

Official Bulletin



MHz to GHz

The West Australian VHF Group Bulletin

JANUARY 2002

THE WEST AUSTRALIAN VHF GROUP (INC)
PO BOX 189 APPECROSS

e-mail for editor to: pi@multiline.com.au

President	Alan	VK6ZWZ	Acting Sec.	Don	VK6HK
Vice President	Terry	VK6ZLT	Treasurer	Ces	VK6AO
Activities			Materials		
Publicity					
Librarian	Al	VK6ZAY	Museum Rep	Tom	VK6ZAF
Bulletin Editor	Luigi	VK6YEH			
Councillor	Wally	VK6KZ	Councillor	Terry	VK6TRG

NOV	12	VHF MICROWAVE NET	JAN	14	VHF MICROWAVE NET
	19	COMMITTEE MEETING		21	COMMITTEE MEETING
	26	GENERAL MEETING		28	GENERAL MEETING
FEB	11	VHF MICROWAVE NET	MAR	11	VHF MICROWAVE NET
	18	COMMITTEE MEETING		18	COMMITTEE MEETING
	25	GENERAL MEETING		25	GENERAL MEETING
APR	8	VHF MICROWAVE NET	MAY	13	VHF MICROWAVE NET
	15	COMMITTEE MEETING		20	COMMITTEE MEETING
	22	ANNUAL GEN. MEETING		27	GENERAL MEETING

Editors Note:

A new year has dawned and I trust you have all had a good Christmas and New Year. Several calls were received over the holiday season with some good activity on 6m. For those of you who missed it, Novembers meeting had a very interesting presentation by Al (VK6ZWZ). This consisted of a dish antenna made from a golfing umbrella capable of receiving 1269MHz and 2.4GHz. We will be looking at a future technical article on this.

The technical article dominates this months bulletin so introductions need to be short. 73's until the next bulletin.

Technical Article:

Remote Keyer with Fault Detection Software

Continuing on our remote keyer project, the following is a listing of the software used to control the keyer. To assist those who may not have had experience in programming a PIC processor a description of how the software operates is included. The software assumes a 4MHz oscillator was used. As you can see it has been configured to operate as a remote keyer for either a beacon or repeater application. This allows for some flexibility in its use. As with any micro-controller project, there is a limit as to the amount of flexibility verses code. It is believed that this code allows sufficient flexibility without adding code, which may prove redundant in many applications.

Before discussing the code itself we will look at the predefined parameters to see where changes may be made.

BEACON_OP	If this is define the software acts in beacon mode ie. the transmitter is assumed to be continuously operating and a tone is generated for a time interval. The identifier is sent followed by a character depicting a fault if one exists. If this definition is commented out the software operates in repeater mode ie. There is silence until the predetermined time has expired. The transmitter is turned on by setting port A bit 1 HIGH and the identifier sent. Again, if a fault is detected the fault character is sent before the transmitter is keyed down.
DOT	Defines the length of the dot tone symbol in milliseconds.
DASH	Defines the length of the dash tone symbol in milliseconds.
BIT_DELAY	Defines the length of time between a dot or dash tone. This number must be different to the dot or dash delay.
CHAR_DELAY	Defines the length of time between a morse character. This number must be different to the dot or dash delay.
STATE0/1	Defines operational states and should not be changed.
TX_INTVL_LOW	Defines the low byte of the transmission interval in milliseconds.
TX_INTVL_HIGH	Defines the high byte of the transmission interval in milliseconds. A maximum interval of 65.535 seconds may be currently set.
TIMER_DLY	Defines the preset internal timer value. This coupled with the internal hardware divider gives the 1ms increments on which we have based the time.
CHAN_x_BIT	Defines the fault channel port bits.
TONE_BIT	Defines the tone output port bit.
TX_EN_BIT	Defines the transmitter enable output port bit.
MORSE_x	Defines the morse alphabet and numerals. Note the each character has a inter-character delay value appended to it.

