SO2R Box Plus – notes

The SO2R box provides a keyer, SO2R switching for headphones, keyer, PTT, microphone, RTTY keying, and aux outputs. It can connect to up to four radios. It can be controlled via USB. It also has front-panel controls so the station can be used when the computer is not running.

Features

Keyer

The keyer has the following capabilities:

- Speed from 2-99 WPM. The speed can be set from the computer or from the speed control on the SO2R box. The speed control is limited to 5-49 WPM.
- Iambic, Ultimatic, dit preference, dah preference, or straight key.
- Timing options including A and B and loose or tight timing.
- Six or seven dit word spacing.
- Paddle reverse
- Auto space

Keyer type, timing, spacing, and reverse are remembered when the SO2R box is turned off.

SO2R

The SO2R box handles headphone, keyer, and PTT switching between four radios, two at a time. It also provides outputs to control additional switching.

There are three modes of headphone switching:

- Normal mode is similar to what a DX Doubler does. The headphones can be connected to radio 1, radio 2, or stereo where radio 1 is connected to the left headphone and radio 2 is connected to the right headphone.
- Spatial mode is where the left headphone outputs radio 1 and the right headphone outputs radio 2. Stereo is the same as in normal mode.
- Symmetric mode is where both the left and right headphones output the same audio.

In addition the left and right headphones can be blended – some of output of each headphone is fed into the other. The amount of blending can be set from the computer. Blending is only used in normal and spatial modes while in stereo and it can be turned off entirely.

The selected mode and blending amount are remembered when the SO2R box is turned off.

PTT can be turned on and off by the computer or by a footswitch or by sending CW.

The computer can select radio 1 or radio 2 for receive and transmit and stereo for receive.

These selections can be overridden by SO2R box switches. If the SO2R box transmit switch disagrees with what the computer has selected the computer will not be allowed to send via the keyer or to set PTT on.

Microphone

Isolated microphone switching is available. Both the hot and ground connections are switched.

SO4R

Up to four radios can be connected to the SO2R box (hence the "plus" in the name). Any two can be used for SO2R. The computer can change which two are used and can set which two are used if the SO2R box is operated without a computer.

Auxiliary Outputs

16 bits of output are provided. These are similar to what is traditionally available on LPT ports. They can be used to drive band decoders such as those from Top Ten Devices and they can be used to control items such as outboard voice keyers or antenna switches.

RTTY

Four auxiliary outputs can be used for RTTY. The baud rate and stop bits can be set for each radio. A level converter is needed to connect the RTTY output to most radios. There are no provisions for RTTY input. Most people use a computer and sound card to demodulate RTTY.

Schematic

The schematic is in three pages, one for the digital circuitry, one for the audio circuits, and one the additional circuitry for the third and fourth radio and the microphone switching.

Digital Circuitry

The PIC 18F4550 microprocessor is the brains of the SO2R box. It was chosen because Microchip has the only family of USB processors that come in DIP packages. It is a fairly powerful processor and is overkill for this application.

Auxiliary outputs are provided by two 74HC594 shift registers.

Key and PTT outputs are isolated by optoisolators. It is possible to wire these lines so that no ground current can flow between radios through the SO2R box. The first production runs of the SO2R Box used PS2502-4 Darlington optoisolators. It was discovered that if the output impedance is high the turn-off time of this device could be

long enough that it noticeably affects the weight of the keying. Later production runs use bipolar output optoisolators.

Audio Circuitry

The three SO2R modes involve a total of seven different possibilities for connections to the left and right headphones.

The MAX4574 switch selects the audio inputs. This part was chosen because it has a "clickless" mode which switches softly and sounds better than other analog switches or relays that were tested.

The MAX9722A is used as an audio amplifier. This chip was designed to be a headphone amplifier. It generates a negative voltage and does not require output capacitors. It is protected against short circuits and ESD.

The MCP41010 is used as the blend pot. It can be controlled from the microprocessor and is less expensive than a "real" potentiometer.

Transformers are used for all audio inputs and grounds are isolated. There are 1 Meg resistors to ground to avoid static build-up.

The audio ICs are all 5 volt parts. The audio switch in particular will not handle signals above +5V or below 0 volts gracefully – large signals produce breakthrough and very poor sounding audio. To avoid this problem the transformer taps are used to reduce the voltage by half and the output amplifier is set for a gain of two. The load resistors drop the voltage from normal headphone outputs. The worst-case test radio I tried was an IC-706 IIg which delivers in excess of 12 volts on the headphone output if it is unloaded. With the resistors used it is dropped to below 10 volts and there are no breakthrough or distortion issues.

Microphone Circuitry

Microphone switching is done with relays. The microprocessor controls them through switching transistors.

Connectors

As shown in the schematic there are twelve connectors:

Function	Connector
Power	2.1mm concentric
USB	4 Pin type B
Paddle	1/4" Stereo
PTT	RCA Phono
Aux	DB-25
Stn 1	8 Pin DIN
Stn 2	8 Pin DIN
Stn 3	8 Pin DIN
Stn 4	8 Pin DIN
Headphones	1/4" Stereo
Microphone	1/8" mono

The per-radio connections are on the 8 pin connector. This is the pin-out:

1	CW/PTT common
2	Receiver main (left) audio
3	Receiver secondary (right) audio
4	PTT
5	Audio common
6	CW Key
7	Mic
8	Mic ground

The shield is grounded

Note that the pin numbers of DIN connectors are not consecutive, that is they are not numbered 1-2-3-4-5-6-7 but 7-3-5-2-4-1-6. Pin 8 is in the middle.

