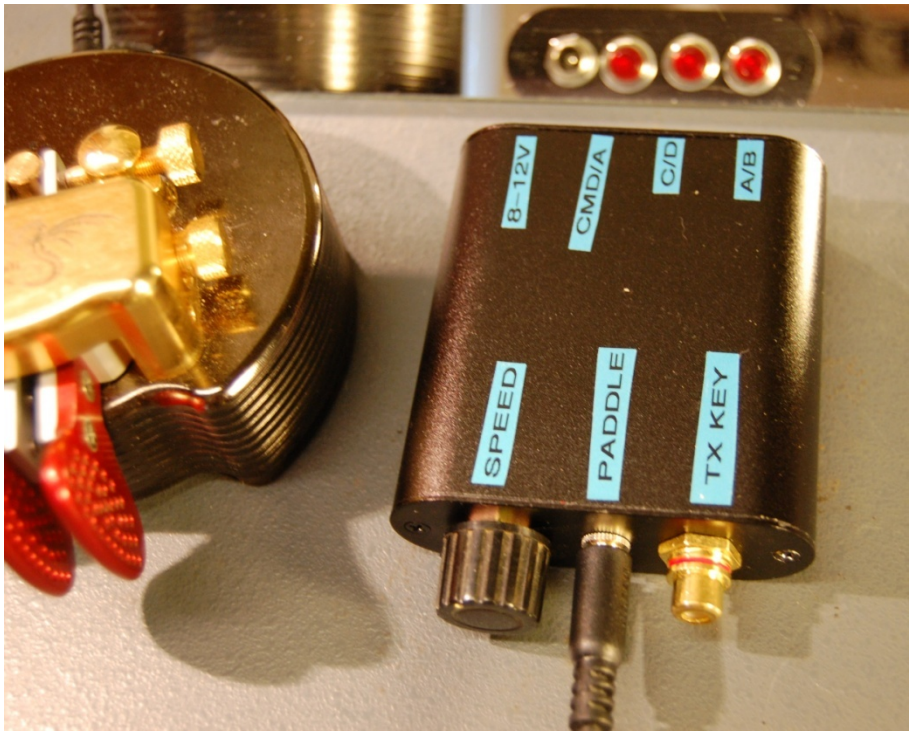
The WA5BDU Arduino Keyer



The WA5BDU Arduino Keyer is intended to be a simple keyer that can be implemented and operated with a minimum of hardware overhead. At its most minimal, only a paddle would be required. Arduino boards can be had in versions as small as 0.7” x 1.25”, so the physical space requirement is also modest.

The keyer can be controlled by commands sent via the paddle, so a lot of features can be (and are) included despite the small hardware footprint. The plan was to make a simple and physically small keyer that could be built into a homebrew transceiver or transmitter project without adding much to its complexity or size.

As features have inevitably been added, the original vision of a minimalist keyer has been maintained, but now the option of expanding it to a stand-alone unit with traditional controls has been realized.

Ease of use can be increased by adding a speed control pot and one or more pushbuttons as desired.

Features

Parameters such as speed, sidetone pitch, keyer mode A or B and others can be configured in real time via the paddle. See the “Commands” section for full information.

There are four 75 character message memories which can be sent by paddle command or pushbutton. One message can call another.

The keyer detects stuck paddle and locks out keying after 255 elements have been sent. Clears when paddle contacts open.

A hand key mode allows using a straight key or tapping the paddle to send manual Morse.

Generates a sidetone which can be wired to a miniature speaker or connected into a radio's audio chain. The pitch is adjustable and the sidetone can be muted by a command sequence.

Keying logic is iambic with mode A or B selectable.

Decoded text is sent to the serial port, along with keyer parameter information.

The keyer has the ability to control a T/R relay. The user can set the time allowed for the relay to close before energizing the transmitter, and the hold time before the relay is allowed to drop out after sending stops.

A PTT input is available for users who might use the keyer for T/R control and work modes other than CW.

The Arduino board can be powered by a 5 VDC supply or by 8 to 12 VDC unregulated.

