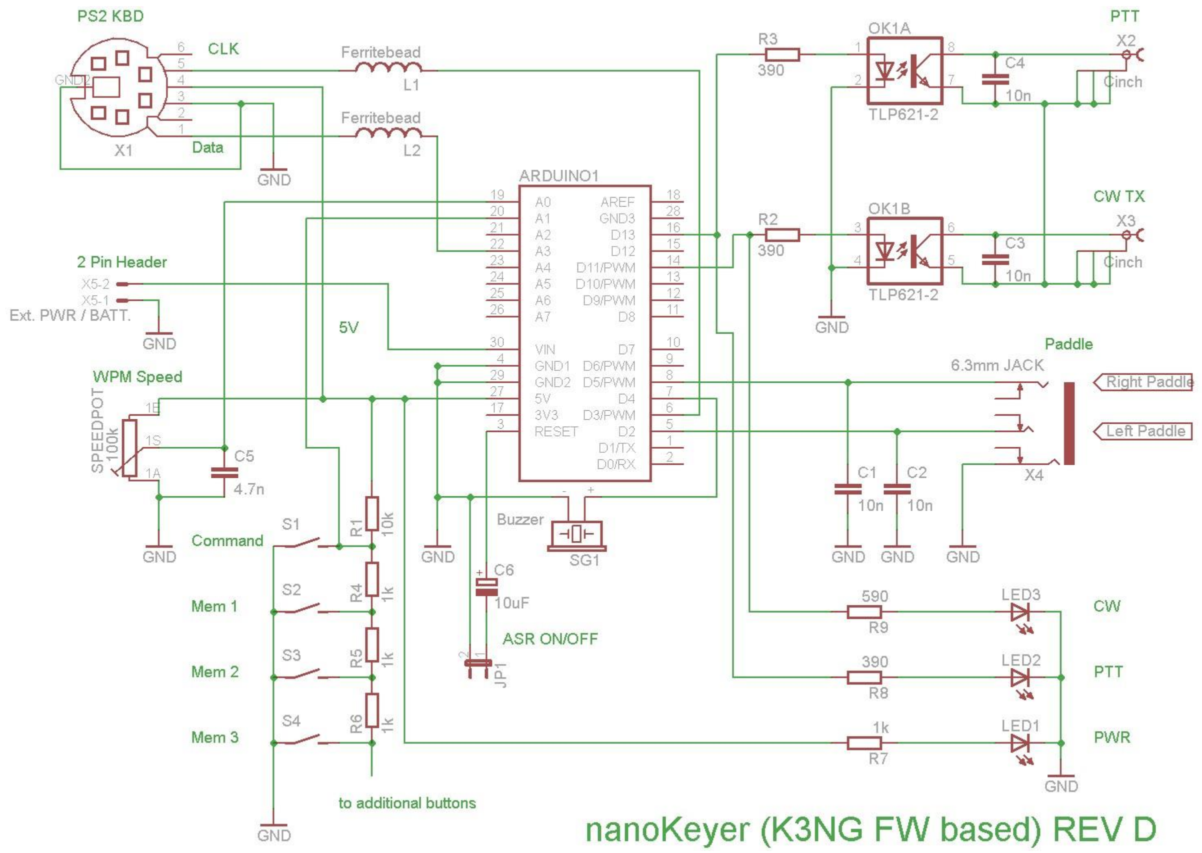
This keyer kit requires in addition an Arduino Nano microcontroller plug-in board. The Arduino Nano must be purchased separately from one of the various Arduino dealers.

Make sure to use only an Arduino Nano V3.0 (or compatible) based on a ATMEGA 328 chip. The Arduino Nano V2.x, which is ATMEGA 168 based, has not sufficient flash memory to hold the size required by the K3NG Arduino keyer source code.

PCB REV D errata:

No entries yet…

nanoKeyer REV D PCB - Schematic Diaagram



Bill of Materials

Before starting to build your kit we strongly suggest to make an inventory and check if all parts are supplied as listed below.

Part list for PCB REV C:

Part	Qty.	Value	Package	Description	Source	Order No.
ARDUINO1	2	20 pin male header	Arduino Nano Socket	2 x 15 pin Headers (break off from 20 pins)	Various	
C1-4	4	10n	2.5mm	CAPACITOR	Various	
C5	1	4.7n	7.5mm	Capacitor WIMA, MKP-10	Various	
C6	1	10uF/100V	2.5mm, radial	Electrolytic capacitor	Various	
L1-2	2	Ferrite	5mm bead	Ferrite bead	Various	
OK1	1	TLP621	DIP8	Optocoupler	Various	
OK1 Socket	1	DIP8 Socket	DIP8	Socket for Optocoupler	Various	
R1	1	10k ohm	Wired resistor	RESISTOR	Various	
R2-3, 8	3	390 ohm	Wired resistor	RESISTOR	Various	
R4-7	4	1k ohm	Wired resistor	RESISTOR	Various	
R9	1	590 ohm	Wired resistor	RESISTOR	Various	
SG1	1	Buzzer	EPM121 A1	Deltron Mini Buzzer	Reichelt	SUMMER EPM 121

R7	**1**	**100k**	**RK09K111**	**ALPS POTENTIOMETER**	**Reichelt**	**RK09K111-LIN100K**
Key1-4	4	Key	SCHURTER	Schurter Switch Model 1082	Reichelt	Taster 1082.7
X1	1		Mini DIN 6pin	Assmann, shielded	Reichelt	EB-DIOS M06V
X2-3	2	TOBU3	TOBU3	Female Cinch Print Connector	Reichelt	CBP
X4	1	6.3mm Stereo Jack		6.3mm Stereo Jack	Reichelt	EBS 63P
X5	1	Ext PWR	2.54mm	2 pin polr'd header	Various	
JP 1	1	Jumper	2.54mm	2 pin Header/Jumper	Various	
LED 1	1	LED Green	5mm dia.	Ultrabright, transpar.	Reichelt	LED 5-08000 GN
LED 2	1	LED Red	5mm dia.	Ultrabright, transpar.	Reichelt	LED 5-4500 RT
LED 3	1	LED Blue	5mm dia.	Ultrabright, transpar.	Reichelt	LED 5-3500 BL

Note:

Since this is an open source hardware project we provide the parts source (mostly from www.reichelt.de in Germany) for all footprint critical parts. Parts from other manufacturers may not fit the PCB holes or may even have a different pin polarity.

Also the LED's are given with its part numbers, because the current limiting resistors are adapted to this specific type of LEDs and each LED color requires a different current setup, since they are operated in low current mode to avoid overloading the Arduino digital pins.

If you use alternative LEDs please make sure to adapt R7-9 to values resulting in e.g. 2-8mA current at the LEDs specified operating voltage. Operating at the typical 20mA may overload the Arduino digital pins, since the LEDs are operated in parallel to the optocouplers…

Trick: How to identify the colour of transparent LED's:

There is an easy way to check for the colour of a colour LED with transparent case.

Just use your ohmmeter in a sufficiently dark room to measure the LED (with the right polarity, so that current from the ohmmeter can run through the LED). Due to the high resistance of the ohmmeter only a very low current will flow and the LED will shine only very faintly, but it should be sufficient to identify the colour of the unknown transparent LED.

Technical Specifications:

These page summarizes some of the key specifications of the nanoKeyer.

Arduino Nano V3.0 Microcontroller:

Microcontroller	Atmel AVR ATmega328
Operating Voltage (logic level)	5 V
Supply Voltage (recommended)	7-12 V
Supply Voltage (limits)	6-20 V
Flash Memory	32 KB of which 2 KB used by bootloader
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz

Power Supply Options:

The Arduino Nano can be powered via the Mini-B USB connection, 6-20V unregulated external power supply (via pin 30), or 5V regulated external power supply (via pin 27). The power source is automatically selected to the highest voltage source.

nanoKeyer:

PTT / CW Keying	RCA/Cinch connector, optical isolated
Paddle	6.3mm Stereo Jack
USB Port	Mini B USB via Arduino Nano
External Keyboard	6pin mini DIN PS2 Jack
Dimensions	100 x 100 x 1.6 mm

CAUTION – MAXIMUM KEYING RATINGS

The absolute maximum collector current rating of the used Toshiba TLP 621 opto-coupler is 50mA. This should be more than sufficient to key most modern transistorized radios.

In case you need to switch older radios (e.g. tube radios or power amplifiers) you may need to change the voltage limiting resistors R2 and R3 and replace the TLP621 by higher rating optocouplers such as TLP627 (150mA) or use solid state relays in pin compatible packages

Preparation Work

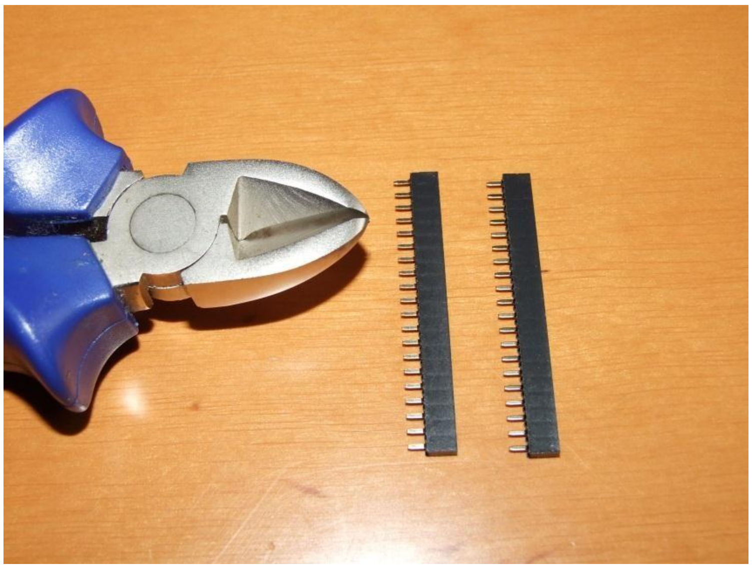
Before you start building the keyer please read the following information regarding the female headers used as a socket for the Arduino Nano plug-in board.

