Introduction

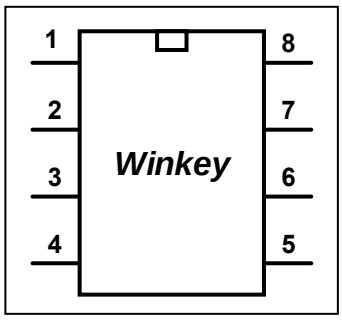
WinKey is a single chip Morse keyer IC that is designed to attach to a PC's serial port and provide accurate transmitter keying to a Windows based logging or other ham radio software package. Due to timing latency inherent in the multi-threaded Windows operating system, it is difficult to generate accurately timed Morse. WinKey buffers ASCII characters sent by a Windows based software application. It then translates them to Morse, directly keying a transmitter or transceiver. In addition, WinKey has paddle inputs so that an operator can break-in and send using paddles at any time. WinKey also provides a speed potentiometer interface so that an operator can instantly dial any speed desired.

The host PC communicates to WinKey over a simple RS232 serial interface. Letters to send along with operational commands are sent from the host to WinKey over the serial link. A substantial feature list is provided allowing the user to precisely tailor WinKey's keying characteristics to a particular transmitter. WinKey has a very low power requirement; in fact, it is designed to be powered from the PC's serial port.

A PCB board with a component kit is available for users who would like to get going with WinKey quickly and easily.

Features

- 1200 Baud Serial Rx/Tx Interface
- Iambic CW Paddle Interface
- Key Output (high true TTL)
- PTT Output: (high true TTL)
- 25 ma output sink/source
- Adjustable PTT lead in and tail delays
- Adjustable Speed 5-99 WPM
- Adjustable Weighting
- Adjustable Farnsworth Character Spacing
- Adjustable Keying Compensation
- Autospace
- Adjustable Dit/Dah Ratio
- Dit/Dah Memory Control
- Adjustable First Dit/Dah correction
- Adjustable Paddle Switchpoint
- Iambic A, B, ultimatic & "Bug" modes
- Speed Pot Interface
- Adjustable speed pot range
- Embedded commands
- 32 character input buffer
- No crystals or oscillators
- Single 5-volt operation
- Current Draw: < 2 ma
- HSCW Capability



Pin 1 – Vcc (5.0 volts)
Pin 2 – Serial Transmit TTL Output
Pin 3 – Key TTL Output
Pin 4 – Serial Receive Input
Pin 5 – PTT/Sidetone TTL Output
Pin 6 – Speed Pot Analog Input
Pin 7 – Paddle Iambic Input
Pin 8 – Vss (Ground)

Figure 1 – WinKey Package & Pinout

Theory of Operation

This section will describe how the WinKey works. As shown in Figure 1, the host PC is connected to WinKey over one of its serial COM ports (A connection through a USB hub to a serial port can also be used). WinKey is a slave to the PC in that it receives commands and data from the PC and acts upon them. The PC can send commands while WinKey is sending Morse allowing dynamic configuration changes. WinKey will communicate back to the host for four reasons:

1) lnform the host of a status change in WinKey.
2) Inform the host of a speed pot change.
3) Respond to a request for information from the host.
4) Echo back morse in ASCII as it's being sent from either the serial port or the paddles.

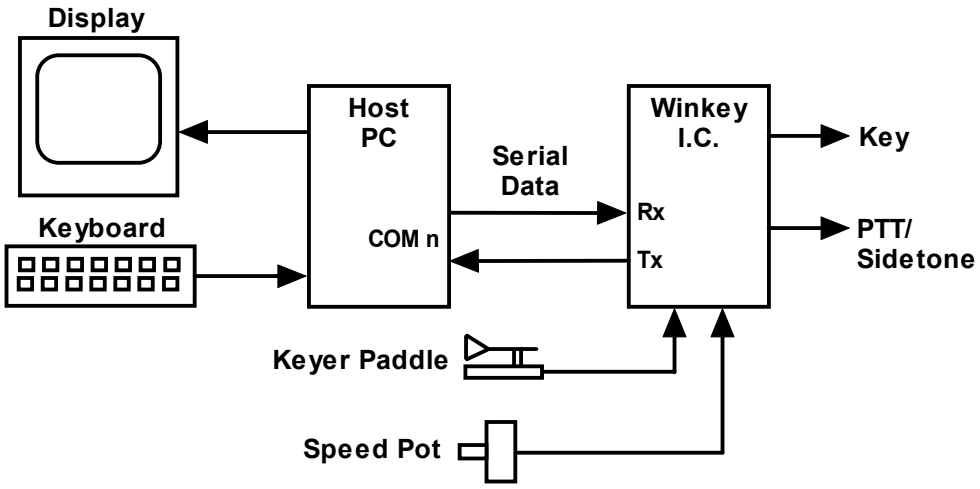


Figure 2 – WinKey to PC Connection

There are two types of serial input from the host to WinKey: Command and Data. Commands modify WinKey's operation in some way, for example changing operating speed, pausing transmission, or asking for status. Data can be letters, numbers, or prosigns that are to be sent in Morse. Commands and data are processed differently in WinKey. Data is put into a serial buffer that allows the host to send data ahead of the Morse being sent. The size of this buffer is 32 characters and is a FIFO which is an acronym for First In First Out. This means that characters are taken out in the order they were put in. Since there can be a considerable delay from host input to Morse output, commands bypass the input FIFO and are acted upon immediately. This allows changes to be made while sending is underway.