Using the keyer

Commands

The command mode is entered by sending the wait [AS] sign, which I translate to an asterisk. There's a small blip of acknowledgement on the speaker and the transmit line is disabled so the rest of the command won't be sent on the air. Why did I use [AS] for this? Well, if you're in QSO and are going to pause to address the keyer, it tells the other guy to stand by for a second. And if you really just meant "standby" and didn't want to be in the command mode – I've kept 'E' as an invalid command so just sending a single dit exits back to the normal keyer mode.

Again, if you choose to install PB3, you don't need the [AS] thing.

After the command mode has been entered, one or more additional characters are sent to complete the command sequence. After entering the command mode, the keyer speed is reduced to 22 WPM if it is currently set faster than that.

S – Speed

After the 'S', send two digits to indicate the desired speed. If a non-numeric character is entered (other than 'Q'), there is an error beep and the keyer waits for an acceptable entry. After a successful speed entry, the keyer sends 'R'.

To cancel the speed function after entering it, send 'Q'.

After a speed has been entered from the paddle, the speed pot will likely disagree with the programmed speed, so it will be ignored. To restore the speed pot function, just turn the pot about 10% of travel and it will be re-enabled.

A/B – Mode

To specify keyer Mode A or B, simply enter the command mode and send 'A' or 'B' as desired. The keyer will respond by sending 'OK' and 'A' or 'B' as applicable.

P – Pitch

To change the sidetone pitch, enter the command 'P', then press the dot paddle to increase the pitch or the dash paddle to decrease it. Sample tones are sent while the paddle is held. When the desired pitch is reached, close both paddles to exit the pitch command routine.

H – Hand Key

To enter the hand key mode, send the 'H' command. After the mode is entered, either the dot or the dash contact can be used for manual keying. To exit the hand key mode, close both paddles simultaneously. Note that characters aren't interpreted in hand key mode, so you must exit the mode to send commands to the keyer.

T – TX Line Logic

This determines the state of the keyed line output under Key Down conditions. Send 'T' followed by 'H' for HIGH or 'L' for LOW. The keyer will respond by sending "TX HI" or "TX LO" to acknowledge the change. The standard for a keyer is to go LOW on Key Down, which is what a bug or straight key would do. But often a transistor is used to interface between the keyer and transmitter and that usually inverts the logic.

Q – Quiet (mute sidetone)

Enter the command mode and send 'Q' to mute the sidetone generated by the Arduino. Note that it is re-enabled during command mode operations. Entering command mode and sending 'Q' again will re-enable the sidetone. This may be difficult to do unless you have an alternate means of monitoring your sending such as a transceiver with sidetone. One configuration of the keyer has a pushbutton for entering command mode, eliminating this problem.

I – Information request

Enter the command mode and send 'I' and the keyer will respond by sending a list of its current parameters and settings in Morse and via the serial port. After the command is entered, the keyer responds with this information in Morse and similar via the serial port:

nn WPM	(Keyer speed)
nnn Hz	(Sidetone pitch)
MODE A(B)	(Keyer mode)
TX LO (HI)	(Keyed line logic)
T/R ON (OFF)	(T/R relay control enabled)
POT (Y/N)	(Speed pot in use or not)
RPT DLY 3.3	(Delay for message repeat function)

REV 1.01 (Keyer firmware revision)

It takes a while to send the report. To cancel during the report, hold the dot or dash paddle until you hear [AR] and the rest is omitted.

V – Save setup to EEPROM

Enter the command mode and send 'V' to save keyer setup to EEPROM memory. The following parameters are saved:

Speed, Sidetone Pitch, Key Down: High/Low, Sidetone: Off/On, Keyer Mode: A/B, Speed Pot: Y/N, Control T/R relay: Y/N, Message Repeat Delay Time

After completion of the 'V' command, the keyer responds with 'OK' in Morse.

The parameters are loaded each time the keyer is energized or reset. Note that message text is saved immediately after recording.

Hold both paddles closed during boot-up to prevent loading parameters from EEPROM.