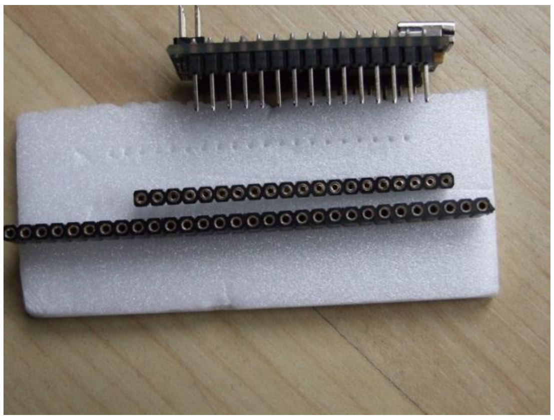
The kit is supplied with 2 x 20 pin female headers.

The Arduino Nano has a total of 30 pins, therefore you will need only 2 x 15 pins. The remaining 5 pins need to be cut-off from the headers prior to building the kit using a small saw or a wire cutting pliers (as on the photo below).

CAUTION:

Please cut in the centre of the 16th pin or you will destroy the headers !
(cutting exactly between the pins will not work due to the fragile plastic)



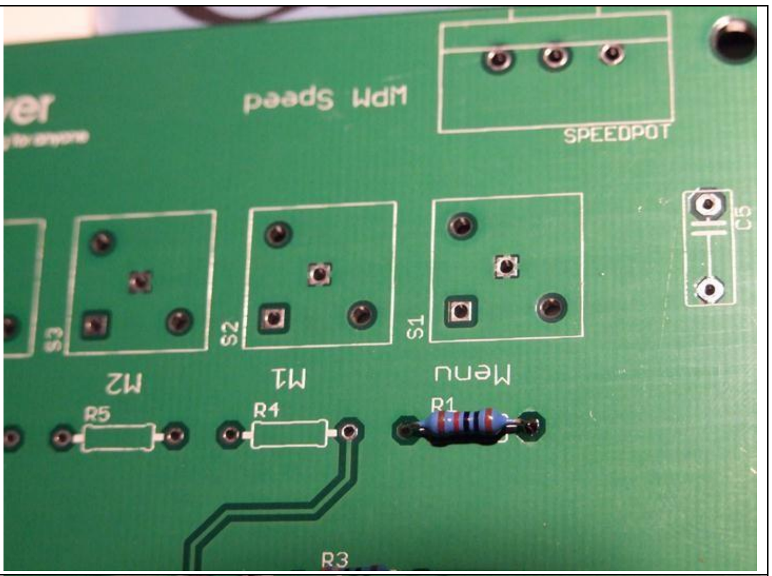
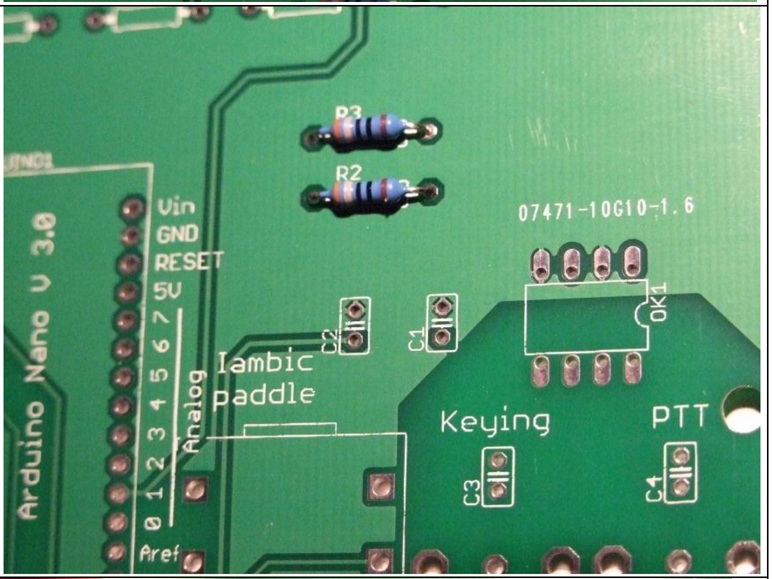
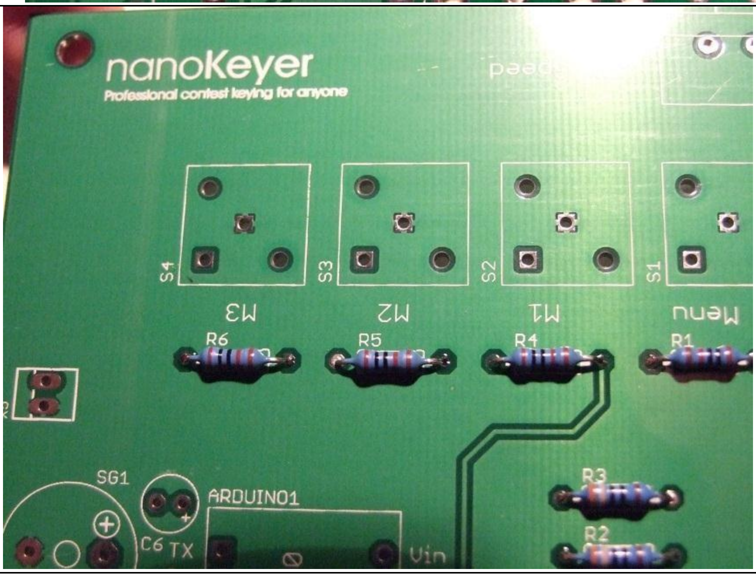


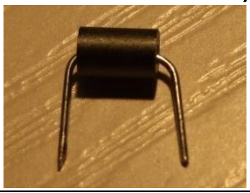
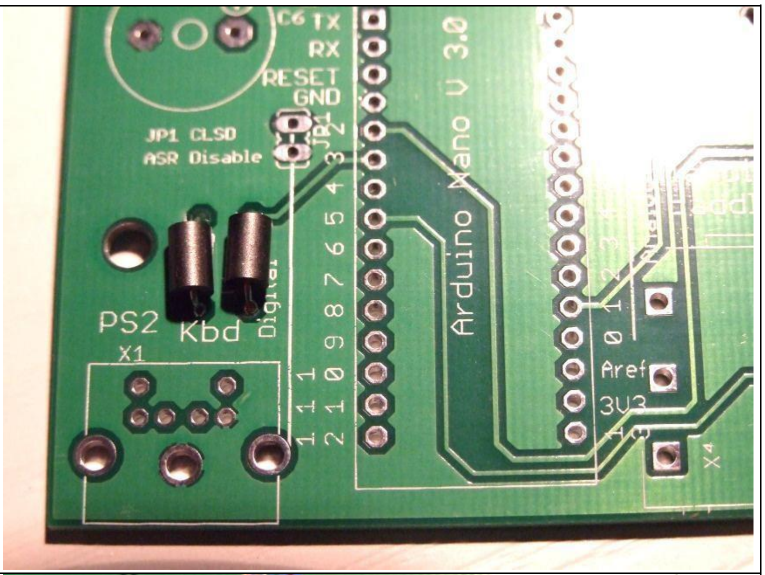
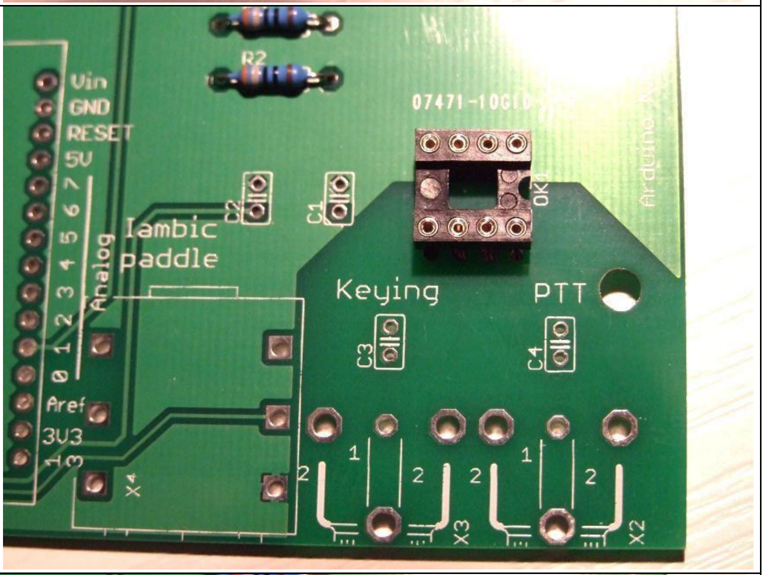
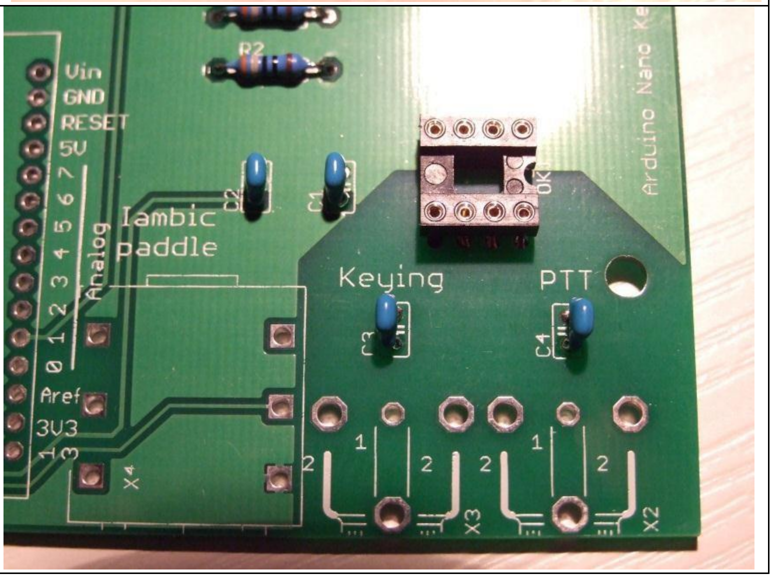
ATTENTION: Use of "Precission Headers"