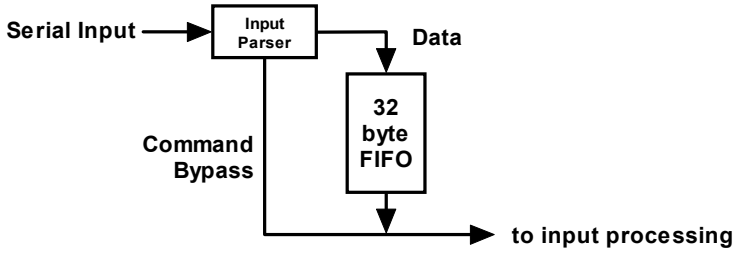


Figure 3 – Data and Command Flow inside WinKey

Since there are times when you don't want commands to take effect immediately, WinKey allows commands to be buffered. This means that the command is placed in the serial buffer and won't be acted on until it comes out of the buffer. An example of the use of a buffered command would be to send two words at two different speeds, the first at 15 WPM and the second at 20 WPM. By placing a buffered speed command between the words the speed will not be changed until the first word is completely sent. Not all, but many of the immediate commands can be entered as buffered commands.

Communication from WinKey to the host operates in a loosely coupled manner. This means that the host never issues a command and waits for a response. Instead, the host sends a request for information and WinKey queues this request and will respond to the host when processing time allows. Winkey processes tasks in parallel, there may be other bytes waiting to be sent back to the host before the latest request can be handled. Rather than wait for a return the host should divide its WinKey driver interface into two parts, one part that issues command bytes and a second part that checks for returned bytes and processes them when they arrive. Following is a bit of psuedo-code that illustrates this concept. It will make more sense as you learn about the WinKey command set.

```
while (1) {
        if (host has a command to send to WInKey) {
                send command to winkey;
        }
        else if (WinKey:uart_byte_ready) {
                wkbyte = WinKey:uart_read();
                if (( wkbyte & 0xc0) == 0xc0 {
                        it's a status byte.  (Host may or may not have asked for it.)
                        process status change
                }
                else if ((wkbyte & 0xc0) == 0x80) {
                        it's a speed pot byte (Host may or may not have asked for it.)
                        process speed pot change
                }
                else {
                        it must be an echo back byte
                        if (break-in==1) { it's a paddle echo }
                        else { it's a serial echo }
                }
        }
}
```

Notice that unless WinKey has something for the host to read the host continues to process outgoing commands and other tasks. Also note that speed pot and status bytes can be unsolicited, in other words WinKey can send these at any time a state change occurs inside Winkey. Echo back bytes are also unsolicited as they are based on asynchronous Morse sending. The host has to be able to handle these as they occur. If host processing is slow, a serial input buffer on the host side is required to make sure no returned bytes are missed.

Power Up Default State

On power up WinKey comes up in a state determined by the last Load Defaults command and will operate from paddle input and speed pot control. When the host takes over it should download a block of initialization parameters (see Load Defaults command) to set the operating state as desired and to sync WinKey with the host.

Paddle Input Priority

One of the PIC's analog to digital converters is used to read the paddle and a second analog to digital converter is used to read the speed pot. WinKey checks the speed pot at a rate of 10 times per second while the paddles are sampled 2000 times a second.

WinKey accepts input from either the serial port or iambic paddle. The paddle will always take priority and will interrupt serial data and will automatically clear WinKey's serial input buffer. When a paddle break-in occurs any additional serial data that arrives from the host will be processed, but will be ignored unless it is an immediate command. After paddling ceases, WinKey will pause for one word space time before it allows incoming serial data to be sent.

Command Descriptions

Commands are special hex codes that are sent to WinKey. These codes range from 0x01 through 0x1F. In this document a hex value will be presented in angle brackets, for example **<02>**. Some commands have one or more parameters sent immediately after the command code is sent, this will be documented as the command code followed by a value: **<02><nn>** where nn is a single byte binary value. The notation **[c]** represents a single ASCII character sent to WinKey as a single serial byte.

Immediate Commands

These commands are processed as soon as they are received, they bypass the input buffer.

● Admin <00><nn> nn is a value from 0 to 8

After power-up the serial interface will be closed, no serial status, echo, or pot change data will be sent to the host. Admin commands can be issued while the serial interface is closed and WK will return command result bytes according to the command issued. Admin commands are received, processed and any return will be sent back immediately. Admin commands calibrate the interface, reset WK, obtain debug information and open the interface. With the exception of the Amin:Close command, all Admin commands should only be issued while the serial interface is closed. As soon as the serial interface is opened it will be impossible to sort out Admin responses from unsolicited WK status response. Following are descriptions of all the Admin commands:

0: Calibrate WinKey's timebase is derived from an internal RC oscillator. Even though the timer is trimmed to within 10% accuracy when programmed, it is still susceptible to additional error due to variation in temperature. This command will trim the oscillator to within 1% of the serial timing of the host. Follow this sequence when issuing the calibrate command: <00><00> pause 100 mSec <FF>

1: Reset Resets the WinKey processor to the power up state.