R – Record a message

Enter the command mode and send 'R' plus a memory ID letter A, B, C or D. Send your message with the paddle. After a word space is detected, a short beep is sounded. You can pause between words and only one word space will be recorded. When you've finished recording the message, send the End Of Message character which is [EOM] sent as one, or "._ _ _ _ _" .
You can also think of it as [JM] if that helps to visualize the sound in your mind. The message is then written to EEPROM. You can also finish the message by pressing any button.

Again, if you have any buttons installed, pressing one is an alternate way to end a message recording.

If you should make an error you can also cancel message recording during sending. Just send the "quit" special character which is [QT] or " - - . - - " .

So in summary there are two special characters which may be used in message recording: [EOM] to indicate end of message and [QT] to cancel a message in progress.

Messages can be up to 75 characters long, including spaces.

M – Send a stored message

Enter the command mode and send 'M' plus a memory ID letter A, B, C or D. The keyer will then send the message. Close either paddle or press any button to cancel sending. With pushbuttons installed, just tap or hold the appropriate button to send the desired message.

Message A can be programmed to repeat indefinitely with a specified pause between message ending and starting again. See command **L**.

L – Specify message A repeat time and enable repeats

A repeating message might be used for a CQ in a contest, or for a beacon.

Enter the command mode and send ‘L’ plus two digits representing seconds and tenths of seconds delay between repeats of message A. For example L34 sets the delay at 3.4 seconds. Entering a value enables the repeat function. Entering a value of 00 turns the function off. (Enter both zeroes.)

This parameter is saved when you use the V command to save setup to EEPROM. It is also reported when you invoke the ‘I’ (Information) command.

After entering the L command, entering ‘Q’ for either digit will cancel the command.

While the message is repeating, pressing any button or paddle contact will cancel it. Cancelling is most responsive in the pause between iterations. Otherwise, the action must occur during a character space.

K – Enable/disable T/R relay control

This is a toggle command. Enter the command mode and send ‘K’ and the T/R relay control function will go to the opposite of its current state of ON or OFF. Here ON and OFF don’t refer to the state of the relay but to whether or not relay control will be performed. The user will set the initial delay time and drop-out time in the source code before compiling the program. The effect of having the function ON is that the first Morse element will be shortened by the length of the initial delay time. I haven’t found it to be noticeable at speeds up to 30 WPM.

If you don’t edit the source code, the default initial delay time is 15 ms and the hold time is 11 Morse spaces (about 1.5 word spaces).

After entering the ‘K’ command, the keyer will announce the new state of T/R control in Morse and via the serial link.

C – Compose message on serial terminal

Connect the keyer to a serial terminal via USB. Enter the command mode and send ‘C’ on the paddle. Then on your serial monitor or terminal, type in your message. The first letter must be the message ID, A – D. Press enter or send to send the message to the keyer and it will be programmed into the desired message memory.

Entering long or tricky messages using the paddle can be difficult. I leave it to the user to decide whether connecting the keyer to a serial terminal makes the task easier.

Optional pushbuttons

One to three pushbutton switches may be used with the keyer to simplify operations. But everything that can be done with the pushbuttons can be done without them as well.

Each switch provides two functions. A brief closure provides the “Tap” action and closure for 0.5 seconds or more gives the “Hold” action.

The simplest step up would be to add one pushbutton, PB3. A tap of this pushbutton will cause entry into the command mode without the need to send a special symbol from the paddle. A long press will send message A.

Two more pushbuttons can be added for easier message initiation, giving access to all four messages. Audio feedback is given with a short beep when the pushbutton is initially closed and two short beeps after it has been held long enough for the "hold" logic status.

Pushbutton summary:

Switch	Tap Action	Hold Action
PB1	Send Message A	Send Message B
PB2	Send Message C	Send Message D
PB3	Enter Command Mode	Send Message A

If only one pushbutton is installed, PB3 is the best choice since it allows entering the command mode without sending the [AS] character and it gives you one message.