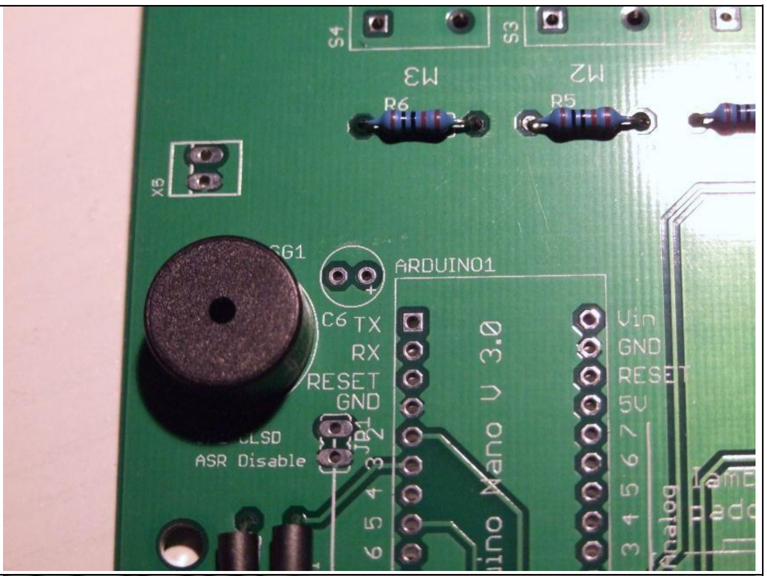
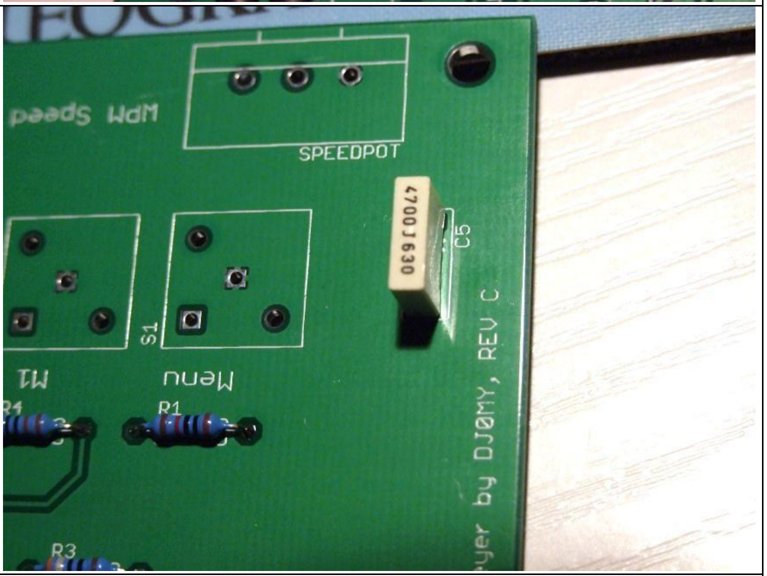
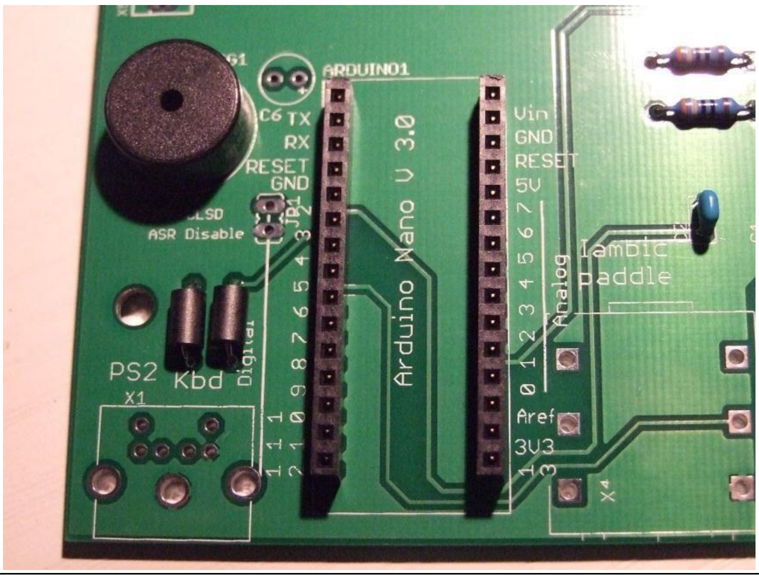
In case you do not want to use the supplied female pin headers and want to use your own precision headers, please make sure you use the large diameter ones (approx.. 0.8mm hole dia.)

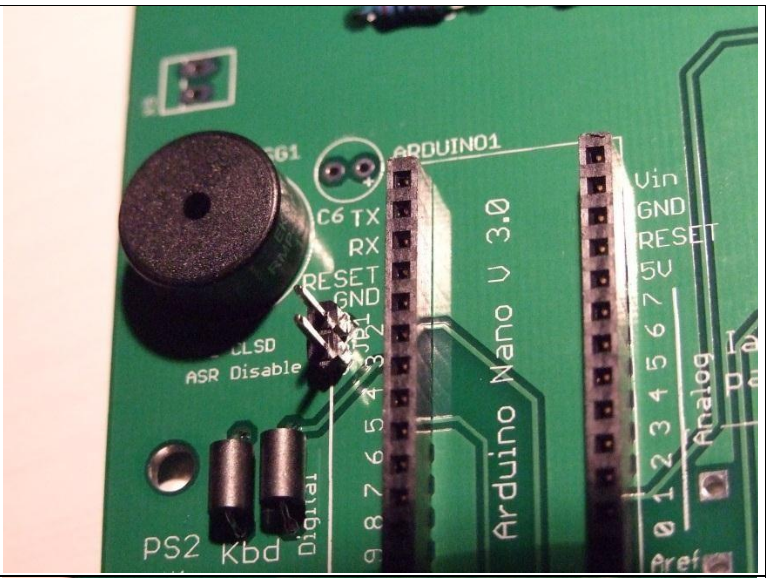
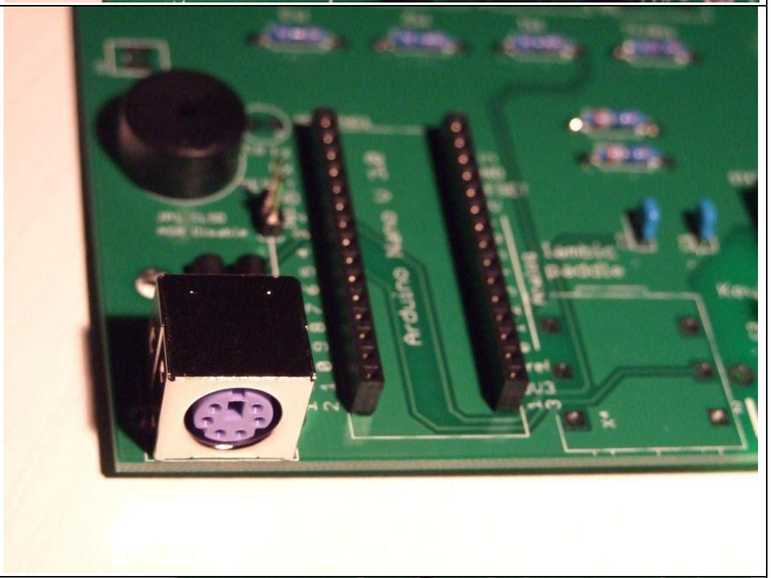
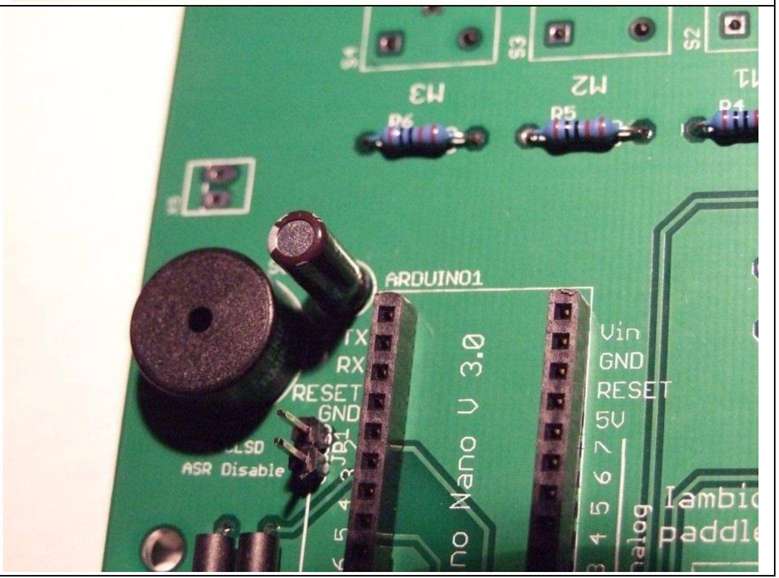
The standard smaller ones (for IC wires) do not fit with the thick square shaped male headers used on most Arduino Nano boards. (See photo: Arduino square male header vs. small diameter vs. large diameter – only the bottom one fits !)