The auxiliary outputs and additional connections are on the DB-25 connector. This is the pin-out:

1-4	Aux 1
5-8	Aux 2
9	Ground
10	Force receive on station 1 (pull low)
11	Force receive on station 2 (pull low)
12	Force transmit on station 1 (pull low)
13	Force transmit on station 2 (pull low)
14	High if transmit on station 2
15	High if receive on station 1
16	High if receive on station 2
17	+12 volts switched
18-21	Aux 3
22-25	Aux 4

Don't try to draw much current from pins 14-16. You can take significant power from pin 17 – if you increase the current consumption enough that it blows the fuse you will need to put in a larger fuse.

Schematic Notes

The digital and audio circuits are on one PC board. The "W" connectors indicate places where a pad was placed on the board for future expansion.

Computer Interface

The computer communicates with the SO2R box through a USB interface. Unlike other SO2R boxes, in particular the Microham MK2R and Ham Radio Solutions EZMaster which use the CDC (USB serial port) interface class the SO2R box uses the HID (Human Interface Device) interface class.

The HID class has several advantages. There are no drivers needed, as Microsoft supplies them (USB keyboards and mice also use the HID class). There is no router or serial port configuration needed. And HID messages can be multiple bytes, simplifying protocol and synchronization.

The disadvantage is that logging software must be coded to know how to talk to the HID interface.

Communicating with HID in Microsoft Windows is a three step process.

1 Find the name of the device

2 Open the device using the standard Windows CreateFile call.

3 Read and write using the standard ReadFile and WriteFile calls.

Step 1 is the complicated part. Rather than explain it, the best thing to do is to go to http://www.janaxelson.com/hidpage.htm where there are examples in Visual Basic .NET, Visual Basic 6, Visual C++ 6, and Visual C#. The SO2R box test program and interface programs use code from the Visual C++ example.

All messages to and from the SO2R box are three bytes long. The first byte is the report number which is always zero. The second byte is the command and the third byte is the data for the command. Note that this is also true in the examples on the website shown above. The SO2R box code is loosely based on the PIC18F4550 example and the sample programs are almost enough to run it.

All USB devices must have a unique combination of vendor ID and product ID. For the SO2R box the vendor ID is 16c0 and the product ID is 065e.

Firmware Update

The firmware (microprocessor code) can be updated through the USB bus. The SO2R box test program can do this. The firmware update bootloader uses a different product ID and programs which communicate with the SO2R box do not need to be concerned with firmware update mode.

Protocol

NOTE: This is current as of V1.5-0 firmware

This description uses the following conventions:

- Bits within a byte are numbered 0 through 7, with 7 being the most significant bit.
- Numbers are decimal except where proceeded by 0x in which case they are hexadecimal.

All commands have a command byte and a data byte.

Most messages from the computer will cause a response from the SO2R box. In addition the SO2R box will send messages for various events. These do not require a response from the computer.

Some responses contain a status. Here are the defined status values:

0	Success
1	Bad value
2	Busy
3	Late

A message with an unknown command will cause a response with a command of 0x7f and the value will be that of the command sent.

These are the various commands:

Inquiry

The argument for the inquiry command is a command number. The command specified with its value will be the reply.

If the value of the inquiry command is inquiry the returned value will be the SO2R box firmware version number in BCD format. This should be the first command. The current version is 1.3.

Box Patch Level

This command requests the SO2R box's patch level. The value in the command is ignored. The patch level will be incremented as bugs are fixed or minor features added.

The same response will be sent if an inquiry command is sent with the value being the Box Patch Level command.

Box Version Special

This command requests the SO2R box's special version level. The value in the command is ignored. For now the response value should be zero.

The same response will be sent if an inquiry command is sent with the value being the Box Version Special command.

Box Update

This command sets the SO2R box into firmware update mode. If successful no response is sent and the SO2R box will disconnect. If the value is not '!' a response with a status of Bad Value will be returned.

Box Reset

This command resets the EEPROM values to the defaults. If successful no response is sent and the SO2R box will disconnect. If the value is not '$' a response with a status of Bad Value will be returned.

Keyer Status

This command returns information about the current state of the keyer.

Bit	Meaning
7 – 5	Reserved, must be zero
4	Buffer is empty, OK to send a character
3	Sending from computer is disabled
1 – 0	Keyer state

Buffer is empty is set when there is no character in the keyer's buffer. A character can be sent from the computer.

Sending from computer is disabled is set when the transmit station is set from the switch on the SO2R box and the station selected by the computer is not the one selected from the switch.

Keyer state is the current state of the keyer. The possibilities are:

0 Keyer is idle

1 Keyer is sending from the paddle

2 Keyer is sending from the computer

3 Keyer is tuning (key down)

Keyer Speed

This command changes the keyer speed. The value is a number in the range 2-99. The keyer speed is immediately changed to that speed.

If the speed is changed using the SO2R box speed control the SO2R box will send an unsolicited Keyer Speed message with a value of the new keyer speed.

The response to a query command for Keyer Speed is the current keyer speed.

This command ignores any delta speed which may be in effect. The speed returned does not include the delta.

Keyer Configuration

This command sets the configuration of the Morse code keyer. It is stored in non-volatile memory.

Bit	Meaning
7	Keyer sets PTT
6	Tight paddle timing
5	Type A keyer timing
4	Seven dit word spacing
3	Paddle reverse
2 – 0	Keyer type

If Keyer sets PTT is set the keyer will set the PTT line when sending to a radio. If this is not set the only way to activate the PTT line is through the footswitch or a computer command.

Tight paddle timing affects when a paddle being pressed is latched for the next dit or dah. Loose timing (value 0) allows earlier latching. The effect is reduced at higher speeds and has no effect at very high speeds.