The software uses the following variables.

opstate	Contains the current operational state
time_low	Contains the low byte of the interval timer countdown value.
time_high	Contains the high byte of the interval timer countdown value. These values can be expanded to more bytes to make the transmission interval longer.
table_ptr	Is a counter used when stepping through the morse strings.
morse_data	Temporarily stores a morse delay value.

morse_dly	Loaded with the delay required for a tone or space interval.
W_TEMP	Temporary storage for W register.
STAT_TEMP	Temporary storage for status register.
tone_flag	When this is set to ON the tone output is toggling.
count	General counter.

The messages transmitted consists of a string of morse characters terminated by a '0'. The station identifier is set first. Simply change the definitions for the identifier required. The remainder of the tables are status characters assigned to a fault channel. These status characters may be expanded to be short messages.

Eg.

```
curr_stat_a      addwf  PCL,F
                  dt      MORSE_S,MORSE,W,MORSE_R,WORD_DELAY,0
```

This can be used to transmit SWR for an SWR fault.

The watchdog timer is enabled in case a software fault occurs. Ensure the watchdog is reset at regular intervals to ensure the software does not reset.

The timer and I/O ports are initialized before the main routines are executed. When in state 0 the software is waiting for the timer interval to expire. This is checked inside the interrupt routine. Once expired the state is set to 1. With the state set to 1 the station identifier followed by the fault status, if applicable, is sent.

The timer interrupt routine performs the following functions. The interval timer is decremented. When this reaches 0 it changes the state to 1.

When in state 1 it checks the delay register to see if a delay was set. The delay is consequently decremented.

The tone flag is then tested to see if the tone bit needs to be toggled.

```

;-----
;   Name:  radiomon.asm
;   Version: 1.00
;   Author: L.Jemi (VK6YEH)
;   Purpose:These routines control the radio monitor unit
;   Release Date:  20th January 2002
;   Modification History:  20/01/2002      Initial release
;-----

        include p16F84.inc

;-----
;   definitions
;-----

#define BEACON_OP  0      ; define this if used in a beacon
                       ; otherwise comment it out for repeater
#define DOT        0x55  ; number of timer cycles for a morse dot
#define DASH      0xff  ; number of timer cycles for a morse dash
#define BIT_DELAY  0x56  ; bit space space
#define CHAR_DELAY 0xfe  ; character space
#define STATE_0    0      ; wait for tx interval
#define STATE_1    0x01  ; send the station ID
; transmission interval 5 seconds
#define TX_INTVL_LOW  0x88 ; transmission interval (ms) low
#define TX_INTVL_HIGH 0x13 ; transmission interval (ms) high

```

```

#define TIMER_DLY    0x83    ; timer delay (125) along with prescaler of 4
                        ; gives 1ms delay
#define CHAN_A_BIT  0        ; Channel A input
#define CHAN_B_BIT  1        ; Channel B input
#define CHAN_C_BIT  2        ; Channel C input
#define CHAN_D_BIT  3        ; Channel D input
#define CHAN_E_BIT  4        ; Channel E input
#define CHAN_F_BIT  5        ; Channel F input
#define CHAN_G_BIT  6        ; Channel G input
#define CHAN_H_BIT  7        ; Channel H input
#define TONE_BIT    0        ; Tone output
#define TX_EN_BIT   1        ; Tx enable output
#define OFF         0        ; off
#define ON          1        ; on