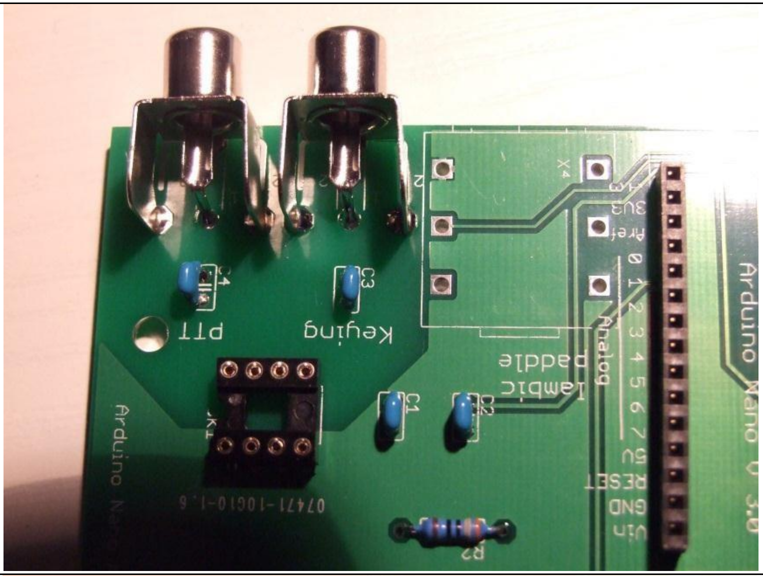
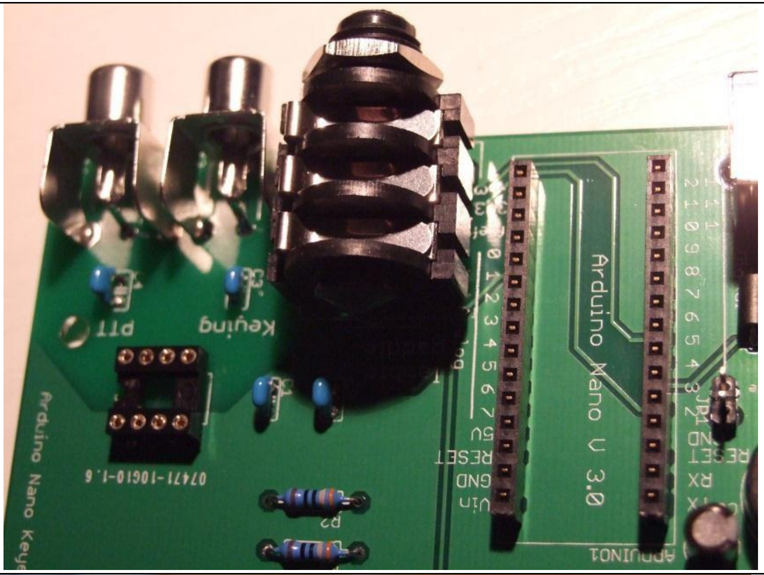
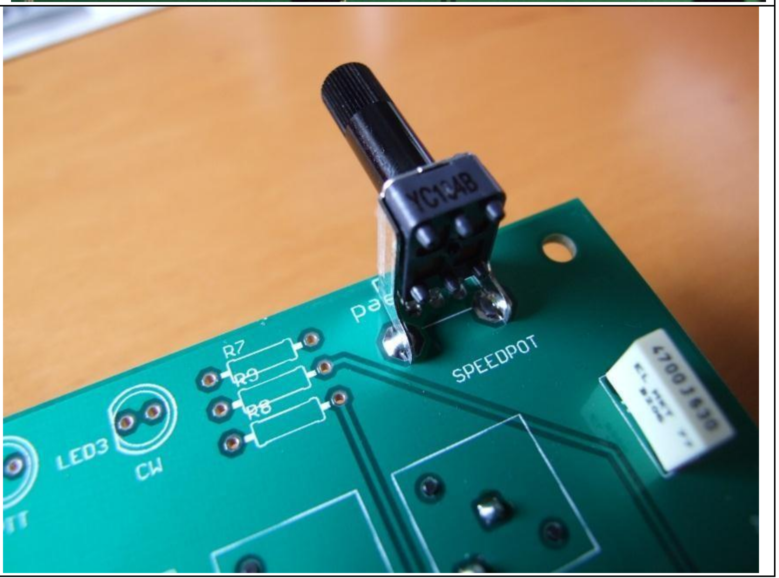
Keyer building steps

Install analog button array resistor R1 (10k ohms)	
Install opto-coupler current limiting resistors R2 and R3 (390 ohms)	
Install the remaining analog button array resistors R4, R5 and R6 (all three 1k ohms)	

Take some cut-off wire leads from the previously soldered resistors, thread them through the ferrite beads, bent into Ushape as shown below and solder into L1 and L2 positions next to the PS2 keyboard jack 	
Solder the 8 pin IC socket for the opto-coupler into the designated position. Make sure the notch of the socket is pointing in the same direction as the notch on the silk screen. You may add the opto-coupler IC now or at the end of building. Make sure the notch of the IC matches the silk screen notch.	
Next install capacitors C 1 – 4 (all 10 nF)	

Install the buzzer matching polarity (see sticker markings) on the silkscreen. Remove the protective sticker after soldering as seen in the picture to the right. Some kits may be delivered with bipolar types (therefore without markings and polarity doesn't matter)	
Next install capacitor C5 (4.7nF)	
Make sure you cutted-off 5 pins from the ends of the 2x20 pin female headers as in the section "Preparation Work"… Solder the two socket rows into position ensuring they sit evenly and unbent on the PCB surface. Please do apply only minimum amount of solder needed to make contact. Otherwise the contacts may be jammed by solder. If this happens it is not a tragedy – the Arduino will just sit a bit higher than usual.	

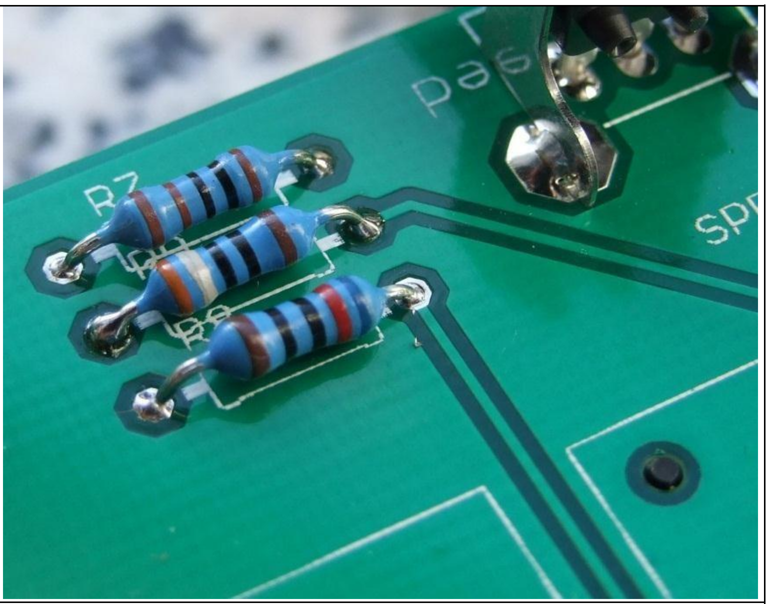
Install the 2pin jumper header(ASR enable / disable jumper) at JP1 Add the JP1 Jumper, but keep the contacts open for the first firmware programming. **NOTE IN REV D: the jumper header has moved near to the LEDs at the front PCB edge**	
Install the PS2 Jack making sure it snapped corrently into position and sitting evenly flat on the PCB surface.	
Install the (ASR disabling) electrolytic capacitor C6 (10uF) making sure you are matching the polarity on the silkscreen.	

Install the two RCA (Cinch) jacks making sure they sit as close as possible and evenly on the PCB surface.	
Install the 6.3mm stereo jack making sure it sits as close as possible and evenly on the PCB surface.	
Install the WPM keying speed potentiometer (100k ohms).	

Install the Green LED current limiting resistor R7 (1k ohms)

Install the Red LED current limiting resistor R8 (390 ohms)

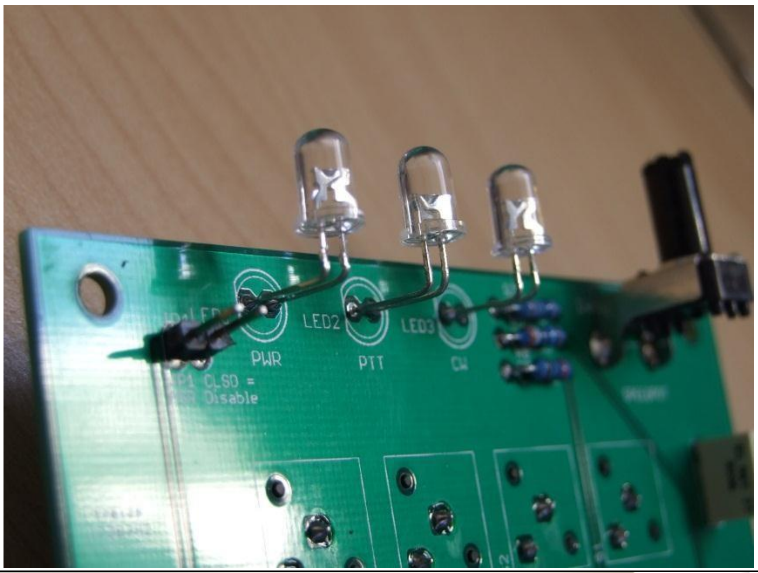
Install the Blue LED current limiting resistor R9 (590 ohms)



Install the Green (PWR), Red (PTT) and Blue (CW) LEDs. Identify the colours with the help of an ohmmeter (look out for a faint light of the LED)

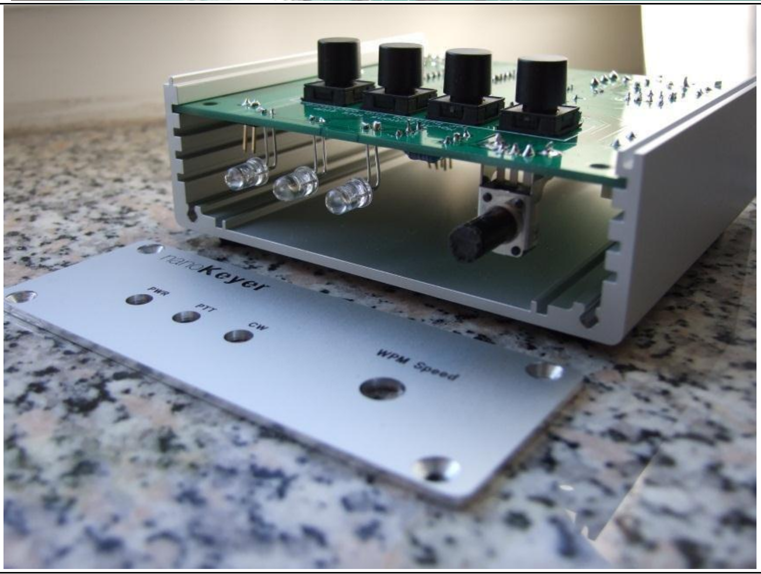
For enclosure mounting bent them with tweezers as shown on the photo. The shorter LED lead must go into the hole at the flat side of the LED silkscreen circle.