Type A keyer timing sets what is commonly called Type A keying. Zero is type B keying.

Seven dit word spacing sets the time between words to seven dits. Zero is six bits.

Paddle reverse changes which paddle is the dit paddle and which the dah.

Keyer type determines what the keyer sends when both paddles are pressed. The choices are:

0 Iambic – send dit if dah was just sent and vice versa.

1 Ultimatic – send whichever paddle was pressed last.

2 Dit – send a dit always

3 Dah – send a dah always.

4 Straight Key – Send when the dit paddle is closed, ignore keyer speed etc.

The response to this command and to a Query command for Keyer Status is the current status after the command has been processed.

Keyer Configuration 2

This command sets additional configuration of the Morse code keyer. It is stored in non-volatile memory.

Bit	Meaning
7 – 1	Reserved, must be zero
0	Autospace

If autospace is set the keyer will force a full character space between characters sent from the paddle.

Keyer Character

This command sets the next character to be sent. The value is the character.

Valid characters are shown in appendix 1.

The keyer buffers one character. Once it has started sending a character it is ready to receive the next.

The response to this command is a status. Valid responses are:

Success	The keyer accepted the character.
Bad Value	The keyer cannot send that character.
Busy	The keyer already has a character in the buffer.

The response to a query command for Keyer Character is the current status from the last Keyer Character command.

Keyer Overwrite

This command overwrites the next character to be sent. The value is the character.

Valid characters are the same as for the Keyer Character command except that zero is a valid value and will clear the next character so that no next character will be sent.

This overwrites the character in the buffer if there is one.

The response to this command is a status. Valid responses are:

Success	The keyer accepted the character.
Bad Value	The keyer cannot send that character.
Late	There is no character in the buffer.

The response to a query command for Keyer Overwrite is the status from the last Keyer Overwrite command.

Keyer Abort

If the keyer was sending from the computer the sending is terminated immediately. The character buffer is also cleared. If the keyer was tuning that is also terminated.

Sending from the paddle is not affected by Keyer Abort.

Note that Keyer Abort will not generate events. The abort completion will be acknowledged by the Keyer Abort response.

Keyer Abort will also be sent by the keyer if sending is aborted. The reason is sent as the data to the command. Valid reasons are:

Bit	Meaning
7-3	Reserved
2	Transmitter selection changed
1	Paddle was touched
0	Abort command was received

The response to a query command for Keyer Abort is a Keyer Abort response with a value of zero.

Keyer Control

This command controls miscellaneous keyer functions.

Bit	Meaning
4-7	Reserved, must be zero
3	Pot on
2	Pot off
1	Tune on
0	Tune off

If Pot on is 1 the speed potentiometer will control the keyer speed.

If Pot off is 1 the speed potentiometer will not control the keyer speed but will send pot speed messages when it is changed.

If Tune on is 1 the keyer starts tuning, closing the key line of the radio selected for transmitting.

If Tune off is 1 the keyer stops tuning. If 30 seconds pass or the operator presses the paddle tuning will also be terminated. If this happens the SO2R box will send an unsolicited Keyer Control response.

The response to a query command for Keyer Control will have either Pot on or Pot off set and will have either Tune on or Tune off set.

Keyer Event

This is usually not sent from the computer to the SO2R box. The SO2R box sends it as an unsolicited message when a keyer event occurs. These are the events:

0 Ignore this event

1 Keyer has sent the end of a character

2 Keyer has finished sending, and the buffer is empty

3 Paddle was pressed

The response to this command and to a Query command for Keyer Event is a Keyer Event response with a value of 0.

Note: If the character is a space the end of character event will be sent just after the keyer starts "sending" the space.

Keyer Delta

This command changes the keyer speed. The value is the amount to the base speed.

The delta is applied after the last character that has been sent to the keyer. In other words it is buffered.

Keyer PTT Pre-send

This command sets the length of time between when the PTT line is asserted and the first character will be sent by the keyer. The value can be 0-255 ms.

The response to this command and to a Query command for Keyer PTT Pre-Send will be the currently set pre-send time.

Keyer PTT Post-send

This command sets the length of time between when the end of last character will be sent by the keyer and when the PTT line will be dropped. The value can be 0-255 ms.

The response to this command and to a Query command for Keyer PTT Post-Send will be the currently set post-send time.

Keyer Weight

This command sets the keyer weight. The value is 0-255. A value of 128 will produce the normal 50% duty cycle. Varying this far from the center value is not recommended because the CW will sound lousy.

Keyer Compensation 1

This command sets the amount of time to be added or subtracted from the key down time of all dits and dahs for radio 1.

Bit	Meaning
7	Set Nonvolatile
0 – 6	Time in milliseconds (signed)

If Set Nonvolatile is set the value is stored in non-volatile memory and this compensation will be used when the SO2R box is turned on.
The response to this command and to a Query command for Keyer Compensation 1 will be the current compensation.

Keyer Compensation 2

This command sets the amount of time to be added or subtracted from the key down time of all dits and dahs for radio 2. The value and effects are as described for Keyer Compensation 1.

Keyer Compensation 3

This command sets the amount of time to be added or subtracted from the key down time of all dits and dahs for radio 3. The value and effects are as described for Keyer Compensation 1.

Keyer Compensation 4

This command sets the amount of time to be added or subtracted from the key down time of all dits and dahs for radio 4. The value and effects are as described for Keyer Compensation 1.

Keyer Potentiometer Speed

This is usually not sent from the computer to the SO2R box. The SO2R box sends it as an unsolicited message when the potentiometer is changed when the pot has been turned off with a keyer control command.

The response to this command and to a Query command for Keyer Potentiometer Speed will be the current potentiometer speed setting.