Hardware information

Arduino pin usage

	D1/TX	1	30	Vin	Power in, 8 to 12 VDC
	D0/RX	2	29	GND	
	RESET	3	28	RESET	
	GND	4	27	+5V	Power in, 5 VDC
(Pushbutton 3) Pushbutton 1	D2	5	26	A7	
(Pushbutton 3) Pushbutton 2	D3	6	25	A6	PTT input
T/R relay control	D4	7	24	A5 / SCL	
Paddle dot contact	D5	8	23	A4 / SDA	
Paddle dash contact	D6	9	22	A3	
Speaker	D7	10	21	A2	
Keyed line out	D8	11	20	A1	Speed Pot
T/R relay control	D9	12	19	A0 aka 14	
LED on board	D10	13	18	AREF	
	D11mosi	14	17	3V3	
	D12miso	15	16	D13sck	

A **minimal configuration** would include a power source, a paddle and either a speaker or something to produce sound via the Keyed Line connection.

Components and connections which may be used are discussed below:

Power Source - The Arduino Nano I'm using can be powered with 5 VDC at the +5V input or by approximately 8 to 12 VDC at the Vin pin. The bare Arduino Nano draws about 20 mA from either

source. Note that 3.3 Volt Arduinos are also available. The keyer can also be powered from a USB connection.

Speaker – I use a miniature speaker or sounder P/N CEM 1201(50) available from Mouser, Digikey and similar sources. It is a plastic cylinder about ½ inch in diameter and more properly called a sounder than a speaker since the response is very "peaky". This is a 50 Ω device so being mindful of the Arduino's 20 mA maximum output, I have a 200 Ω resistor in series with it. This also keeps the volume low.

Paddle - Connect the dot contact to D5, the dash contact to D6 and the ground contact to ground. In a permanent build, you'll want a 3.5 mm or ¼ inch jack with the dot contact to tip and dash to ring.

Speed pot – I use a 10 kΩ linear pot. I think anywhere from 5 kΩ to 25 kΩ should be OK. Connect the wiper to A1, the CW terminal to +5 V and the CCW terminal to ground.

Pushbutton switches – Switches are momentary and normally open. PB1 connects from D2 to ground and PB2 connects from D3 to ground. For PB3, two small signal diodes (1N914, 1N4148, etc) are connected from D2 and D3 to one side of PB3. The other side goes to ground. The banded ends (cathode) of the diodes go toward the switch. For a minimal one-switch configuration, go with PB3. That will give a command mode entry action plus one message memory.

Line out – This line (D8) keys your transmitter. While it could directly key a modern transmitter having 5 VDC or less at the key jack, it's better to isolate the transmitter from the keyer with a transistor or MOSFET. See the schematic. Also note that pin D10 is keyed along with D8. It connects to the on-board LED on the Arduino but can still be used to control external devices. D10 is always High on Key Down and Low on Key Up while D8 can be configured to have High or Low Key Down logic.

T/R relay control – if you'd like your keyer to control a T/R relay with timing to avoid hot switching and relay clattering, use the D4 line to operate the relay. It is normally set up as HIGH in the transmit mode and LOW in receive and should be used to control a transistor or MOSFET which switches the relay.

Other features and general discussion:

Stuck paddle detection and lockout. The keyer will detect when the count of a continuous stream of dots, dashes or both exceeds 255 and then will go into a lockout (key up) condition until it detects both paddle inputs open.

Serial port use. The design philosophy of the keyer was to keep it simple and avoid using a display. But I included serial output as a development aid and it can be useful or amusing to play with at other times. If the keyer is connected to a PC via USB and the Arduino IDE is running, you can turn on the serial monitor by pressing CONTROL-SHIFT-M. On startup the keyer will display an introductory message including revision number, WPM, and the contents of all message memories.

It's not necessary to use the Arduino IDE as a serial terminal. I've also tried it with Tera Term and that worked OK too. Any serial terminal set at 9600 baud should do OK.

Text sent from the paddle will be displayed on the monitor. The Arduino monitor doesn't word wrap, so I've coded the CW prosign [AA] to generate a line feed by sending ASCII 10 to the serial terminal. Old CW

traffic handlers may recall that [AA] was used in formal messages to break between the lines of the address.