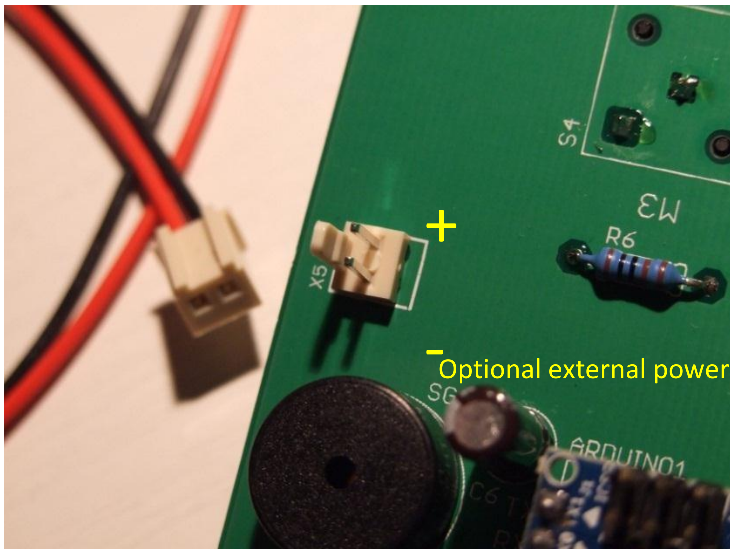
<u>Make sure you don’t mix up the colors</u> as resulting overcurrent may damage the LED.
Make sure polarity is 100% OK.



This is how the LED assembly should look like (incl. the mounting option of buttons mounted on top PCB side as explained on the next page)

It is recommended to chose an enclosure first and have the LED holes drilled so that you can adapt the lead lenghts during enclosure mounting.



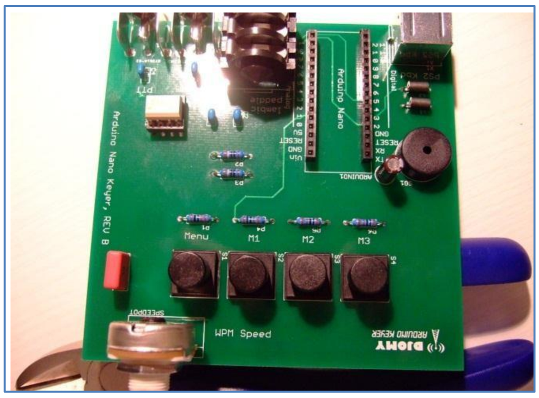
This step is optional. **The device can be powered solely via the computer USB port. External power is only needed for computerless operation.** Install the 2 pin polarized male external power header. Make sure the tab of the header points to the direction of the nearest PCB edge (as in the photo). Solder it with the cable + female header attached to prevent the pins from bending due to the solder heat	 + - Optional external power X5 R6 M3 S4 ARDUINO1 Instead of an external power jack you may use a USB cable connected into a modern USB mobile phone charger or USB cigarette lighter adapter for mobile use.

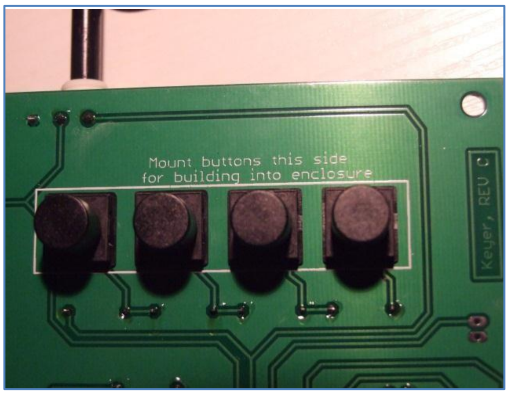
Button Mounting Options:

There are two options for mounting the command and memory buttons.

1. If you do not plan to build the PCB into an enclosure and use it as an open PCB module you will operate the nanoKeyer from the top side and therefore you will need to mount the buttons on the component side
2. If you plan to build the PCB into an enclosure you would mount the PCB upside down and therefore need to mount the buttons on the solder side. See pictures on the next page. When mounting this way make sure the K3NG firware is compiled with this option “uncommented” to reverse the button order response of the Firmware:

   #define OPTION_REVERSE_BUTTON_ORDER   (see Arduino Firmware source code)





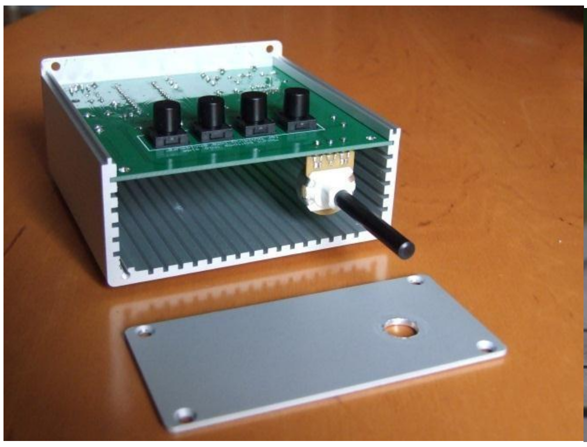
Buttons mounted on top side (for “open use”)   Buttons mounted on solder side (for enclosure use)

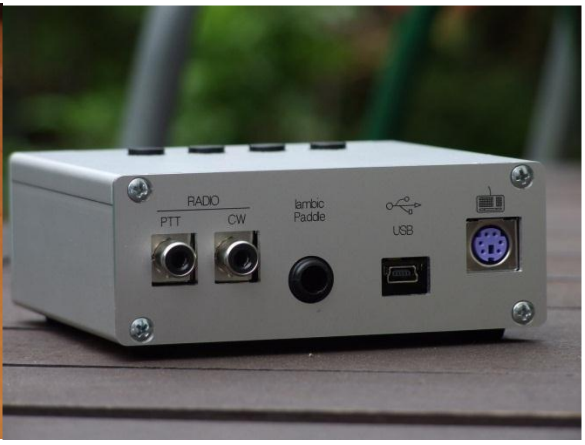
Enclosure mounting

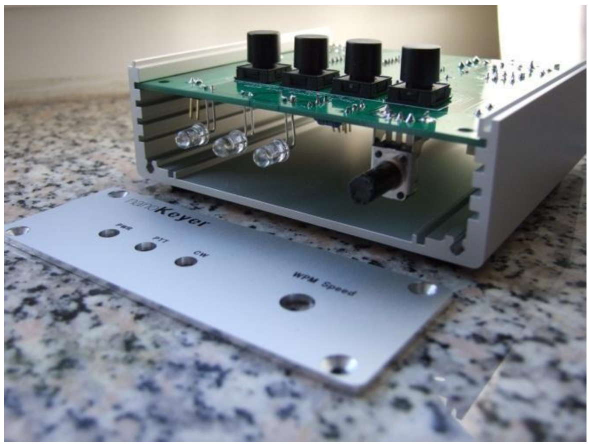
The nanoKeyer PCB can be built into a standard enclosure of your choice by means of the four mounting holes and using standoffs and screws. It is advisable to do the up-side-down mounting option in this case.

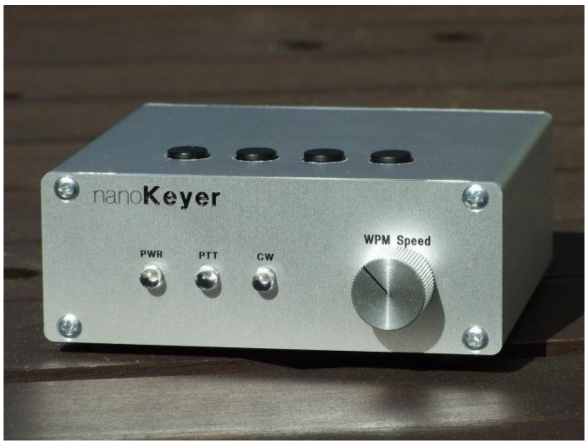
However, we strongly recommend building it into an 10x10cm (approx. 4-5cm height) extruded aluminum enclosure of your choice where you would just slide the PCB into some support notches to make the mounting easy and look good.