```

```

;-----
; morse code alphabet
;-----

```

```

#define MORSE_A      DOT,DASH,CHAR_DELAY
#define MORSE_B      DASH,DOT,DOT,DOT,CHAR_DELAY
#define MORSE_C      DASH,DOT,DASH,DOT,CHAR_DELAY
#define MORSE_D      DASH,DOT,DOT,CHAR_DELAY
#define MORSE_E      DOT,CHAR_DELAY
#define MORSE_F      DOT,DOT,DASH,DOT,CHAR_DELAY
#define MORSE_G      DASH,DASH,DOT,CHAR_DELAY
#define MORSE_H      DOT,DOT,DOT,DOT,CHAR_DELAY
#define MORSE_I      DOT,DOT,CHAR_DELAY
#define MORSE_J      DOT,DASH,DASH,DASH,CHAR_DELAY
#define MORSE_K      DASH,DOT,DASH,CHAR_DELAY
#define MORSE_L      DOT,DASH,DOT,DOT,CHAR_DELAY
#define MORSE_M      DASH,DASH,CHAR_DELAY
#define MORSE_N      DASH,DOT,CHAR_DELAY
#define MORSE_O      DASH,DASH,DASH,CHAR_DELAY
#define MORSE_P      DOT,DASH,DASH,DOT,CHAR_DELAY
#define MORSE_Q      DASH,DASH,DOT,DASH,CHAR_DELAY
#define MORSE_R      DOT,DASH,DOT,CHAR_DELAY
#define MORSE_S      DOT,DOT,DOT,CHAR_DELAY
#define MORSE_T      DASH,CHAR_DELAY
#define MORSE_U      DOT,DOT,DASH,CHAR_DELAY
#define MORSE_V      DOT,DOT,DOT,DASH,CHAR_DELAY
#define MORSE_W      DOT,DASH,DASH,CHAR_DELAY
#define MORSE_X      DASH,DOT,DOT,DASH,CHAR_DELAY
#define MORSE_Y      DASH,DOT,DASH,DASH,CHAR_DELAY
#define MORSE_Z      DASH,DASH,DOT,DOT,CHAR_DELAY

#define MORSE_1      DOT,DASH,DASH,DASH,DASH,CHAR_DELAY
#define MORSE_2      DOT,DOT,DASH,DASH,DASH,CHAR_DELAY
#define MORSE_3      DOT,DOT,DOT,DASH,DASH,CHAR_DELAY
#define MORSE_4      DOT,DOT,DOT,DOT,DASH,CHAR_DELAY
#define MORSE_5      DOT,DOT,DOT,DOT,DOT,CHAR_DELAY
#define MORSE_6      DASH,DOT,DOT,DOT,DOT,CHAR_DELAY
#define MORSE_7      DASH,DASH,DOT,DOT,DOT,CHAR_DELAY
#define MORSE_8      DASH,DASH,DASH,DOT,DOT,CHAR_DELAY
#define MORSE_9      DASH,DASH,DASH,DASH,DOT,CHAR_DELAY
#define MORSE_0      DASH,DASH,DASH,DASH,DASH,CHAR_DELAY

```

```

;-----
; equates

```

```

;-----
opstate      equ    0x0D    ; state of processing
time_low     equ    0x0E    ; minute interval counter (low byte)
time_high    equ    0x0F    ; minute interval counter (high byte)
table_ptr    equ    0x10    ; table pointer
morse_data   equ    0x11    ; holds morse delay value
morse_dly    equ    0x12    ; holds morse delay
W_TEMP       equ    0x13    ; temporary W stack
STAT_TEMP    equ    0x14    ; temporary status stack
tone_flag    equ    0x15    ; tone enable flag
count        equ    0x16    ; delay counter

; init interrupt vectors

    org  0      ;start address 0
    goto start

    org  0x04   ; interrupt start address 0x04
    goto timerInt

; start of program

    org  0x08

;-----
;      station identifier - VK6YEH
;-----
station_id

    addwf PCL,F
    dt    MORSE_V,MORSE_K,MORSE_6,MORSE_Y,MORSE_E,MORSE_H,0

;-----
;      Fault channel identifier
;-----
curr_stat_a

    addwf PCL,F
    dt    MORSE_A,0

curr_stat_b

    addwf PCL,F
    dt    MORSE_B,0

curr_stat_c

    addwf PCL,F
    dt    MORSE_C,0

curr_stat_d

    addwf PCL,F
    dt    MORSE_D,0

curr_stat_e

    addwf PCL,F
    dt    MORSE_E,0

curr_stat_f

    addwf PCL,F
    dt    MORSE_F,0

curr_stat_g

    addwf PCL,F
    dt    MORSE_G,0

curr_stat_h

    addwf PCL,F
    dt    MORSE_H,0