Power ON reminder. If the keyer is to be powered from batteries, a reminder that it is powered ON can prevent leaving it on and wasting batteries. One option would be to have an LED show that power is on. The keyer's software includes another method: After 15 minutes of no activity, it will send "ESE" or "shave and a haircut" on the speaker (if installed). The pitch jumps a bit to make the sound distinctive. This repeats every 15 minutes unless there is user activity via the paddle or push buttons. The Key Out line is inhibited so the transmitter won't be keyed by this alert.

More about message recording. The keyer sends a fast double-beep after a word space has been recorded so you'll know it happened. The first time I used a keyer with this feature, I found it distracting. But it's actually useful. If you inserted a space and didn't mean to, you may want to start over. But if the space was intended you can pause before sending the next word because the keyer is in a paused state. This actually takes the pressure off when recording a message. Send each word, then pause before sending the next one.

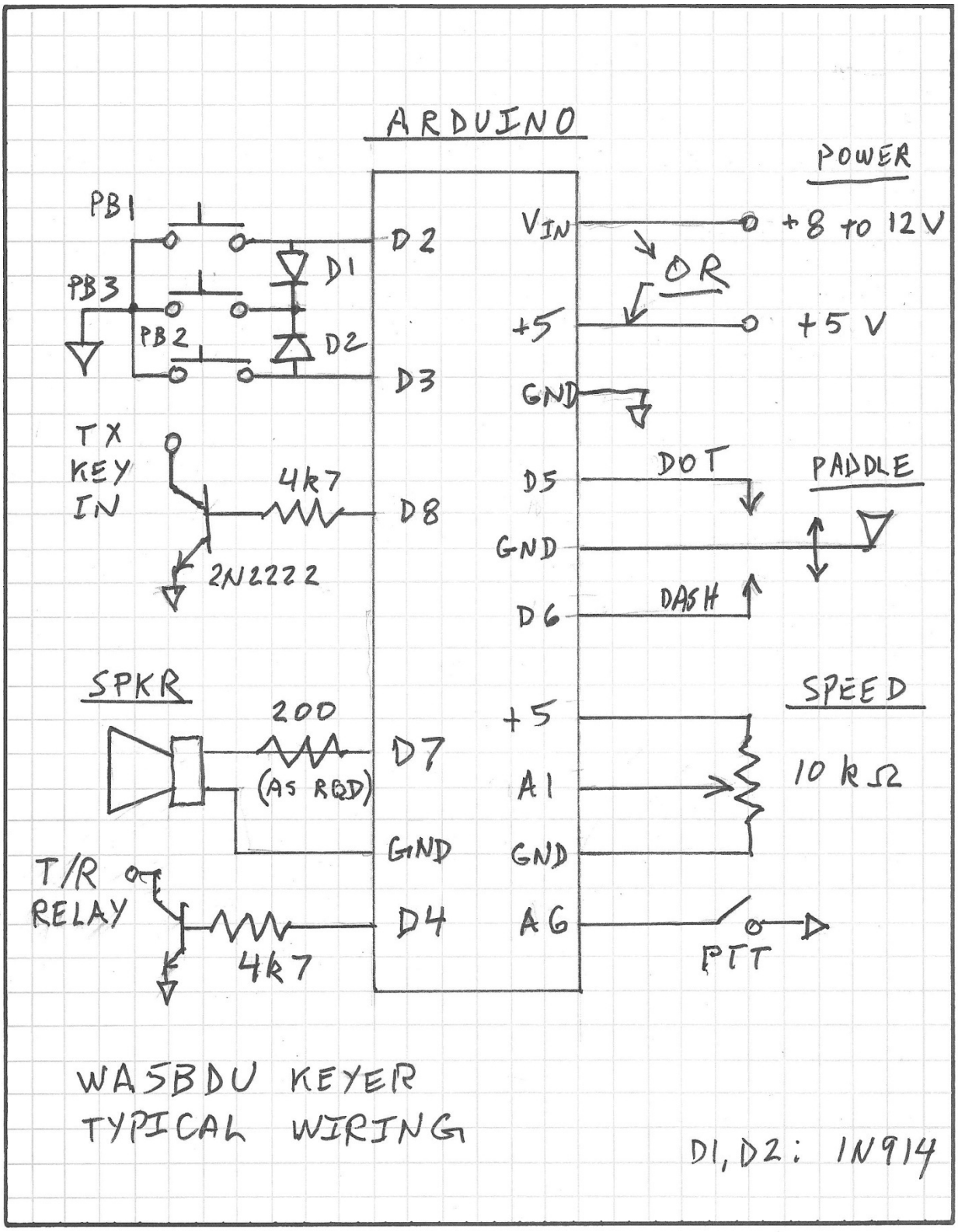
If you want to record a command within a message, send the [AS] sign and then the character(s) which form the command sequence. Most common would be to change speed, as in '*S25' or to call another message as in '*MB'. Remember that the [AS] prosign prints to the serial monitor as '*' on this keyer.