See photos with mounting examples below:





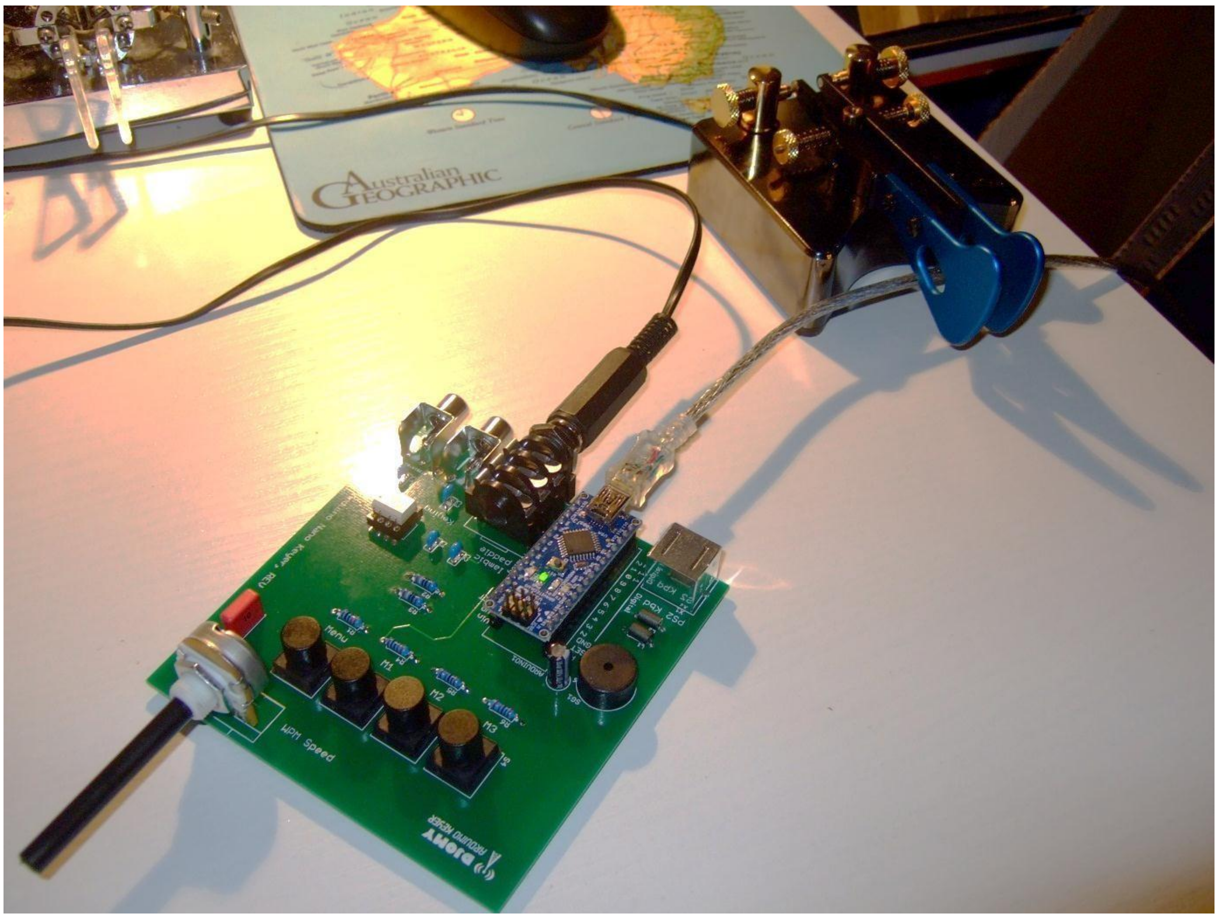




How to Avoid Ground Loops

In case you use a conductive enclosure (as on the photos above) make sure you isolate the enclosure inside around the RCA keying jacks to avoid losing the optical isolation feature of the nanoKeyer. The same is true when using external power or power via USB cable. Make sure that the same power source is not connected to your radio (e.g car battery, computer sound card used as modem, etc.) to avoid ground loops risking RFI pickup.

The completed keyer board (REV. B shown here) should look like this:



Now it is time to upload the firmware into the Arduino Nano and do the first test ride !!!

Got to the downloads section of my website

http://nanokeyer.wordpress.com

Please download

nanoKeyer – K3NG Firmware Upload Guide

A step by step guide into uploading the firmware

ANNEX I

For making maximum use of the nanoKeyer hardware features we suggest to compile the following features of the K3NG sourcecode:

To make the PS2 keyboard library to compile without errors I suggest to copy the library files (PS2Keyboard.cpp and PS2Keyboard.h) into a new subfolder (named PS2Keyboard) of the library folder of your Arduino IDE installation location on the HDD. (see K3NG website)

Recommended to compile (have "uncommented" in source code) as follows:

```
#include <PS2Keyboard.h>

#define FEATURE_SERIAL
#define FEATURE_WINKEY_EMULATION
#define FEATURE_SAY_HI
#define FEATURE_MEMORIES
#define FEATURE_POTENTIOMETER
#define FEATURE_PS2_KEYBOARD
#define FEATURE_DEAD_OP_WATCHDOG
#define FEATURE_AUTOSPACE
#define FEATURE_FARNSWORTH
#define OPTION_INCLUDE_PTT_TAIL_FOR_MANUAL_SENDING
#define OPTION_SERIAL_PORT_DEFAULT_WINKEY_EMULATION

Only if buttons mounted up-side-down (for enclosure mounting:
#define OPTION_REVERSE_BUTTON_ORDER

PS2Keyboard keyboard
```

All other default features should be disabled ("commented" out with // at the beginning of each line). Otherwise the compiled binary file becomes too large and does not fit into the Arduino Nano flash memory of approx.. 30kB anymore.

ANNEX II

K3NG Firmware Source Code Notes and Instructions.

Source: http://radioartisan.wordpress.com/arduino-cw-keyer/

Connecting the Keyer (ignore this section for nanoKeyer)

Here are the main pins you need to connect up to get started:

- Left Paddle – pin 2 – connect to your left paddle (grounding will send dits)
- Right Paddle – pin 5 – connect to your right paddle (grounding will send dahs)
- Transmitter Key – pin 11 – goes high for key down; use to drive a transistor to ground the TX key
- Sidetone – pin 4 – this outputs square wave sidetone to drive a speaker (schematic coming out shortly for driving with a transistor). The sidetone can be deactivated on transmit for transmitters that generate their own sidetone.
- The command button – pin A1 and at least R7
- Memory buttons, up to 12. Add buttons and resistors R8, R9, R10, etc. (You can do just a few memory buttons, all 12, or none at all.

Additional pins you may be interested in for other functionality:

- PTT (push to talk) Transmitter 1 – pin 13 (described in more detail below)
- Additional PTT pins for multi-transmitter capability
- Potentiometer – pin A0 – connect one end of the pot to +5V, the other end to ground, and connect the wiper to pin A0

All pins can be easily changed at the beginning of the code if desired, though note that if the PS2 keyboard functionality is used, the clock pin must remain at pin 3 due to interrupt requirements. Also, the future optional I2C functionality which is in development must use pins A4 and A5.

Buttons

Button 0 is the command button. Pressing it will put the keyer into command mode which is described in detail below. Holding down the command button and pressing the left or right paddles will increase or decrease the CW speed.

Buttons 1 through 12 will play memories when momentarily depressed. To have a memory autorepeat (such as for doing a repetitive CQ), hold down the memory button and tap the left paddle. Holding buttons 1 through 6 down for a half second will switch the transmitter (1 through 6), if multiple PTT lines are enabled.

Command Mode

To enter command mode, press button 0, the command button and you will hear a “boop beep”, after which you can enter various commands by sending character using the paddle. (Note that if you’re in bug or straight key mode, you will temporarily be switched to iambic in command mode.)

If you enter a bogus command or the keyer didn’t recognize the character you sent, it will send a question mark, upon which you can retry your command.

To exit command mode, send X in CW using the paddles or just press the command button again upon which you will hear “beep boop” and you’ll be back in regular sending mode.

A – Switch to Iambic A mode

B – Switch to Iambic B mode

D – Switch to Ultimatic mode

F – Adjust sidetone frequency

G – Switch to bug mode

I – TX enable / disable

J – Dah to dit ratio adjust

N – Toggle paddle reverse

O – Toggle sidetone on / off

P# – Program a memory

T – Tune mode

V – Toggle potentiometer active / inactive

W – Change speed

X – Exit command mode (you can also press the command button (button0) to exit)

Z – Autospace On/Off

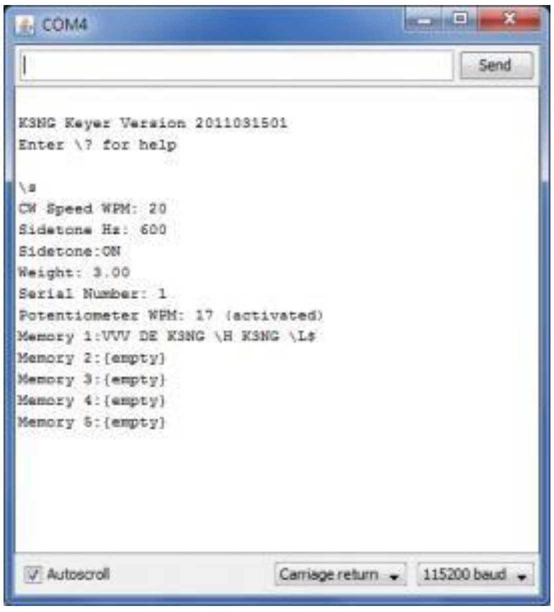
# – Play a memory without transmitting

Serial Command Line Interface ("CLI") / CW Keyboard

The keyer has a serial command line interface using the built in Arduino USB port. Simply connect to your computer and use a terminal program such as the Arduino serial port program or Putty. If you use the Arduino program, it's recommended that you set it for carriage return (lower right).

To use the CW keyer functionality, simply type in what you want to send. In the Arduino serial interface you will need to hit Enter to send the data to the keyer for it to start sending. Programs like Putty will immediately send the characters and the keyer will send the code immediately as well.