```

```

;-----
; Start of program
;-----
start:
    clrf    PORTA        ; set port A outputs low
    clrf    PORTB        ; set all port B outputs low

;-----
; Delay by timer interrupt
;-----
    clrwdt        ; clear watchdog

    bsf     STATUS,RP0   ; select bank 1

    movlw   0x1c        ; set data direction of port A
                        ; RA0 = output - tone output
                        ; RA1 = output - transmitter enable
                        ; RA2 = input
                        ; RA3 = input
                        ; RA4 = input

    movwf   TRISA

    movlw   0xff        ; set port B direction
                        ; RB0 = input
                        ; RB1 = input
                        ; RB2 = input
                        ; RB3 = input
                        ; RB4 = input
                        ; RB5 = input
                        ; RB6 = input
                        ; RB7 = input

    movwf   TRISB

    movlw   b'10000010' ; bit 7 - port b pullups disabled
                        ; bit 6 - don't care
                        ; bit 5 - internal clock
                        ; bit 4 - don't care
                        ; bit 3 - prescaler set to TMR0
                        ; bit 2,1,0 - divide by 4

    movwf   OPTION_REG

    bcf     STATUS,RP0   ; select bank 0

    movlw   TIMER_DLY   ; set timer delay
    movwf   TMR0        ; set the delay clearing the prescaler as well

    bsf     INTCON,TOIE ; enable timer overflow interrupt
    bcf     INTCON,TOIF ; clear the interrupt flag

main_start:
    bcf     INTCON,GIE   ; disable global interrupts

    clrf   opstate      ; clear the operational state

    movlw  TX_INTVL_LOW ; initialize message interval
    movwf  time_low
    movlw  TX_INTVL_HIGH
    movwf  time_high

    bsf     INTCON,GIE   ; enable global interrupts

    ifdef  BEACON_OP
    movlw  ON            ; turn tone on

```

```

        movwf tone_flag
    else
        clrf    tone_flag        ; turn tone off
    endif
;-----
; state 0
;-----
state0:
    movf    opstate,W        ; wait here until the timer has expired
    sublw   STATE_0
    btfss   STATUS,Z
    goto    state1
    clrwdt          ; clear watchdog
    goto    state0
;-----
; state 1
;-----
state1:
    movf    opstate,W        ; start sending the identifier
    sublw   STATE_1
    btfss   STATUS,Z
    goto    chk_a

    #ifdef BEACON_OP
        clrf    tone_flag        ; turn tone off
    else
        bsf    PORTA,TX_EN_BIT    ; enable transmitter in repeater mode
    endif
    call    sendWordSpace    ; brief pause before message

    clrf    table_ptr; set to start of table
state1a:
    movlw   HIGH station_id ; get station ID offset
    movwf   PCLATH
    movf    table_ptr,W
    call    station_id

    movwf   morse_data        ; save value
    andlw   0xff              ; is data 0 terminator
    btfsc   STATUS,Z
    goto    state1b
    call    txMorse
    incf    table_ptr,F
    goto    state1a
state1b:
    call    sendWordSpace
;-----
; Check channel A - port B bit 0
;-----
chk_a:
    btfss   PORTB,CHAN_A_BIT    ; check channel A
    goto    chk_b
    clrf    table_ptr; set to start of table

chk_aa:
    movlw   HIGH curr_stat_a; get 'A' morse table
    movwf   PCLATH
    movf    table_ptr,W
    call    curr_stat_a