2: Host Open Upon power-up WinKey initializes with the host serial port turned off. To enable the port the host must issue the admin:open command. Upon open WinKey will respond by sending the revision code back to the host. The host must wait for this return code before any other commands or data can be sent to WinKey.

3: Host Close Use this command to turn off the host interface. WinKey will still respond to paddle input, speed pot changes, and admin commands even though the host port is closed. The settings that were in force before the Host port was closed will remain in force.

4: Echo Test Used to test the serial interface. The character sent to WinKey after this command will be echoed back to the host.

5: Paddle A2D Returns the real time value of the paddle A2D. This diagnostic tool is provided to help debug problems with the paddle analog interface. The paddles are read through one of WinKey's built in analog to digital converters. The paddle position is determined by reading the voltage on a resistor divider circuit and comparing the value to pre-determined thresholds. To allow for tolerance in resistor values and connection paths there is a range of values that will meet each of the four possibilities. The following table gives the ranges for each:

Paddle State	Keyer Pin 7 Voltage	A/D Output
Both paddles up	5 - 3 volts	> 152
Dit paddle down	2.9 – 2.0 volts	151 – 104
Dah paddle down	1.9 – 1.4 volts	103 – 71
Both paddles down	1.39 – 0 volts	70 – 00

Figure 4 – Paddle Analog Voltage Range

6: Speed A2D A value ranging from 00 to 63 is returned indicating the raw rotational position of the speed pot, in other words the returned value is not affected by the speed pot setup command settings.

7: Get Values Returns all of the internal setup parameters. They are sent back in the same order as issued by the Load Defaults command. Again, this command is a diagnostic aid. Only issue this command when host interface is closed.

8: Reserved ***K1EL Debug use only***

9: Get Cal Returns the internal calibration result byte.

● Sidetone Frequency <01><nn> nn is a value from 1 to 10

Normally WinKey's pin 5 is the PTT output. Sacrificing PTT functionality, pin 5 can be configured to output a square wave sidetone instead by using the ***Set Pin 5 Mode*** command. When sidetone is enabled, pin 5 functions as a sidetone square wave output. The PTT delays still work as described by the ***Set PTT Lead/Tail*** Command, but a PTT output is not available. The following table specifies the sidetone frequencies that are available:

nn	Frequency		nn	Frequency
1	3759 Hz		6	625 Hz
2	1879 Hz		7	535 Hz
3	1252 Hz		8	469 Hz
4	940 Hz		9	417 Hz
5	752 Hz		10	375 Hz

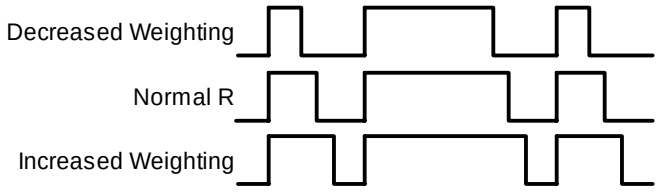
Figure 5 – Sidetone Selection Table

● Set WPM Speed <02><nn> nn is in the range of 5-99 WPM
Example: <02><12> set 18 WPM

Set a new Morse operating speed, this command takes effect as soon as WinKey receives it. If speed is set to zero then WinKey will take it's speed setting directly from the speed pot., this is the reset default.

● Set Weighting <03><nn> nn is in the range of 10-90%
Example: <03><32> for weight=50

This command allows a proportional amount to be either added or subtracted from the length of all dits and dahs sent. A value of 50 (0x32) selects no weighting adjustment. Values less than 50 reduce weighting and values greater than 50 increase weighting. Note that weighting does not affect sending speed because any increase in keyed time is subtracted from spacing time. Reduction in weighting results in a thinner sounding keying while increased weighting results in a heavier sound. Since weighting tracks speed, a given weighting will sound the same at all speeds.



WEIGHTED DIT TIME =
(((WEIGHT×50)/50)×DIT TIME)

WEIGHTED DAH TIME =
(((WEIGHT×50)/50)×DAH TIME)

Figure 6 - Weighting Example

● Set PTT Lead/Tail <04><nn1><nn2> nn1 sets lead in time, nn2 sets tail time
both values range 0 to 250 in 10 mSecs steps
Example:
<04><01><A0> lead-in = 10 mSecs, tail = 1.6 sec

WinKey provides a transmitter PTT output that can be used to switch a transmitter or linear amplifier over to transmit mode in advance of actual CW keying. You have control over the time delay between when PTT is asserted and when CW keying will start, this is lead-in. You also have control over how long the transmitter will stay in transmit after keying has stopped; this is the tail delay. Note that the PTT pin 5 can be reconfigured as a sidetone output, see Sidetone Frequency command.