Commands are preceded with a backslash (" \ "), the key above your Enter key (at least on US PC keyboards). To see a help screen, enter backslash question mark " \? " (no quotes). The status command (\s) is a useful command for viewing various settings and seeing the contents of the memories. If you enter a double backslash ("\\"), all sending buffers will be cleared and any memory sending will stop (this includes sending invoked by the PS2 keyboard or Winkey interface emulation features).



```
K3NG Keyer Version 2011031501
Enter \? for help

\s
CW Speed WPM: 20
Sidetone Hz: 600
Sidetone:ON
Weight: 3.00
Serial Number: 1
Potentiometer WPM: 17 (activated)
Memory 1:VVV DE K3NG \H K3NG \L$
Memory 2:{empty}
Memory 3:{empty}
Memory 4:{empty}
Memory 5:{empty}
```

Carriage return
115200 baud

Command Line Interface Showing Status (\s) Command Output

CLI Commands:

\? Help

\# Play memory #

\a Iambic A mode

\b Iambic B mode

\c Switch to CW (from Hell)

\d Ultimatic mode

\e#### Set serial number to ####

\f#### Set sidetone frequency to #### hertz

\g Bug mode

\h Switch to Hell sending

\i Transmit enable/disable

\j### Dah to dit ratio (300 = 3.00)

\k Callsign receive practice

\l## Set weighting (50 = normal)

\m### Set Farnsworth speed

\n Toggle paddle reverse

\o Toggle sidetone on/off

\p# Program memory #

\q## Switch to QRSS mode, dit length ## seconds

\r Switch to regular speed mode

\s Status

\t Tune mode

\u Manual PTT toggle

\v Toggle potentiometer active / inactive

\w### Set speed in WPM

\x# Switch to transmitter #

\y# Change wordspace to # elements (# = 1 to 9)

\z Autospace on/off \+ Create prosign

\!## Repeat play memory

\|#### Set memory repeat (milliseconds)

* Toggle paddle echo

\^ Toggle wait for carriage return to send CW / send CW immediately \~

Reset unit

To enable the CLI, you must uncomment two lines in the source code before compilation:

#define FEATURE_SERIAL

#define FEATURE_COMMAND_LINE_INTERFACE

CW Speed Adjustment

The CW sending speed can be adjusted several ways:

- The W command in command mode
- The command line interface \w command
- The memory macro \w, or \y and \z for incremental increases or decreases
- Holding down the command button (button 0) and pressing the left and right paddles

The speed potentiometer can also adjust the speed. The pot must first be activated using the command mode V command or command line \v command. Adjusting the speed pot will immediately change the CW speed during manual sending or memory playing, however its changes will not be written to non-volatile memory. If the speed is changed using other methods (command mode, command line interface, memory macro, command button shortcut) that will override the pot setting until the pot is adjusted again.

Beacon Mode

In order to have the keyer go directly into beacon mode at power up or reset and stay in beacon mode, simply ground pin 2. This is useful for keyers that are dedicated to beacon or fox service.

Iambic Modes

To switch to Iambic A mode, use the A command in command mode or \a in the command line interface.

To switch to Iambic B mode, use the B command in command mode or \b in the command line interface.

(An explanation of Iambic A and B can be found here.)

Straight Key Mode

To go into straight key mode, hold down the right paddle when powering up or power resetting.

Bug Mode

To go into bug mode, use the command mode G command or the command line \g command.

Ultimate Mode

To go into Ultimatic mode, use the command mode D command or the command line \d command.

Sidetone Line

The sidetone line normally outputs square wave sidetone for driving a speaker. Sidetone can be disabled on transmit using the command mode O command. This is for transmitters that generate their own sidetone.

The sidetone frequency can be adjusted using the F command in command mode.

PTT ("Push To Talk")

The PTT pins go high whenever code is sent. If it's desired to have the PTT line go high before code is sent or stay high for a period of time after code stops being sent, these two lines can be adjusted:

```
#define initial_ptt_lead_time 0 // PTT lead time in mS
```

```
#define initial_ptt_tail_time 200 // PTT tail time in mS
```

The lead and tail times are in milliseconds. This feature is useful for driving T/R switches or older transmitters than need a little more time to get keyed up, or FM fox transmitters that need to have PTT keyed and sidetone pumped into the microphone line.

Hang time can be set by modifying this line:

```
#define default_ptt_hang_time_wordspace_units 0.0
```

PTT tail time is invoked when sending code automatically, such as via a memory play, the CLI, the PS2 keyboard, or Winkey interface emulation. PTT hang time is invoked for manual sending using the paddle and is speed (wpm) dependent.

Note that if you activate PTT lead time, you should activate tail time as well, otherwise PTT lead time will be invoked before each dit or dah, significantly slowing down the sending speed.

Currently PTT lead, tail, and hang times can only be changed at runtime using the Winkey interface emulation. (Let me know if you would like CLI commands to do this.)

For testing purposes the PTT line can be manually toggled on and off using the \u CLI command.

If your CW transmitter keys up when the CW line is keyed (or you are not going to use multitransmitter support), there is probably no need to use the PTT line.

If you do not need the PTT lines and wish to use the Arduino pins for another function such as a transmitter keying line, simply set the pin number to zero, as so:

```
#define ptt_tx_1 0 #define

ptt_tx_2 0
```

QRSS (Slow Speed CW)

QRSS mode can be activated using the command line \q command or in memory macros using the \q macro. Both take the dit length in seconds (double digit number) as an argument. For example: \q09 would put the keyer in QRSS mode with nine second long dits (and 27 second long dahs).

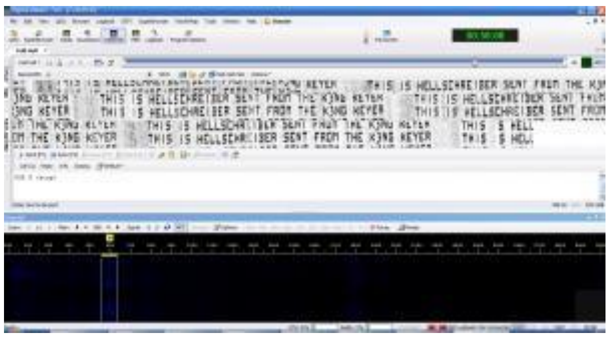
The \r command will switch back to regular CW speed mode in both the command line and in memories.

HSCW (High Speed CW)

High speed CW can be accomplished by using the \w command line interface command or in memories as a macro. Whereas the command mode speed adjustment and the speed potentiometer allow the speed to go up to a maximum of 60 WPM, the \w command will let you take it up to 255 WPM.

Hellschreiber

The keyer will send Hellschreiber characters by placing it in Hellscreiber mode using the \h command in the serial command line interface or memory macros. In the command line interface \c will return the keyer to CW mode and the \l (as in lima) memory macro will change back to CW. While in Hellschreiber mode, the paddle will still send CW. The Hellschreiber mode is intended mainly for beacons but works just find for direct keyboard sending.



```
KEYER      THIS IS HELLSCHREIBER SENT FROM THE K3
3NG KEYER      THIS IS HELLSCHREIBER SENT FROM THE K3NG KEYER      THIS IS HELLSCHREIBER SENT FR
3NG KEYER      THIS IS HELLSCHREIBER SENT FROM THE K3NG KEYER      THIS IS HELLSCHREIBER SENT FROM
OM THE K3NG KEYER      THIS IS HELLSCHREIBER SENT FROM THE K3NG KEYER      THIS IS HELL
OM THE K3NG KEYER      THIS IS HELLSCHREIBER SENT FROM THE K3NG KEYER      THIS IS HELL
```

Hellschreiber Copied From the Keyer Speaker into a Laptop (click to enlarge)

Memory Operation and Memory Macros

Memories can be manually played using buttons 1 through 5, or using the \# command in the command line interface (for example, \1 plays memory 1). In command mode the memories can be sent without transmitting by entering the number of the memory.

Memories are programmed using the command line interface \p# command or in command mode using the P command.

To program memory 1 with CQ CQ CQ DE K3NG, the command would be \p1CQ CQ CQ DE K3NG.

To program memory 1 using command mode, enter command mode by pressing the command button and sending the P command. After hearing a beep, send the CW code to be stored and when finished, hit the command button to exit programming. The keyer will then play back the memory. If the keyer didn't recognize a character you sent it will send a question mark in its place.

Macros can be placed in memories to do cool things. Some macros include:

\# Jump to memory # (1 through 9)

\c Play serial number with cut numbers

\d### Delay for ### seconds

\e Play serial number, then increment

\f#### Set sidetone to #### hertz

\h Switch to Hell sending

\l Switch to CW (from Hell mode)

\n Decrement serial number, do not send

\q## Switch to QRSS mode, dit length ## seconds

\r Switch to regular speed mode

\t### Transmit for ### seconds

\u Activate PTT

\v Deactivate PTT

\w### Set regular mode speed to ### WPM

\x# Switch to transmitter # (1, 2, 3, 4, 5, or 6)

\y# Increase speed # WPM

\z# Decrease speed # WPM

\+ Prosign the next two characters

(Note that both command line commands and CW memories are case insensitive.)

The number of memories is set at compile time using these lines:

```
#define number_of_memories 12

#define memory_area_start 20

#define memory_area_end 1023
```

Up to 12 memories can be configured, with some caveats. Nine memories are supported in the CLI and in memory macros, and the full 12 are supported with the PS2 keyboard.

Memory_area_start and memory_area_end define the starting and ending EEPROM locations for the entire bank of memory. The memory area is divided up evenly between the memories. The example settings above will result in 12 memories each with 83 bytes, or 83 characters.