```

```

        movwf morse_data      ; save value
        andlw OFFH           ; is data 0 terminator
        btfsc STATUS,Z
        goto   chk_ab
        call   txMorse
        incf   table_ptr,F
        goto   chk_aa

chk_ab:
        call   sendWordSpace
;-----
; Check channel B - port B bit 1
;-----
chk_b:
        btfss PORTB,CHAN_B_BIT ; check channel b
        goto   chk_c
        clrf   table_ptr; set to start of table

chk_ba:
        movlw HIGH curr_stat_b ; get 'B' morse table
        movwf PCLATH
        movf   table_ptr,W
        call   curr_stat_b

        movwf morse_data      ; save value
        andlw OFFH           ; is data 0 terminator
        btfsc STATUS,Z
        goto   chk_bb
        call   txMorse
        incf   table_ptr,F
        goto   chk_ba

chk_bb:
        call   sendWordSpace
;-----
; Check channel C - port B bit 2
;-----
chk_c:
        btfss PORTB,CHAN_C_BIT ; check channel C
        goto   chk_d
        clrf   table_ptr; set to start of table

chk_ca:
        movlw HIGH curr_stat_c; get 'C' morse table
        movwf PCLATH
        movf   table_ptr,W
        call   curr_stat_c

        movwf morse_data      ; save value
        andlw OFFH           ; is data 0 terminator
        btfsc STATUS,Z
        goto   chk_cb
        call   txMorse
        incf   table_ptr,F
        goto   chk_ca

chk_cb:
        call   sendWordSpace
;-----
; Check channel D - port B bit 3
;-----

```

```

chk_d:
    btfss    PORTB,CHAN_D_BIT    ; check channel D
    goto     chk_e
    clrf     table_ptr; set to start of table
chk_da:
    movlw   HIGH curr_stat_d      ; get 'D' morse table
    movwf   PCLATH
    movf    table_ptr,W
    call    curr_stat_d

    movwf   morse_data           ; save value
    andlw   0FFH                 ; is data 0 terminator
    btfsc   STATUS,Z
    goto    chk_db
    call    txMorse
    incf    table_ptr,F
    goto    chk_da
chk_db:
    call    sendWordSpace
;-----
; Check channel E - port B bit 4
;-----
chk_e:
    btfss    PORTB,CHAN_E_BIT    ; check channel E
    goto     chk_f
    clrf     table_ptr; set to start of table
chk_ea:
    movlw   HIGH curr_stat_e; get 'E' morse table
    movwf   PCLATH
    movf    table_ptr,W
    call    curr_stat_e

    movwf   morse_data           ; save value
    andlw   0FFH                 ; is data 0 terminator
    btfsc   STATUS,Z
    goto    chk_eb
    call    txMorse
    incf    table_ptr,F
    goto    chk_ea
chk_eb:
    call    sendWordSpace
;-----
; Check channel F - port B bit 5
;-----
chk_f:
    btfss    PORTB,CHAN_F_BIT    ; check channel F
    goto     chk_g
    clrf     table_ptr; set to start of table
chk_fa:
    movlw   HIGH curr_stat_f; get 'F' morse table
    movwf   PCLATH
    movf    table_ptr,W
    call    curr_stat_f

    movwf   morse_data           ; save value
    andlw   0FFH                 ; is data 0 terminator
    btfsc   STATUS,Z
    goto    chk_fb
    call    txMorse

```

```

        incf    table_ptr,F
        goto   chk_fa

chk_fb:
        call   sendWordSpace
;-----
; Check channel G - port A bit 2
;-----
chk_g:
        btfss  PORTA,CHAN_C_BIT    ; check channel G
        goto   chk_h
        clrf   table_ptr; set to start of table
chk_ga:
        movlw  HIGH curr_stat_g      ; get 'G' morse table
        movwf  PCLATH
        movf   table_ptr,W
        call   curr_stat_g

        movwf  morse_data            ; save value
        andlw  0FFH                  ; is data 0 terminator
        btfsc  STATUS,Z
        goto   chk_gb
        call   txMorse
        incf   table_ptr,F
        goto   chk_ga

chk_gb:
        call   sendWordSpace
;-----
; Check channel H - port A bit 3
;-----
chk_h:
        btfss  PORTA,CHAN_D_BIT    ; check channel H
        goto   key_down
        clrf   table_ptr; set to start of table
chk_ha:
        movlw  HIGH curr_stat_h      ; get 'H' morse table
        movwf  PCLATH
        movf   table_ptr,W
        call   curr_stat_h

        movwf  morse_data            ; save value
        andlw  0FFH                  ; is data 0 terminator
        btfsc  STATUS,Z
        goto   chk_hb
        call   txMorse
        incf   table_ptr,F
        goto   chk_ha

chk_hb:
        call   sendWordSpace ; brief pause at end of message
;-----
; Key down
;-----
key_down:
        ifndef BEACON_OP    ; if operating in repeater mode
        bcf    PORTA,TX_EN_BIT    ; disable transmitter
        endif

        goto   main_start        ; do it again