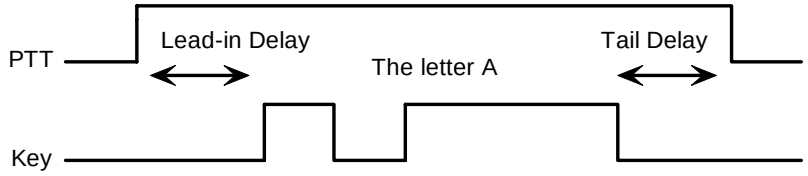


Figure 7 – PTT Lead-in and Tail Example

● Setup Speed Pot <05><nn1><nn2><nn3>
nn1 = MINWPM, nn2 = WPMRANGE, nn3 = POTRANGE

This command sets the limits for the speed pot. MINWPM sets the lowest value returned; WPMRANGE indirectly specifies the maximum value returned. For example if MINWPM=10 and WPMRANGE=15, the full pot swing values, min to max, would be 10 to 25 WPM. Note that the max value is MINWPM+WPMRANGE. POTRANGE specifies the maximum value read from the pot. It is determined by the pot configuration as illustrated below. The configuration on the left is the preferred setup as it gives the best linearity across the full rotation of the speed pot. The only advantage to the right setup is that only two wires need to be connected to the pot.

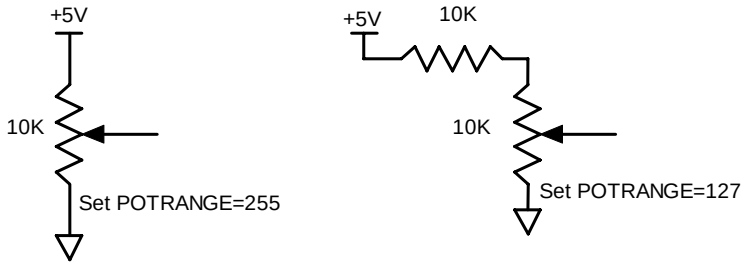


Figure 8 – Speed Pot Configurations vs. POTRANGE

● Set Pause State <06><nn> nn = 01 pause, value = 00 unpause

Sending will stop immediately when WinKey is paused and will not resume until an unpause state is set. The current character being sent in Morse will be completed before pause will commence. Note that the Clear Buffer command will return WinKey back to an unpaused state.

● Get Speed Pot <07> no parameter
Request to WinKey to return current speed pot setting.

This command will cause a speed pot command request to be queued in WinKey and it will be acted on as soon as possible. Depending on current processing load the pot status byte will be sent no longer than 200 milliseconds after command receipt. The application should not wait for a response but process the returned

data in an unsolicited status handler. The returned value will range from 0 to 31 and is governed by the setting of the MINWPM and WPMRANGE values set via the POTSET command. The returned value will be the actual speed pot value minus the MIN_WPM setting. This allows the speed pot to be windowed into any 32 step range from 5 to 99 WPM. The two MSBs of a Speed Pot status byte will always be 10:

1	0	6 bit value in WPM

Figure 9 - Speed Pot Status Byte Format

● **Backspace** **<08>** **no parameters**
Backup the input buffer pointer by one character. This command is only meaningful if there is something in the serial input buffer, otherwise it is ignored.

● **Set PinConfig** **<09><nn>** **nn determines how output pins are mapped**
nn=05: Pin 5 = PTT (reset default)
nn=06: Pin 5 = Sidetone
nn=04: Pin 5 = Deasserted, Pin 3 = Key Output
nn=08: Pin 5 = Key Output, Pin 3 = Deasserted

See Below	Pin5 KeyOut Enable	Pin3 KeyOut Enable	Pin 5 Sidetone Enable	Pin5 PTT Enable

Figure 10a - PINCFG Byte Format

In version 9 the PINCFG register is overloaded with two new features, Dit/Dah priority and Paddle hang time. These settings are allocated to the upper four bits as follows:

Priority 1	Priority 0	Hang Time 1	Hang Time 0	See Above

Figure 10b – Priority/Hang Time Format

Priority = 00 Normal Ultimatic
Priority = 01 Will send dahs when both paddles are pressed in Ultimatic mode
Priority = 10 Will send dits when both paddles are pressed in Ultimatic mode
Priority = 11 Undefined

HangTime = 00 Wait 1.0 wordspace before ending paddle insertion
HangTime = 01 Wait 1.33 wordspace before ending paddle insertion
HangTime = 10 Wait 1.66 wordspace before ending paddle insertion
HangTime = 11 Wait 2.0 wordspace before ending paddle insertion

Hang Time works like tail time in that it holds PTT on between paddle presses, but it is different in that the wait time is proportional to sending speed by virtue of the fact that it is measured in word space time while tail time is measured in absolute milliseconds.

● **Clear Buffer** **<0A>** **no parameters**
This command will reset the input buffer pointers to an empty state. It is a general clear also in that Tune and Pause are also cancelled by this command.

Clear Buffer can be sent at any time to abort a message, abort a command, or to clear the serial buffer. It will cancel any Morse character in progress immediately ending it in midstream if necessary.

● **Key Immediate** **<0B><nn>** **nn = 01 keydown, n = 00 keyup**

Use this command to implement a tune function. Once asserted, key down will remain in effect until either a key immediate with a zero value is received or the internal tune watchdog timer expires. The tune timer is hard coded

to a value of 100 seconds and cannot be disabled. The key down can be aborted either by the paddles or by a clear buffer command.