About embedding messages within messages. You can embed messages to a depth of one. For example, message A can call message B. When you send message A, the text will be sent until encountering the command to send message B, which will then be sent. The keyer will then return to message A and finish that message.

The restriction is that message B cannot call another message. It's OK if A calls B, then calls C (or B again) and so on. But having A call B and B call C will not work as it would be a form of stack overflow.

A note on text formatting in this document: In Morse, various non-text prosigns are indicated by showing two letters shown sent as one. As in AR for "end of message". The standard is to draw a line across the top to show that they letters are sent with no space. My word processor can't do that so I use brackets instead, as in [AR] for "end of message" or [AS] for "wait, standby".

Schematic of keyer with full configuration:



Appendix I

Parameters the user may wish to change in the source code before compiling

You don't have to be a programmer to make minor changes in the source code. Some things you might want to change to customize the keyer more closely to your needs are described below. Variables and constants the user might want to change are marked in the source code with "**USER**".

T/R relay timing

If you are using the keyer to control a T/R relay, you may want to change the numbers in the following two lines:

```
uint8_t TXOnDelay = 15; // milliseconds from TX relay to close key
uint16_t TXHold = 11; // time in Morse space units until drop out T/R
```

The first parameter is the time in milliseconds from energizing the relay control time until enabling the Key Out line. I'm using a relay with a close time of 15 ms. Another relay I've used for T/R control has a close time of 7 ms. Check the specifications on the relay you choose and set TXOnDelay appropriately.

The second parameter is in units of Morse space (or dot) time so that it will be relative to sending speed. Recall that a word space is 7 Morse spaces long. I think you'd want to allow at least that much time to prevent excess relay clicking. If you want to think in terms of time, recall that the duration of a space is 1200/WPM. With my specified time of 11 and at 20 WPM (60 ms dots), my drop out time is 0.66 seconds at 20 WPM.

Other items related to T/R control are shown below:

```
uint8_t TX_RX_Rstate = LOW; // line state while NOT in transmit
uint8_t TX_RX_Tstate = HIGH; // keep these two negative of each other
bool controlTR = false;  // make true to enable T/R relay control
```

The first two items control the state of the relay control line in the receive and transmit modes. Most often, HIGH would be used for transmit because it would energize a transistor or MOSFET which would switch the relay ON. If some other logic is used, be sure to make the two opposite to each other.

The 'controlTR' flag is saved to EEPROM so it's not necessary to edit it in source code. However if you're sure you want T/R control enabled you might change its value to 'true'.

Some other values that can be saved to EEPROM after you set them to your liking are listed below. Again, you may change the default values in the source code if you wish:

```
unsigned int pitch = 600; // sidetone pitch
char PitchString[] = "600 HZ"; // text field to announce sidetone peak
uint8_t keyDNstate = LOW; // state of Key Out line when in key down state
uint8_t keyUPstate = HIGH; // must be opposite of above
```

The flag below is not saved to EEPROM but if you do not want the “battery saver” reminder after 15 minutes of no activity, change ***true*** to ***false.***

```
bool OnIdleAlarm = true; // make 'false' for no "power on" reminder
```

These values for sidetone pitch and Key Down (Up) state can be set with the ‘P’ and ‘T’ commands and made permanent with the ‘V’ command, but you may alter them if you wish so they’ll be as desired even before saving to EEPROM.

73 & enjoy this keyer,

Nick Kennedy, WA5BDU

kennnick@gmail.com