Keyer Potentiometer Min Speed

This command sets the speed at the counterclockwise limit of the front panel potentiometer.

Bit	Meaning
7	Set Nonvolatile
0	Pot min speed

If Set Nonvolatile is set the value is stored in non-volatile memory and this min speed will be used when the SO2R box is turned on.

The response to this command and to a Query command for Keyer Potentiometer Min Speed will be the current min speed.

Keyer Potentiometer Max Speed

These commands set the speeds at the clockwise limits of the front panel potentiometer.

Bit	Meaning
7	Set Nonvolatile
0	Pot max speed

If Set Nonvolatile is set the value is stored in non-volatile memory and this max speed will be used when the SO2R box is turned on.

The response to this command and to a Query command for Keyer Potentiometer Max Speed will be the current min or max speed.

Keyer Sending Character

This is usually not sent from the computer to the SO2R box. The SO2R box sends it as an unsolicited message when it starts sending a character. The value is the character which is being sent.

The response to this command and to a Query command for Keyer Sending Character is a Keyer Sending Character response with the character being sent.

Note: When the computer receives this message it means the buffer is empty and the keyer is ready for another character.

Keyer Sending Character

This is usually not sent from the computer to the SO2R box. The SO2R box sends it as an unsolicited message when it starts sending a character. The value is the character which is being sent.

The response to this command and to a Query command for Keyer Sending Character is a Keyer Sending Character response with the character being sent.

Aux 1

This command changes the Aux 1 output.

Bit	Meaning
7 – 5	Reserved, must be zero
4	Set outputs
3 – 0	Value

Set outputs and causes all aux information to be updated to the values set by the most recent Aux 1, and Aux 2, Aux 3, and Aux 4 commands. The band and aux outputs will not change values until a command has this bit set – then all values will be updated.

The response to a query command for Aux 1 is the current Aux 1 value. This may not be the value on the output pins if Set outputs was not set since the last band 1 update.

Value is the value to output.

Aux 2

This command changes the outputs for Aux 2. The value and effects are as described for Aux 1.

Aux 3

This command changes the outputs for Aux 3. The value and effects are as described for Aux 1.

Aux 4

This command changes the outputs for Aux 4. The value and effects are as described for Aux 1.

RTTY Configuration

This command sets the RTTY configuration.

Bit	Meaning
7 – 5	Reserved, must be zero
4	RTTY on Aux
3	Invert output radio 4
2	Invert output radio 3
1	Invert output radio 2
0	Invert output radio 1

If RTTY on Aux is set Aux pins 12 – 15 are used as RTTY outputs instead of their normal function.

If Invert output radio 1, 2, 3, or 4 is set the output will be high on space. Otherwise it will be high on mark.

RTTY Status

This is usually not sent from the computer to the SO2R box. The SO2R box sends it as an unsolicited message.

Bit	Meaning
7 – 2	Reserved, must be zero
1	Idle
0	Ready

If Ready is set the SO2R Box is ready to accept another character.

If Idle is set the SO2R Box has sent all characters and the RTTY output is idle.

RTTY Speed and Bits

This command sets the RTTY speed and number of bits

Bit	Meaning
6 – 7	Stop
4 – 5	Length
0 – 3	Speed

Stop is the number of stop bits. The choices are:

0 1 stop bit

1 1.5 stop bits

2 2 stop bits

Length is the length of the character. The choices are

0 5 bits

1 6 bits

2 7 bits

3 8 bits

Speed is the speed in baud. The choices are:

0 22 baud

1 45.45 baud

2 50 baud

3 56 baud

4 75 baud

5 100 baud

6 110 baud

7 150 baud

8 200 baud

9 300 baud

RTTY Character

This command sets the next RTTY character to be sent. The value is the character.

SO2R State

This command sets the SO2R state.

Bit	Meaning
7 – 5	Reserved, must be zero
4	Stereo Reverse
3	Assert PTT
2	Stereo receive
1	Receive on radio 2
0	Transmit on radio 2

If Assert PTT is set PTT line for the radio selected for transmit is activated.

If Stereo Receive is set stereo reception will be used if the current SO2R configuration allows it. Radio 1 left will be in the left channel and radio 2 left will be in the right channel.

If Stereo Reverse is set stereo reception will be used if the current SO2R configuration allows it. Radio 2 left will be in the left channel and radio 1 left will be in the right channel.

If Receive on radio 2 is set then radio 2 will be connected to the headphones. If it is not set radio 1 will be connected to the headphones.

If Transmit on radio 2 is set any keyer or PTT output will be directed to radio 2. If it is not set keyer or PTT output will be directed to radio 1.

SO2R Configuration

This sets the configuration of the SO2R box, particularly which radio is connected to the left and right headphones. It is stored in non-volatile memory.

Bit	Meaning
7	Reserved, must be zero
6	Invert PTT switch
5	Show TX
4	Microphone relays
3	Blend
2 – 0	Audio Routing

If Invert PTT switch is set PTT will be active if the rear-panel PTT connector is not shorted. If it is not set PTT will be active when the connector is shorted.

If Show TX is set Aux 8-11 will be used to show which transmitter is selected. AUX8 will be high if TX 1 is selected, AUX 9 will be high if TX 2 is selected etc.

If microphone relays is set, the relays will be activated and will connect the microphone to the transmitting radio.

If Blend is set the right and left headphone audio will be mixed when the audio routing allows it.

Audio Routing selects how the SO2R box connects the headphones to the radios. The choices are:

0 Normal

1 Symmetric

2 Spatial

The response to this command and to a Query command for SO2R Configuration is the current configuration.

SO2R Switches

This command shows the current state of the SO2R box switches and inputs

The value in the command is ignored and query with a value of SO2R Switches has the same effect as sending the command.