● Set HSCW <0C><nn> nn = the lpm rate divided by 100

WinKey supports HSCW (High Speed CW) transmit rates through the use of this immediate command. .

For example nn=20 selects 2000 lpm and nn=35 selects 3500 lpm. Any rate from 1000 to 8000 can be picked although only a handful are actually used by radio amateurs. In the US, common rates are 1000, 2000, 4000 and 6000 lpm while in Europe 1000, 1500, 3000, 4000 lpm are common.

● Set Farns WPM <0D><nn> nn is in the range of 10-99
Example: <0D><12> for Farnsworth=18 WPM

Farnsworth spacing is useful for CW practice because it encourages you to learn characters by sound not individual dits and dahs. In WinKey, Farnsworth is implemented by sending letters at a fixed rate of **nn** WPM regardless what the WPM sending rate is. Spacing between characters is determined by the sending rate. When the WPM rate is set above the Farnsworth WPM, Farnsworth is automatically disabled.

● Set WinKey Mode <0E><nn> nn = Mode bit field in binary
Example: <0E><13> set bits 4,1,0, clear the rest

The operational mode of WinKey can be modified by directly altering its internal mode register. This register is made up of eight bits which each control a particular mode.

Mode Bit	Function
7 (MSB)	Disable Paddle watchdog
6	Paddle Echoback (1=Enabled, 0=Disabled)
5	Key Mode: 00 = Iambic B 01 = Iambic A
4	10 = Ultimatic 11 = Bug Mode
3	Paddle Swap (1=Swap, 0=Normal)
2	Serial Echoback (1=Enabled, 0=Disabled)
1	Autospace (1=Enabled, 0=Disabled)
0 (LSB)	CT Spacing when=1, Normal Wordspace when=0

Figure 11 – Winkey Mode Selection Table
The Winkey mode register is cleared at reset.

Bit 7
Winkey has a paddle watchdog counter that will disable the key output after 128 consecutive dits or dahs. This is to guard against the paddles being accidentally keyed continuously. By default the paddle watchdog is on but it can be turned off by setting this mode bit.

Bit 6
When this bit is set to one all characters entered on the paddles will be echoed back to the host. From the host perspective paddle echo and serial echo are the same, in either case the letter sent in Morse by WinKey is echoed back to the host. The echo occurs after the letter has been completely sent. The host can determine the source by the sense of the “break-in” status bit. If the bit is high when the echoed letter comes in then the letter’s source was from the paddles, if break-in is low the source if from the serial port.

Bit 5,4
WinKey supports Iambic A, B, Ultimatic, and Bug keying modes. In iambic mode WinKey makes both dits and dahs automatically based on which paddle you press. In bug mode WinKey makes the dits and you make the dahs. You also can use bug mode to operate in straight key mode or if you want to key through WinKey with a different keyer, simply set bug mode and use the dah input to key WinKey.

In either iambic mode, alternating dits and dahs are sent while both paddles are held closed. In mode B an extra alternate dit or dah is sent after both paddles are released. In Ultimatic mode when both paddles are pressed the keyer will send a continuous stream of whichever paddle was last pressed.

Bit 3
Paddle swap: this is a nice feature to have when right and left handed ops want to share the same keyer.

Bit 2
Echo back is a feature that is included to allow a host application to stay exactly in sync with Morse letters sent. When this mode is enabled all data taken out of the serial buffer is sent to the host after it has been sent in Morse. This allows the host to reconcile differences in timing introduced by WinKey’s internal 32 byte serial buffer. Note that only letters, and not buffered commands with their parameters or wordspaces, are echoed back to the host.

Bit 1
Here is how autospace works: If you pause for more than one dit time between a dit or dah WinKey will interpret this as a letter-space and will not send the next dit or dah until full letter-space time has been met. The normal letter-space is 3 dit spaces. WinKey has a paddle event memory so that you can enter dits or dahs during the inter-letter space and WinKey will send them as they were entered. With a little practice, autospace will help you to send near perfect Morse.

Bit 0
Winkey supports contest spacing which reduces the wordspace time by one dit. Instead of 7 dits per wordspace, Contest spacing selects six dits per wordspace.

● Load Defaults <0F><value list> value list is a set of 15 binary values

This command is provided to allow all the operating parameters to be loaded into WinKey in one block transfer. The values are binary and must be loaded in order. The values are exactly the same as those loaded for the individual commands. The preferred time to issue this command is at reset just after the interface has been opened. Issuing this command while sending Morse is not advised.

1) Mode Register	2) Speed in WPM	3) Sidetone Frequency
4) Weight	5) Lead-In Time	6) Tail Time
7) MinWPM	8) WPM Range	9) 1st Extension
10) Key Compensation	11) Farnsworth WPM	12) Paddle Setpoint
13) Dit/Dah Ratio	14) Pin Configuration	15) Pot Range

Figure 12 - Default Value List in order of issuance:

● Set 1st Extension <10><nn> nn is in the range of (0 to 250) × 1 mSecs
Example: <04><80> sets lead in to 80 mSecs

WinKey addresses a problem often encountered when keying older transceivers that have a slow break-in response. Due to a slow receive to transmit changeover time, the first dit or dah of a letter sequence can be chopped and reduced in length. Adding a fixed amount to the first element of a sequence can compensate for this. For example, an R would be sent with the first dit elongated but the subsequent dah-dit sent normally. The compensation amount is transceiver dependent and is generally independent of sending speed. Note though that this is usually only a noticeable problem at higher CW speeds >25 WPM.