CW Dah to Dit Ratio Adjust

The CW dash length to dot length ratio can be adjusted using the J command in command mode. Upon entering the J command you will hear a repeating dit dah. Use the left and right paddles to shorten or lengthen the dah. Squeeze both paddles to exit the weight adjust command. After that you can enter X or press the command button to exit command mode.

The ratio can also be adjusted in the command line interface using the \j command. \j300 sets the keyer for a normal 3:1 ratio, \j250 would set it for a 2.5:1 ratio, for example.

Paddle Reverse

The command mode N command switches the left and right paddles. The equivalent function in the CLI is \n and using the PS2 keyboard it’s CTRL-N.

Tune Mode

The command mode T command or command line interface \t command goes into tune up mode. In the PS2 keyboard, use CTRL-T.

TX Disable / Enable

The transmit line can be disabled and enabled using the \i CLI command or I command in command mode. The equivalent PS2 keyboard command is CTRL-I. This feature can be used for sending practice without keying the transmitter.

Autospace

The autospace feature can be toggled on and off with the Z command in command mode, the \z command in the command line interface, and CTRL-Z using the PS2 keyboard. This feature "cleans up" manual sending a bit by automatically inserting a wordspace delay if the operator waits more than one dit after sending a dit or dah to paddle either paddle. The autospace feature is activated by uncommenting this line:

```
#define FEATURE_AUTOSPACE
```

Wordspace Adjustment

Wordspace is the key up time in between words. By default it is set for seven dit lengths by this line:

```
#define default_length_wordspace 7
```

This can be adjusted using the \y command line interface command.

Keying Compensation

The keying compensation filter extends the time of both dits and dahs to compensate for older transmitters that are slow on the draw in QSK at higher speeds. The inter-element key up times are reduced a corresponding amount of time. The time in mS can be set here:

```
#define default_keying_compensation 0
```

Currently there is no command to adjust this at runtime, however the Winkey emulation will adjust this if it is set in the host application.

First Element Extension Time

This feature makes the first dit or dah sent longer to compensate for slow T/R switches in rigs. The time is set here:

```
#define default_first_extension_time 0
```

Currently there is no command to adjust this at runtime, however the Winkey emulation will adjust this if it is set in the host application.

Prosigns

Custom prosigns can be sent using \+ in the CLI or in memories as a macro. Several “hard wired” / common prosigns are available for various keys on the PS2 keyboard like =, -, &, etc. and the Sroll Lock key can be used to create custom prosigns on the fly.

Receive Callsign Practice

In the command line interface the \k goes into callsign receive practice. Random callsigns are sent, the user enters the received callsigns, and the keyer will tell the user if they were correct.

Currently this code produces only US callsigns. I’ll be working on enhancements later to add other country callsigns, allow various user settings and adjustments, and variable speed based on the user’s accuracy.

This feature requires this to be uncommented:

```
#define FEATURE_CALLSIGN_RECEIVE_PRACTICE
```

PS2 Keyboard Interface

A common PC keyboard (PS2) can be interfaced with the keyer to create a computerless CW keyboard. Here’s what you need to do:

1. Download the modified PS2Keyboard library files below. Create a directory in your sketchbook directory called *PS2Keyboard* and place the two files in there.

2. Uncomment the following lines in the K3NG Arduino Keyer code:

```
#include <PS2Keyboard.h>

#define FEATURE_PS2_KEYBOARD PS2

Keyboard keyboard;
```

3. Connect up a PS2 keyboard to your Arduino. Details on the pinouts of a PS2 keyboard connector can be found here.

Special Key Assignments:

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-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…

ALT-F1, F2, F3… – Repeat memory 1, 2, 3…

CTRL-F1, F2, F3… – Switch to transmitter 1, 2, 3…

Myself and others have experienced issues using just the USB +5V to power the Arduino and keyboard, with operation being erratic or the keyboard just not functioning at all. This is due to computer USB ports not being able to supply enough current. The solution is simple: power the Arduino board directly using the power connector.

Note that the keyboard data line can be relocated to other pins if desired, but the keyboard clock line must remain at pin 3 as that pin has special functionality for interrupt operation which is required by the PS2 keyboard library code.

Interfacing to Logging and Contest Programs / Winkey 1.0 Interface Protocol Emulation

The keyer can be interfaced to logging and contest programs with the Winkey emulation feature. To enable, uncomment the following line:

```
#define FEATURE_WINKEY_EMULATION
```

If you want compile both the CLI and Winkey emulation features and upload to a unit, uncommenting the line below will cause the unit to default to Winkey emulation rather than the normal Command Line Interface mode at power up or reset.

```
#define SERIAL_PORT_DEFAULT_WINKEY_EMULATION
```

With the Winkey emulation feature enabled, if you hold down the command button (button 0) and reset or power up the unit, it will go into the non-default mode. (If the default is Winkey Emulation, it will go into Command Line Interface mode, and vice versa.)

You may need to disable some features to get both the CLI and Winkey features to fit into an Arduino Uno. Other larger Arduino variants may be able to host all features. (Your mileage may vary.)

In Winkey Emulation mode the USB port will be set for 1200 baud. The emulation is a 99.9% complete emulation, and it should work with most programs that support Winkey interfacing. The N1MM contesting program and Ham Radio Deluxe (HRD) have been tested and work with all features I've tried.

Currently the following functions are implemented:

- CW Sending (of course)
- Pause
- Key Down
- Unbuffered and Buffer Speed Setting
- Iambic A & B / Bug Mode Settings
- Ultimatic in normal, dit priority, and dah priority modes
- Farnsworth
- Pointer Operations
- Backspace
- Sidetone Frequency Setting
- Paddle Reverse
- Paddle Watchdog
- Keying Compensation
- Dit to Dah Ratio
- Contest Wordspace
- Autospace
- PTT Lead, Tail, and Hang Time
- Speed Pot Setup and Query
- First Extension
- Software Paddle
- Weighting
- HSCW
- Serial Echoback
- Prosigns

This functionality is intended to interface to logging and contest programs and is not intended to be a Winkey replacement. The Winkey "protocol" is a de facto standard and many programs support it, and developing an open standard protocol and getting all the major programs to support it would be a monumental undertaking. So it made sense to merely emulate the existing protocol everyone else is talking.

I have found to have this emulation work reliably with programs other than N1MM, you should disable the Arduino *Automatic Software Reset* described here. This is done by cutting the PC board trace labeled RESET-EN on the Arduino Uno board. I have found with some programs, including HRD and the Winkeyer WKDemo program, when the program connects to the COM port, errant bytes are interpreted/received by the Arduino which trips up the protocol conversation and the program and keyer will not connect. In this configuration the keyer will not reset when a program connects to the COM port and it will be "ready to talk" immediately when the program begins sending bytes.

If you do not disable Automatic Software Reset and are using Ham Radio Deluxe uncomment the following line:

```
#define OPTION_WINKEY_DISCARD_BYTES_AT_STARTUP
```

This option will discard the first three bytes that arrive on the USB port. This hack works for my hardware, but your mileage may vary. The number of bytes discards at start up can be set here:

```
#define winkey_discard_bytes_startup 3
```

A side effect of disabling Automatic Software Reset is that you will need to manually hit the reset button when uploading new software to the Arduino. The button should be pressed as soon as you “Binary sketch size: xxxxx bytes” message in the Arduino program.

In the current build of HRD I’m using, there is a bug in its Winkey interface implementation. If you rapidly change the dah to dit ratio in the graphical user interface or change the speed rapidly, HRD will send incomplete commands to the Winkey. This will cause errant characters to be sent by the keyer, but otherwise the keyer will continue to function.

Ham Radio Deluxe offers a very nice Winkey settings interface. Presumably one could use this interface in place of the keyer command mode or command line interface and control most of the functionality in this keyer.

If you attempt to use this emulation with other programs and have issues, please let me know and I’ll attempt to figure it out. Serial port sniffer captures are helpful in troubleshooting these issues.

Dead Operator Watchdog

This feature turns off the transmit line after 100 consecutive dits or dahs. It can be enabled by uncommenting this line:

```
#define FEATURE_DEAD_OP_WATCHDOG
```

EEPROM / NonVolatile Settings

Most settings are stored in non-volatile EEPROM memory. Memory macros which alter the CW speed are not stored to EEPROM as to avoid “wearing out” EEPROM locations, especially in beacon mode.

Reset to Defaults

In order to reset the keyer to defaults, depress both the left and right paddles and do a reset or power reset. This will wipe out all memories and change all the settings back to defaults.

Multi-Transmitter Capability

This keyer supports multiple transmitters that can be selected using the \x CLI command, the CTRL-F1, F2, etc. key combinations on the PS2 keyboard, or using the hardware buttons (button0 hold + button1, button2, etc.). The code by default has two PTT pins defined, pins 13 and 12 which are transmitters 1 and 2, respectively. Up to six transmitter PTT lines can be defined by configuring this code:

```
#define ptt_tx_1 13

#define ptt_tx_2 12

#define ptt_tx_3 0

#define ptt_tx_4 0

#define ptt_tx_5 0

#define ptt_tx_6 0
```

Setting a line to zero disables it. Obviously, with the Arduino Uno, pins are at a premium and each features uses pins. Larger Arduino platforms like the Mega offer more pins and more compiled-in functionality due to the larger memory space.

Code Compilation

All of the features will not fit on an Arduino Uno simultaneously. I find that when the compiled code goes over about 29K, the upload to the Uno will fail.

I'm in the process of getting an Arduino Mega. The code here should compile just fine for that platform, and with 128K of flash memory it should easily hold all of the program functions.

Miscellaneous Notes

To use "legacy style" buttons (one button per I/O pin), uncomment this line:

```
#define FEATURE_LEGACY_BUTTONS
```

Do not enable the potentiometer feature if you do not have a potentiometer connected, otherwise noise on the pin will falsely trigger wpm changes.