These are the bits in the message from the SO2R box to the computer.

Bit	Meaning
7 – 5	Must be zero
4	PTT footswitch is closed
3	The SO2R box RX switch is set to RX2
2	The SO2R box RX switch is set to RX1
1	The SO2R box TX switch is set to TX2
0	The SO2R box TX switch is set to TX1

PTT footswitch is closed if set if the SO2R box PTT connector pin is grounded.

The SO2R box RX switch is set to RX2 is set if the SO2R box switch is set to force receive on radio 2 or if the corresponding pin on the Aux connector is grounded.

The SO2R box RX switch is set to RX1 is set if the SO2R box switch is set to force receive on radio 2 or if the corresponding pin on the Aux connector is grounded.

The SO2R box TX switch is set to TX2 is set if the SO2R box switch is set to force transmit on radio 2 or if the corresponding pin on the Aux connector is grounded.

The SO2R box TX switch is set to TX1 is set if the SO2R box switch is set to force transmit on radio 1 or if the corresponding pin on the Aux connector is grounded.

If the switches or inputs are changed an unsolicited SO2R Switches message will be to the computer by the SO2R box.

SO2R Map Radio 1

This command sets which of the four stations is used as radio 1

Bit	Meaning
7	Set Nonvolatile
6 – 0	Radio number

If Set Nonvolatile is set the value is stored in non-volatile memory and this mapping will be used when the SO2R box is turned on.

Radio Number is the radio, 0 – 3, to use as radio 1.

The response to this command and to a Query command for SO2R Map Radio 1 is the radio 1 mapping. The EEPROM mapping is in the high four bits and the current mapping is in the low four bits.

SO2R Map Radio 2

This command sets which of the four stations is used as radio 2.

The arguments are the same as for SO2R Map Radio 1.

SO2R Startup

This command sets the behavior of the SO2R Box when it is powered up before a computer connects to it.

Bit	Meaning
7 – 1	Reserved, must be zero
0	Stereo

If Stereo is set when the box is powered up and the RECEIVE switch is set to AUTO the box will select stereo. If Stereo is not set and the switch is set to AUTO the box will select radio 1.

Programming Notes

For a minimal implementation the SO2R Switches messages can be ignored. A fancier implementation could set transmit or receive focus to whatever is set on the SO2R box.

The keyer and SO2R configuration are stored in EEPROM. A minimal implementation could ignore them. The user could set them as desired using the SO2R box test program and they won't change. A fancier implementation could set them, or could keep them as per-operator data so switching operators brings in the operators preferred configuration.

The SO2R state is similar to what today exists on a parallel port.

The aux outputs do not have to be used in groups of 4 bits. They are really 16 programmable outputs. However the commands to set them do so 4 bits at a time. If a larger grouping is used the Set Outputs flag should be set only when sending the last 4 bits. This way the update will be atomic.

The keyer's one character buffer allows plenty of time to send characters without interruption – USB is fast but it is basically a network actual speed may vary. The Keyer Sending Character message will tell the logging program when to send the next character.

If the text being sent can be edited while sending is going on and it is desired to allow editing as late as possible the Keyer Overwrite command can be used to change a character already sent. If it is too late the command will return an error and the logging program will know it was unable to make the correction. This should allow editing up to the last few milliseconds.

Using the buffer and overwrite for editing is fairly complicated. An easier implementation would be to just send the next character when the end character event is received. There is plenty of time for USB to send a character during the inter-character space.

The delta speed is buffered along with the characters so that the "right thing" will happen if Keyer Delta commands are interspersed with Keyer Character commands. Also Keyer Delta commands are absolute and will overwrite any previous number sent. For example the keyer sequence:

Character A

Delta +2

Character B

Delta -2

Character C

Delta -2

Delta -2

Character D

Will result in B being sent 2 WPM faster than A, C being sent 2 WPM slower than A, and D being sent 2 WPM slower than A.

Delta speeds can be sent any time whether the buffer has a character or not.

A simple implementation could send the current delta speed before each character.

The SO2R box asserts PTT when sending from the keyer if the keyer PTT configuration bit is set. If the user prefers to control PTT the footswitch will work fine. Otherwise CW VOX will work. If the computer wishes to assert PTT when sending from the keyer it can do so. It can tell when the sending is complete because it will receive a Keyer Event message.

The SO4R commands map one of the four radios to TX1/RX1 and one to TX2/RX2. This way an auxiliary program can select the radios, since no logging program I know of can handle more than two radios at a time.

SO2R Box Interface Program

In order to make the SO2R box immediately useful with popular contest programs an interface program has been written. It connects to two virtual COM ports. It communicates on one COM port using the Open Two Radio Protocol (OTRSP). On the other COM port it partially emulates a Winkey keyer. (Winkey is a trademarks of K1EL who allowed me to use the protocol he developed).

The program implements the OTRSP protocol as of V0.7 within the limits of the hardware.

It also emulates the 0x00 subcommands 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, and 0x09 and the commands 0x01, 0x02, 0x03, 0x04, 0x05, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0e, 0x0f, 0x13, 0x15, 0x16, 0x18, 0x1c, 0x1e, and 0x1f sufficiently that logging programs such as N1MM Logger, Win-Test, and Writelog work well. Other commands are consumed but ignored.

Note that the Winkey has capabilities which the SO2R box does not, and commands which deal with these capabilities may have no effect.