A challenge in this scheme is to determine when sending has stopped long enough to cause the transceiver to switch back to receive. If it has it'll require a new first element correction on the next sequence. WinKey uses the PTT tail timer to determine this, set the tail timer to roughly match the transmit to receive changeover time of the transceiver and things will work fine. It takes some trial and error to get it set up right so make sure you preserve the value and load it as a defaults after reset.

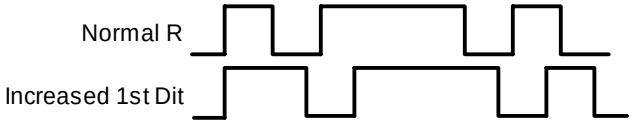


Figure 13 – 1st Extension Example

● Set Key Comp <11><nn> nn is in the range of (0 to 250) × 1 mSecs
Example: <11><B4> sets key comp to 180 mSecs

Keying Compensation allows a fixed amount to be added to the length of all dits and dahs. QSK keying on modern transceivers can cause shortening of the dit and dah elements which is especially noticeable at high speeds. WinKey allows the length of the dit and dah elements to be increased uniformly to compensate for this. The adjustments are made in units of one-millisecond steps. The maximum adjustment is 250 mSecs. Key compensation is very similar to Weighting in that any adjustment added to the dits and dahs is subtracted from the spacing so the speed is not changed. The difference between weighting and compensation is that compensation is independent of speed, so if 10 msec of key compensation is selected 10 msec will be always be added regardless of speed. So be careful at high speeds and large values of key compensation, you may end up with no inter-element space.

When nn = 00 there is no adjustment while nn=12 will add twelve mSecs.

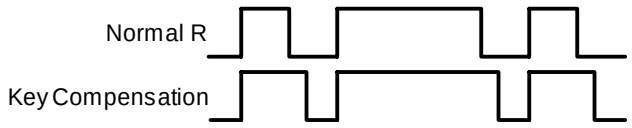


Figure 14 – Keying Compensation Example

● Null Command <13> this command is a NOP

● Set Paddle Switchpoint <12><nn> nn is in the range of 10-90%
Example: <03><37> for sensitivity=55

This controls when WinKey will start looking for a new paddle press after sensing the current one. If there is not enough delay the keyer will send unwanted dits or dahs, if there is too much delay it bogs you down because you can't get ahead of the keyer. The default value is one dit time (50) and is adjustable in percent of a dit time. Faster operators report a setting somewhat less than default is more pleasing. If the paddle sensitivity is set to zero, dit and dah paddle memory is disabled. The delay is calculated with this formula:

DELAY_TIME = (SWITCHPOINT×DIT_TIME)/50 where Switchpoint is a value between 10 and 90.

● Software Paddle <14><nn> nn = 00 paddle up, n=01 dit, n=02 dah, n=03 both

This command provides a way to assert paddle inputs from the host. The PC application would convert keydown codes to Software Paddle commands. Due to the limited response time of the keyboard, operating system, and serial communication the best you can do is around 20 WPM. The paddle watchdog will be used for this interface as if it were a normal paddle input.

● Request WinKey Status <15> no parameter, Return WinKey's status byte

This command is used to queue a request to WinKey to send its current operating state. The status byte returned consists of a bit field that is defined by the following table. The three MSBs of the the status byte are always 110.

Status Bit	Name	Definition
7 (MSB)	Tag	1
6	Tag	1
5	Tag	0
4	WAIT	WK is waiting for an internally timed event to finish
3	KEYDOWN	Keydown status (Tune) 1 = keydown
2	BUSY	Keyer is busy sending Morse when = 1
1	BREAKIN	Paddle break-in active when = 1
0 (LSB)	XOFF	Buffer is more than 2/3 full when = 1

Figure 15 – WinKey Status Definitions

● Pointer Cmd <16><nn> Input Buffer Command Set

This command allows the host app to manipulate the input buffer for special situations such as "on the fly" callsign correction. Four commands make up the pointer command set:

nn=00 Reset input buffer pointers to start of buffer, only issue this when buffer is empty.
nn=01 Move input pointer to new position in overwrite mode
nn=02 Move input pointer to new position in append mode
nn=03 Add multiple nulls to the buffer <16><03><number of nulls>

A detailed description of the pointer command will be detailed in a separate application note.