```

```

;-----
; Send a word delay
;-----
sendWordSpace:
    movlw 0x03
    movwf count           ; set the number of delay increments

sendWordSpaceA:
    movlw CHAR_DELAY
    movwf morse_dly
    call morseWait       ; wait for delay to complete
    decfsz count,f      ; decrement counter
    goto sendWordSpaceA
    return

;-----
; Send a morse bit
;-----
txMorse:
    movf morse_data,W
    sublw DOT           ; is it a DOT
    btfsc STATUS,Z
    goto txMorse1
    movf morse_data,W
    sublw DASH         ; is it a DASH
    btfsc STATUS,Z
    goto txMorse1
    movf morse_data,W ; set other delay
    movwf morse_dly
    call morseWait     ; wait for delay to complete
    return

txMorse1:
    movlw ON           ; turn on tone
    movwf tone_flag
    movf morse_data,W ; set tone delay
    movwf morse_dly
    call morseWait    ; wait for tone to complete
    movlw OFF         ; turn off tone
    movwf tone_flag
    movlw BIT_DELAY   ; set bit delay
    movwf morse_dly
    call morseWait    ; wait for it to complete

txMorse2:
    return

;-----
; Morse delay
;-----
morseWait:
    clrwdt             ; reset watchdog
    movf morse_dly,W  ; see if the delay has expired
    andlw OFFH
    btfss STATUS,Z
    goto morseWait
    return

;-----
; Timer interrupt
;-----
timerInt:
    movwf W_TEMP ; save current W register setting

```

```

swapf STATUS,W
movwf STAT_TEMP    ; save current status register

    movlw TIMER_DLY    ; set timer delay
    movwf TMR0        ; set the delay clearing the prescaler as well
;-----
; 1ms timer
;-----
proc_0:
    movf  opstate,W    ; the timer has expired. Are we still in state 0
    sublw STATE_0      ; see if in state 0
    btfss STATUS,Z
    goto  proc_1
    decfsz time_low,f  ; decrement timer low byte
    goto  proc_1
    movf  time_high,W  ; is the high byte 0
    btfsc STATUS,Z
    goto  proc_0a
    decf  time_high,f  ; decrement timer high byte
    goto  proc_1
proc_0a:
    movlw STATE_1    ; set next state
    movwf opstate

proc_1:
    movf  opstate,W    ; are we in state 1
    sublw STATE_1
    btfss STATUS,Z
    goto  proc_2
    movf  morse_dly,W  ; see if the delay counter is already 0
    andlw OFFH
    btfsc STATUS,Z
    goto  proc_2
    decf  morse_dly,F  ; decrement delay counter
proc_2:
    movf  tone_flag,W  ; control the tone output
    sublw ON           ; see if the tone is enabled
    btfss STATUS,Z
    goto  proc_2a
    btfss PORTA,TONE_BIT ; is the tone bit high
    goto  proc_2b
proc_2a:
    bcf   PORTA,TONE_BIT ; set tone bit LOW
    goto int_exit
proc_2b:
    bsf   PORTA,TONE_BIT ; set tone bit HIGH
    goto int_exit
int_exit
    bcf   INTCON,T0IF    ; clear the interrupt flag
    swapf STAT_TEMP,W    ; restore status register
    movwf STATUS
    swapf W_TEMP,F
    swapf W_TEMP,W
    retfie

    end                ; end of program

```