Appendix 1 – Keyer characters

Value	Character	Sent
32		word space
33	!	..--.
34	"	.-..-.
35	#	dot space
36	$	...-..-
37	%	three dot space
38	&	..--
39	'	.----.
40	(	-.--.
41	)	-.--.-
42	*	.-.-
43	+	.-.-.
44	,	--..--
45	-	-....-
46	.	.-.-.-
47	/	-..-.
48	0	-----
49	1	.----
50	2	..---
51	3	...--
52	4	....-
53	5	.....
54	6	-....
55	7	--...
56	8	---..
57	9	----.
58	:	---...
59	;	-.-.-.

60	<	.--.-
61	=	-...-
62	>	--.--
63	?	..--..
64	@	.--.-.
65	A	.-
66	B	-...
67	C	-.-.
68	D	-..
69	E	.
70	F	..-.
71	G	--.
72	H	....
73	I	..
74	J	.---
75	K	-.-
76	L	.-..
77	M	--
78	N	-.
79	O	---
80	P	.--.
81	Q	--.-
82	R	.-.
83	S	...
84	T	-
85	U	..-
86	V	...-
87	W	.--
88	X	-..-
89	Y	-.--
90	Z	--..

91	[	...-.-
92	\	..-..
93	]	.-...
94	^	-.-..-
95	_	..--.-
96	`	---.