● Set Dit/Dah Ratio <17><nn> nn is in the range of 33-66
Example: <03><42> for ratio=1:4

Allows WinKey to deviate from the standard 1:3 ratio of dit/dah. The formula to determine dah/dit ratio is:

DAH/DIT = 3*(nn/50)

A value of 50 selects 1:3, a value of 33 would select 1:2, and a value of 66 would select 1:4. This causes an intentional distortion of the Morse waveform. Some ops use this option to make their CW sound less "machine like". *A little goes a long way!!*

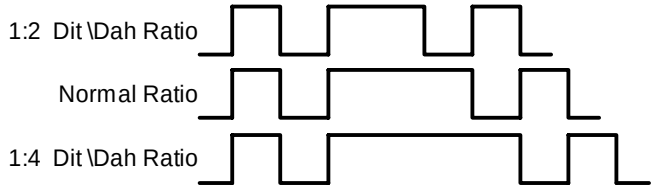


Figure 16 – Three ratio settings for the letter R

Buffered Commands

These commands go into the input buffer maintaining their positional relationship to data.

● PTT On/Off <18><nn> nn = 01 PTT on, n = 00 PTT off

This command allows the PTT output to be used for a custom purpose. The command is operational only when sidetone and PTT are diabled (See PINCFG command) PTT can be turned on or off at will and will be unaffected by all other commands including Clear Buffer. Typical applications could be as a power level control, antenna selector, or to turn on a cooling fan. Since this is a buffered command, the on/off will happen at the command's position in the buffer and remain in effect until the next PTT ON/OFF command is encountered. This command will not stall the output buffer.

● Key Buffered <19><nn> nn = 0 to 99 seconds

Use this command to assert the key output for a specific period of time. Since this is a buffered command, the keydown will begin at the command's position in the buffer and will stall the buffer until the timeout has been satisfied. The keydown can be aborted either by the paddles or by a Clear Buffer command. The maximum allowable key down time is 99 seconds.

● Wait for nn Seconds <1A><nn> nn = 0 to 99 seconds

This command is used to insert a fixed pause into a message. Since this is a buffered command, the pause will begin at the command's position in the buffer and will stall the buffer until the timeout has been satisfied.

● Merge Letters <1B>[C][C] Merge Two Letters into a Prosign

You can build "on the fly" prosigns with this command. Issue the command followed by two letters or numbers and they will be merged together: <1B>[A][R] is sent as **AR.** Note that nothing will be sent until both letters have been received.

Several common prosigns such as AR, SK, BT, And DN are already assigned (see page 13) so you don't have to build these. One application of this feature is to send special European language characters.

● Change Speed Buffered <1C><nn> nn is in the range of 5-99 WPM
Example: <02><23> set 35 WPM

This command places a speed change command into the serial buffer that will be acted upon when it is taken out of the buffer. The current speed in force will be stored and will be reinstated when the buffered speed change is cancelled by a Cancel Speed Change command or any of the following: Unbuffered Speed change, Weight change, Farnsworth change, Ratio change, Compensation change, or Mode change.

This command is useful for building messages with embedded speed changes. In this example the first part of the message will be sent at 5 WPM, the second at 25 WPM and the end at whatever the current speed is:

```
<1C><05>VVV DE K1EL <1C><19> VVV DE K1EL<1E><END DE K1EL>
```

● HSCW Speed Change <1D><nn> nn = (lpm/100)

This command acts the same as the immediate HSCW command. This allows you to insert an HSCW burst in a regular CW message or to put HSCW bursts of two different rates into the same message.

● Cancel Buffered Speed Change <1E>

This command will cancel any buffered speed change command that is in force. The sending speed that was in force before any buffered speed change was encountered will be restored. Several buffered speed changes can be issued within a message but none will alter the original sending speed.

● Buffered NOP <1F>

This command will occupy a position in the input buffer but will result in no action when it is processed.

Unsolicted Status Transmission

WinKey will send two types of unsolicited status to the host: speed pot change, and a change in status byte. Whenever the speed pot is moved its new value will be sent to the host. Likewise whenever there is a change to the internal status register inside WinKey, a copy of it will be sent to the host.

The status byte will be in the same format as previously described in the Get Status command. Likewise the speed pot status will be as described in the Get Pot command.

Since these bytes can arrive at any time and potentially can be mixed with echo back bytes, they have identifying tags. If the MSB is set that identifies the byte as spontaneous, bit 6 then identifies either a speed pot byte or a status byte. Echo back bytes will always have the MSB=0.

The host can force either of these bytes to be returned by using their respective Get Pot or Get Status commands. Due to the parallel task handling nature of Winkey a response may not be immediate, there may be a other bytes in the return queue that need to be sent to the host first. Worst case latency will be 200 milliseconds. It is not advisable for the host to wait for a response, it is better to handle it as illustrated by the code fragment shown in an earlier section of this document.

Prosign Key Assignments

Winkey has mapped several unused character codes to standard prosigns. Table 5 shows the mappings. Any additional prosigns can easily be generated using the merge character command.

ASCII	Hex		Prosign		ASCII	Hex		Prosign
“	0x22	*Is mapped to*	RR		+	0x2B	*Is mapped to*	AR
#	0x23	*Is mapped to*	EE (null)		-	0x2D	*Is mapped to*	DU
$	0x24	*Is mapped to*	SX		/	0x2F	*Is mapped to*	DN
%	0x25	*Is mapped to*	EE (null)		:	0x3A	*Is mapped to*	KN
&	0x26	*Is mapped to*	EE (null)		;	0x3B	*Is mapped to*	AA
‘	0x27	*Is mapped to*	WG		<	0x3C	*Is mapped to*	AR
(	0x28	*Is mapped to*	KN		=	0x3D	*Is mapped to*	BT
)	0x29	*Is mapped to*	KK		>	0x3E	*Is mapped to*	SK
*	0x2A	*Is mapped to*	EE (null)		@	0x40	*Is mapped to*	AC

Figure 17 – Prosign/Abbreviations Assignments

Serial Baud Rate

Winkey’s baud rate is fixed at 1200 baud. The communications setup should be set to eight bit data, 1.5 or 2 stop bits, and no parity.

Gap Insertion

Winkey version 9 interprets the | character (hex 0x7C) a ½ dit delay time. The | character can be included in a text string to add extra emphasis to similar sounding sequences. An example is W1OMO, sending it as W1|O|M|O makes it easier to copy.

Index

Change History:

12.31.2002	ste	Added CT spacing Sidetone now disabled when value set to zero Added Cancel Buffered Speed command
2.28.2003	ste	Support for WK ver 4: Mode register key modes consolidated into two bits Removed “get revision” command, it’s now returned by open command Added Pin Config command Added third parameter (POTRANGE) to “setup speed pot “ command Add paddle watchdog control bit to mode register Add buffer pointer commands Add two new bytes to Load Defaults, Pot Range & Pin Config Moved echo command to ADMIN commands Changed RETCAL admin command code to 9 Changed Buffered Key Down to a timed keydown Changed Buffered PTT control to be general purpose output Added Buffered NOP command WK PCB V2 allows external control of key line
4-01.2003	ste	Corrected PINCFG settings, Keying mode settings
5.05.2003	ste	Typo in Load Defaults command item 8, 14, and 15 U2:Vcc and GND were swapped in schematic
10.18.2004	ste	Added version 9 features
04.02.2005	ste	WK version 10
05.16.2005	ste	Split document into Specification (this doc) and WK PCB assembly manual

Command Table

Command Name	Code	Type	Description	Syntax	Pg
Admin	00	Imm	Administative Commands	<00><type>…	4
Sidetone Freq	01	Imm	Set sidetone frequency	<01><freq>	5
Speed	02	Imm	Set Morse sending speed	<02><WPM>	5
Weighting	03	Imm	Set key weighting	<03><weight>	5
PTT Lead-in/Tail	04	Imm	Set up PTT delays	<04><leadin><tail>	6
Speed Pot Setup	05	Imm	Set up speed pot range	<05><m><wr><pr>	6
Pause	06	Imm	Pause Morse output	<06><0 or 1>	6
Get Speed Pot	07	Imm	Request speed pot value	<07>	7
Backspace	08	Imm	Backup input pointer	<08>	7
Pin Configuration	09	Imm	Set output pin configuration	<09><config>	7
Clear Buffer	0A	Imm	Clear input buffer	<0A>	7
Key Immediate	0B	Imm	Direct control of key output	<0B><0 or 1>	7
HSCW Speed	0C	Imm	Set HSCW speed	<0C><lpm/100>	7
Farnsworth	0D	Imm	Set Farnsworth speed	<0D><WPM>	7
WinKey Mode	0E	Imm	Load Winkey mode byte	<0E><mode>	8
Load Defaults	0F	Imm	Download WK state block	<0F><…15 values…>	9
First Extension	10	Imm	Setup 1st element correction	<10><msec>	9
Key Compensation	11	Imm	Set Keying Compensation	<11><comp>	9
Paddle Switchpoint	12	Imm	Setup paddle sensitivity	<12><sens>	10
Null	13	Imm	Null Command, NOP	<13>	10
S/W Paddle Input	14	Imm	Software Paddle Control	<14><paddle select>	10
Winkey Status	15	Imm	Request Winkey status	<15>	10
Buffer Pointer	16	Imm	Buffer pointer commands	<16><cmd>…	10
Dit/Dah Ratio	17	Imm	Set ratio of dit/dah	<17><ratio>	11
PTT Control	18	Buff	Turn PTT on/off	<18><0 or 1>	11
Timed Key Down	19	Buff	Turn KeyOut on for an interval	<19><secs>	11
Wait	1A	Buff	Wait for N seconds	<1A><secs>	11
Merge Letters	1B	Buff	Merge chars into prosign	<1B>[c][c]	11
Speed Change	1C	Buff	Change Morse speed	<1C><WPM>	12
HSCW Speed	1D	Buff	Set HSCW speed	<1D><lpm/100>	12
Cancel Buff Speed	1E	Buff	Cancel Buff Speed Change	<1E>	12
Buffered NOP	1F	Buff	Null Command (buffered)	<1F>	12

WinKey is fully guaranteed and if you are not satisfied please return the chip or kit for a full refund.

Please post questions on the K1EL Message Board:

http://groups.yahoo.com/group/k1el_keyers/

You can contact K1EL directly at:

Steven T. Elliott K1EL
43 Meadowcrest Drive
Bedford, NH 03110 USA

e-mail: K1EL@k1el.com

website: www.k1el.com