Appendix 2 – protocol.h

```
/* Copyright 2006, 2018 Paul Young
 *
 * Protocol
 *
 * Author                Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Paul Young            12/05/06    Original.
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Paul Young            1/19/08     Added Keyer control.
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Paul Young            12/21/08    Added radio map.
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Paul Young            4/16/09     Misc changes for emulator.
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Paul Young            8/20/09     Add reverse stereo.
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Paul Young            12/30/09    Add RTTY.
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Paul Young            4/23/10     Add pot speeds, straight key,
 *                                   aux_tx.
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Paul Young            4/25/14     Add reset and autospace.
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Paul Young            1/31/17     Add PTT switch invert.
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Paul Young            3/23/18     Add SO2R startup configuration.
 ************************************************************************/

#ifndef PROTOCOL_H
#define PROTOCOL_H

// The USB messages are two bytes long.  The first byte specifies the
// command and the second is the value sent or received.

// These are the vendor and product IDs of the SO2R Box
#define VENDOR_ID               0x16c0;  // See http://www.voti.nl/pids
#define PRODUCT_ID              0x065e;  // 1630

#define PROTOCOL_VERSION        0x15

// The query command
#define CMD_QUERY               0x00

// The box commands
#define CMD_BOX_PATCHLEVEL      0x01 // Send current patch level
#define CMD_BOX_VERSPECIAL      0x02 // Send the version special info
#define CMD_BOX_RESET           0x03 // Reset EEPROM to defaults
```

```
#define CMD_BOX_UPDATE          0x08 // Firmware update

// These are reserved but probably not implemented in shipped code
#define CMD_BOX_DEBUG1          0x0a // Debug 1
#define CMD_BOX_DEBUG2          0x0b // Debug 2
#define CMD_BOX_DEBUG3          0x0c // Debug 3
#define CMD_BOX_DEBUG4          0x0d // Debug 4

// The keyer commands
#define CMD_KEYER_STATUS        0x10 // Idle, sending
#define CMD_KEYER_SPEED         0x11 // Set/get keyer speed
#define CMD_KEYER_CONFIG        0x12 // Set/get keyer configuration
bits
#define CMD_KEYER_CHAR          0x13 // Set next character to send
#define CMD_KEYER_OVERWRITE     0x14 // Overwrite next character to
send
#define CMD_KEYER_ABORT         0x15 // Abort sending from computer
#define CMD_KEYER_CONTROL       0x16 // Misc control functions
#define CMD_KEYER_EVENT         0x17 // Event notification
#define CMD_KEYER_DELTA         0x18 // Delta speed (queued)
#define CMD_KEYER_PTT_PRE       0x19 // PTT time before transmitting
#define CMD_KEYER_PTT_POST      0x1a // PTT time after transmitting
#define CMD_KEYER_WEIGHT        0x1b // Keyer weight
#define CMD_KEYER_POT_SPEED     0x1c // Pot speed
#define CMD_KEYER_SENDING_CHAR  0x1d // Character is being sent
#define CMD_KEYER_POT_MIN       0x1e // Keyer pot speed mininum
#define CMD_KEYER_POT_MAX       0x1f // Keyer pot speed maximum

#define CMD_KEYER_COMP1         0x50 // Radio 1 keyer compensation
#define CMD_KEYER_COMP2         0x51 // Radio 2 keyer compensation
#define CMD_KEYER_COMP3         0x52 // Radio 3 keyer compensation
#define CMD_KEYER_COMP4         0x53 // Radio 4 keyer compensation
#define CMD_KEYER_CONFIG2       0x54 // Set/get additional keyer config

typedef union keyer_config {
    struct {
        unsigned keyer_type     : 3; // Keyer type, see list below
        unsigned paddle_rev     : 1; // Paddle is reversed from
"normal"
        unsigned space_seven    : 1; // Use seven dits for word space
        unsigned timing_a       : 1; // Iambic A style, no last dit/dah
        unsigned tight          : 1; // Don't allow paddle early
        unsigned ptt            : 1; // Set PTT when sending CW
    };
    byte val;
} keyer_config_t;

#define M_KEYER_CONFIG_KEYER_TYPE   0x07
#define M_KEYER_CONFIG_PADDLE_REV   0x08
```

```
#define M_KEYER_CONFIG_SPACE_SEVEN  0x10
#define M_KEYER_CONFIG_TIMING_A     0x20
#define M_KEYER_CONFIG_TIGHT        0x40
#define M_KEYER_CONFIG_PTT          0x80

typedef union keyer_config2 {
    struct {
        unsigned keyer_autospace: 1; // Space after paddle char
        unsigned reserved_1     : 7; // Reserved
    };
    byte val;
} keyer_config2_t;

#define M_KEYER_CONFIG2_AUTOSPACE   0x01

typedef union keyer_status {
    struct {
        unsigned state          : 3; // Keyer current state
        unsigned disabled       : 1; // Keyer is disabled
        unsigned ready          : 1; // Buffer free, OK to send char
        unsigned reserved_1     : 3; // Reserved
    };
    byte val;
} keyer_status_t;

#define M_KEYER_STATUS_STATE        0x07
#define M_KEYER_STATUS_DISABLED     0x08
#define M_KEYER_STATUS_READY        0x10

// Keyer min and max speeds
#define KEYER_SPEED_MIN         2    // Minimum speed
#define KEYER_SPEED_MAX        99    // Maximum speed

// Keyer types
#define KEYER_TYPE_IAMBIC       0    // Both paddles alternate
#define KEYER_TYPE_ULTIMATIC    1    // Both paddles send last pressed
#define KEYER_TYPE_DIT          2    // Both paddles send dit
#define KEYER_TYPE_DAH          3    // Both paddles send dah
#define KEYER_TYPE_STRAIGHTKEY  4    // Straight key

#define KEYER_MAX_TYPE KEYER_TYPE_STRAIGHTKEY

// Keyer visible states
#define KEYER_VSTATE_IDLE       0    // Keyer is idle
#define KEYER_VSTATE_PADDLE     1    // Keyer is sending from paddle
#define KEYER_VSTATE_REMOTE     2    // Keyer is sending from computer
#define KEYER_VSTATE_TUNE       3    // Keyer is sending daaaaaaaaah

typedef union keyer_control {
    struct {
```

```
        unsigned tune_off       : 1; // Keyer not in tune state
        unsigned tune_on        : 1; // Keyer in tune state
        unsigned pot_off        : 1; // Pot does not control speed
        unsigned pot_on         : 1; // Pot controls keyer speed
        unsigned reserved_1     : 4; // Reserved
    };
    byte val;
} keyer_control_t;

#define M_KEYER_CONTROL_TUNE_OFF    0x01
#define M_KEYER_CONTROL_TUNE_ON     0x02
#define M_KEYER_CONTROL_POT_OFF     0x04
#define M_KEYER_CONTROL_POT_ON      0x08

// Keyer abort reasons
typedef union keyer_abort {
    struct {
        unsigned command        : 1; // Command
        unsigned paddle         : 1; // Sending on paddle
        unsigned tx_changed     : 1; // Transmitter was changed
        unsigned reserved_1     : 5; // Reserved
    };
    byte val;
} keyer_abort_t;

#define M_KEYER_ABORT_COMMAND       0x01
#define M_KEYER_ABORT_PADDLE        0x02
#define M_KEYER_ABORT_TX_CHANGED    0x04

// Keyer events
#define KEYER_EVENT_IGNORE      0
#define KEYER_EVENT_END_CHAR    1    // Keyer has sent the end of a
character
#define KEYER_EVENT_IDLE        2    // Keyer has finished sending
#define KEYER_EVENT_CLEAR       3    // Keyer buffer was cleared with
overwrite 0
#define KEYER_EVENT_PADDLE      4    // Paddle was used

// Keyer pot speed min/max
typedef union keyer_pot_speed {
    struct {
        unsigned speed          : 7; // Pot speed
        unsigned eeprom         : 1; // Write value to eeprom
    };
    byte val;
} keyer_pot_speed_t;

#define KEYER_M_POT_SPEED_SPEED     0x7f
#define KEYER_M_POT_SPEED_EEPROM    0x80
```

```
// Keyer pot speed min/max
typedef union keyer_comp {
    struct {
        signed compensation     : 7; // compensation in ms
        unsigned eeprom         : 1; // Write value to eeprom
    };
    byte val;
} keyer_comp_t;

#define KEYER_M_COMP_COMPENSATION   0x7f
#define KEYER_M_COMP_EEPROM         0x80

// The aux commands
#define CMD_AUX_PORT1           0x20
#define CMD_AUX_PORT2           0x21
#define CMD_AUX_PORT3           0x22
#define CMD_AUX_PORT4           0x23

// Aux info
typedef union aux_info {
    struct {
        unsigned aux            : 4; // Aux value
        unsigned update         : 1; // Update aux values
        unsigned reserved_1     : 3; // Reserved
    };
    byte val;
} aux_info_t;

#define M_AUX_INFO_AUX          0x0F
#define M_AUX_INFO_UPDATE       0x10

// The SO2R commands
#define CMD_SO2R_STATE          0x30
#define CMD_SO2R_CONFIG         0x31
#define CMD_SO2R_SWITCHES       0x32
#define CMD_SO2R_BLEND          0x33
#define CMD_SO2R_MAP1           0x34
#define CMD_SO2R_MAP2           0x35
#define CMD_SO2R_STARTUP        0x36

// SO2R State
typedef union so2r_state {
    struct {
        unsigned tx2            : 1; // Transmit on station 2
        unsigned rx2            : 1; // Receive on station 2
        unsigned stereo         : 1; // Stereo receive
        unsigned ptt            : 1; // Key PTT line
        unsigned stereo_reverse : 1; // Reverse stereo receive
        unsigned reserved_1     : 3; // Reserved
    };
```

```
    byte val;
} so2r_state_t;

#define M_SO2R_STATE_TX2        0x01
#define M_SO2R_STATE_RX2        0x02
#define M_SO2R_STATE_STEREO     0x04
#define M_SO2R_STATE_PTT        0x08
#define M_SO2R_STATE_REVERSE    0x10

// SO2R Configuration
typedef union so2r_config {
    struct {
        unsigned type           : 3; // Type of SO2R receive switching
        unsigned blend          : 1; // Use blend
        unsigned relays         : 1; // Set the SO2R+ relays
        unsigned aux_tx             : 1; // Set the transmitter on aux
8-11
            unsigned ptt_invert     : 1; // Invert PTT switch polarity
        unsigned reserved_1     : 1; // Reserved
    };
    byte val;
} so2r_config_t;

#define M_SO2R_CONFIG_TYPE      0x07
#define M_SO2R_CONFIG_BLEND     0x08
#define M_SO2R_CONFIG_RELAYS    0x10
#define M_SO2R_CONFIG_AUX_TX    0x20
#define M_SO2R_CONFIG_PTT_INVERT 0x40
#define M_SO2R_CONFIG_STEREO    0x80

// SO2R Configurations
#define SO2R_CONFIG_NORMAL      0x00 // Normal stereo mode
#define SO2R_CONFIG_SYMMETRIC   0x01 // Left and right always same
#define SO2R_CONFIG_SPATIAL     0x02 // Left and right keep positions

// SO2R Box switches
typedef union so2r_switches {
    struct {
        unsigned tx1            : 1; // TX switch on station 1
        unsigned tx2            : 1; // TX switch on station 2
        unsigned rx1            : 1; // RX switch on station 1
        unsigned rx2            : 1; // RX switch on station 2
        unsigned ptt            : 1; // PTT switch pressed
        unsigned reserved_1     : 3; // Reserved
    };
    byte val;
} so2r_switches_t;

#define M_SO2R_SWITCHES_TX1     0x01
#define M_SO2R_SWITCHES_TX2     0x02
```

```
#define M_SO2R_SWITCHES_RX1     0x04
#define M_SO2R_SWITCHES_RX2     0x08
#define M_SO2R_SWITCHES_PTT     0x10

// SO2R Box radio mapping
typedef union so2r_map {
    struct {
        unsigned radio          : 4; // Radio number
        unsigned eeprom         : 1; // Change EEPROM
        unsigned current        : 1; // Change Current
        unsigned reserved_1     : 2; // Reserved
    };
    byte val;
} so2r_map_t;

#define M_SO2R_MAP_RADIO        0x0f
#define M_SO2R_MAP_EEPROM       0x10
#define M_SO2R_MAP_CURRENT      0x20

typedef union so2r_map_ret {
    struct {
        unsigned current        : 4; // Current radio
        unsigned eeprom         : 4; // EEPROM radio
    };
    byte val;
} so2r_map_ret_t;

#define M_SO2R_MAP_RET_CURRENT  0x0f
#define M_SO2R_MAP_RET_EEPROM   0xf0

// SO2R Startup
typedef union so2r_startup {
    struct {
        unsigned stereo         : 1; // Auto is stereo
        unsigned reserved_1     : 7; // Reserved
    };
    byte val;
} so2r_startup_t;

// The RTTY commands
#define CMD_RTTY_CONFIG         0x40 // Enabled/disabled, inverted
#define CMD_RTTY_STATUS         0x41 // Idle, sending
#define CMD_RTTY_SPEED_BITS     0x42 // Set/get RTTY speed, length,
stop
#define CMD_RTTY_CHAR           0x43 // Set next character to send

typedef union rtty_config {
      struct {
            unsigned inverted       : 4; // RTTY inverted - per radio
```

```
            unsigned enabled        : 1; // Output is used for RTTY,
not Aux
      };
      byte val;
} rtty_config_t;

#define M_RTTY_CONFIG_INVERTED      0x0F
#define M_RTTY_CONFIG_ENABLED       0x10

typedef union rtty_status {
    struct {
        unsigned ready          : 1; // Buffer free, OK to send char
        unsigned idle           : 1; // RTTY FSK UART is idle
        unsigned reserved_1     : 6; // Reserved
    };
    byte val;
} rtty_status_t;

#define M_RTTY_STATUS_READY         0x01
#define M_RTTY_STATUS_IDLE          0x02

typedef union rtty_speed_bits {
    struct {
        unsigned speed          : 4; //
        unsigned length         : 2; // Character length plus 5
        unsigned stop           : 2; // 0=1 bit, 1=1.5 bits, 2=2 bits
      };
    byte val;
} rtty_speed_bits_t;

#define M_RTTY_SPEED_BITS_SPEED  0xf0
#define M_RTTY_SPEED_BITS_LENGTH 0x0C
#define M_RTTY_SPEED_BITS_STOP   0x03

#define RTTY_SPEED_22            0
#define RTTY_SPEED_45            1  // 45.45 Baud
#define RTTY_SPEED_50            2
#define RTTY_SPEED_56            3
#define RTTY_SPEED_75            4
#define RTTY_SPEED_100           5
#define RTTY_SPEED_110           6
#define RTTY_SPEED_150           7
#define RTTY_SPEED_200           8
#define RTTY_SPEED_300           9

#define RTTY_LENGTH_5            0  // 5 bits
#define RTTY_LENGTH_6            1  // 6 bits
#define RTTY_LENGTH_7            2  // 7 bits
#define RTTY_LENGTH_8            3  // 8 bits
```

```
#define RTTY_STOP_1              0  // 1 stop bit
#define RTTY_STOP_1_5            1  // 1.5 stop bits
#define RTTY_STOP_2              2  // 2 stop bits

// Protocol error
#define CMD_ERROR               0x7f

// Status
#define STATUS_SUCCESS          0x00
#define STATUS_BADVALUE         0x01
#define STATUS_BUSY             0x02
#define STATUS_LATE             0x03

#endif
```