Order this document
as AN463/D

MOTOROLA
SEMICONDUCTOR
APPLICATION NOTE

AN463

68HC05K0 Infra-red Remote Control

Tony Breslin,
MCU Applications Group,
Motorola Ltd., East Kilbride, Scotland

The MC68HC05K0 is a low cost, low pin count single chip microcomputer with 504 bytes of user ROM and 32 bytes of RAM. The MC68HC05K0 is a member of the 68HC05K series of devices which are available in 16-pin DIL or SOIC packages. It uses the same CPU as the other devices in the 68HC05 family and has the same instructions and registers. Additionally, the device has a 15-stage multi-function timer and 10 general purpose bi-directional I/0 lines. A mask option is available for software programmable pull-downs on all of the I/O pins and four of the pins are capable of generating interrupts.

The device is ideally suited for remote-control keyboard applications because the pull-downs and the interrupt drivers on the port pins allow keyboards to be built without any external components except the keys themselves. There is no need for external pull-up or pull-down resistors, or diodes for wired-OR interrupts, as these features are already designed into the device.

This application makes use of many of the device features to control an infra-red television remote control. The application could be very easily modified to control any device with a similar transmission protocol. It will run on any of the 'K' devices without modification.

Remote Control Specifications

The basic purpose of a television remote control is to transmit a control instruction to the television. The instruction is generated by a keystroke on the remote control keyboard. The detection and decoding of a key press and the transmission encoding is carried out by the remote control micro controller.

When a key on the remote control keypad is pressed, the micro controller must first determine what key is being pressed and generate an individual code for the key. The key code is then converted to a instruction code that is inserted into the transmission command which, using a defined protocol, is transmitted to the television receiver. The command is continually transmitted as long as the key is being held down.

As the remote control is battery powered it needs to use as little power as possible. This is achieved by entering STOP mode when no keys are being pressed and effectively switches off the device. The micro controller comes out of STOP mode upon receipt of an interrupt request that is generated when a key is pressed.

Remote Control Keyboard

The 68HC05K0 has ten general purpose I/O pins. One of these is used for the transmission signal output leaving nine pins for the keyboard control. Of these, four pins on PortA have internal interrupt request hardware. Using these four pins as inputs allows key presses to be detected without any external interrupt hardware. This leaves the five remaining pins for outputs.

Using the internal pull-down facility and the rising edge interrupt request on the four inputs permits interrupts to be generated. **If the five outputs are set to logic '1', so driving an input from logic '0' to logic '1' when a key is pressed, an interrupt request can be generated**. Using this arrangement a five by four keyboard matrix can be used. An extra four keys can be controlled if the Vdd line is used to drive one row of four keys to logic '1'. Therefore the maximum amount of keys controllable becomes twenty four.

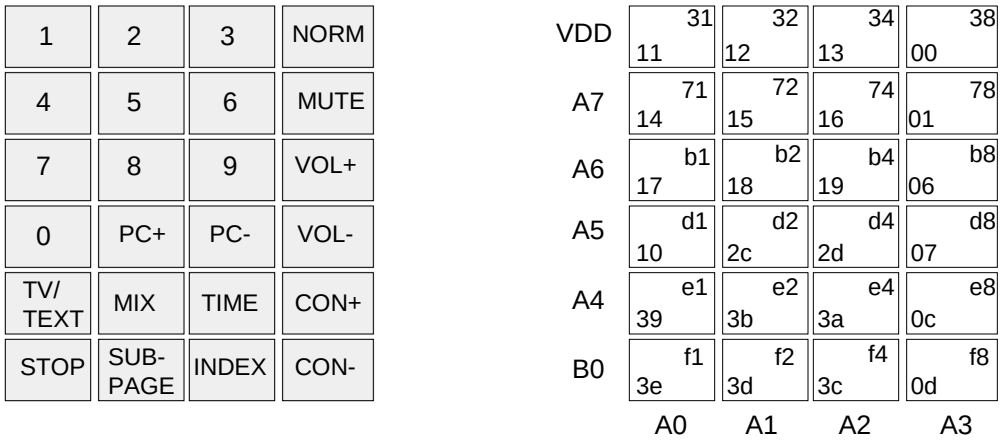


Figure 1 Keyboard layout with associated scanned and transmitted codes

A depressed key will set one of the input columns to logic '1'. By scanning the columns, and setting each row output to logic '0' and then checking if the inputs all become logic '0', the associated row for the key can be determined. If rotating the logic '0' through the five output pins fails to identify a key column, then the key must be connected to the Vdd line. This process gives an individual code for each key which is a combination of the code from the column inputs and the row outputs. This can then be decoded to an instruction that is inserted into the output signal for transmission.

Figure 1 shows the layout of the keyboard on the left and the scanned and transmitted codes on the right. The keyboard layout incorporates the various television controls plus controls for TELETEXT. On the left hand side the codes returned from scanning the keyboard are shown in the upper right-hand corner of each key and the code sent for transmission for that key instruction are shown in the bottom left-hand corner. The I/O pins for each row and column are also shown for each key.

Transmission Protocol

The transmission protocol in this application is that used by the MC144105 IR Remote Control Transmitter. It uses a binary coded 9-bit data word with the LSB being transmitted first. Each bit of the transmitted signal is in the form of a bi-phase pulse code modulated (PCM) signal, whose bit coding is shown in Figure 2. For a transmitted '0' there is a 512µs pause followed by a 32kHz pulse train for 512µs. For a transmitted '1' there is 32kHz pulse train followed by a 512µs pause. This gives a bit time of 1024µs for all bits.This is shown asFigure 2.

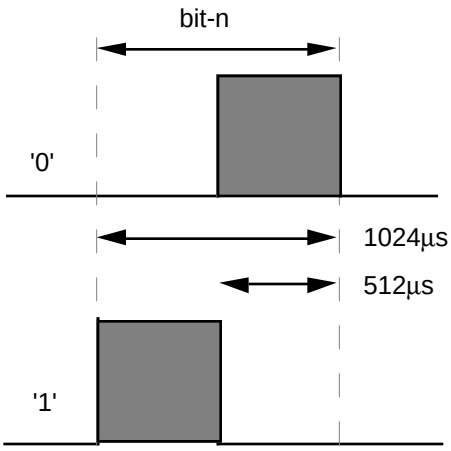


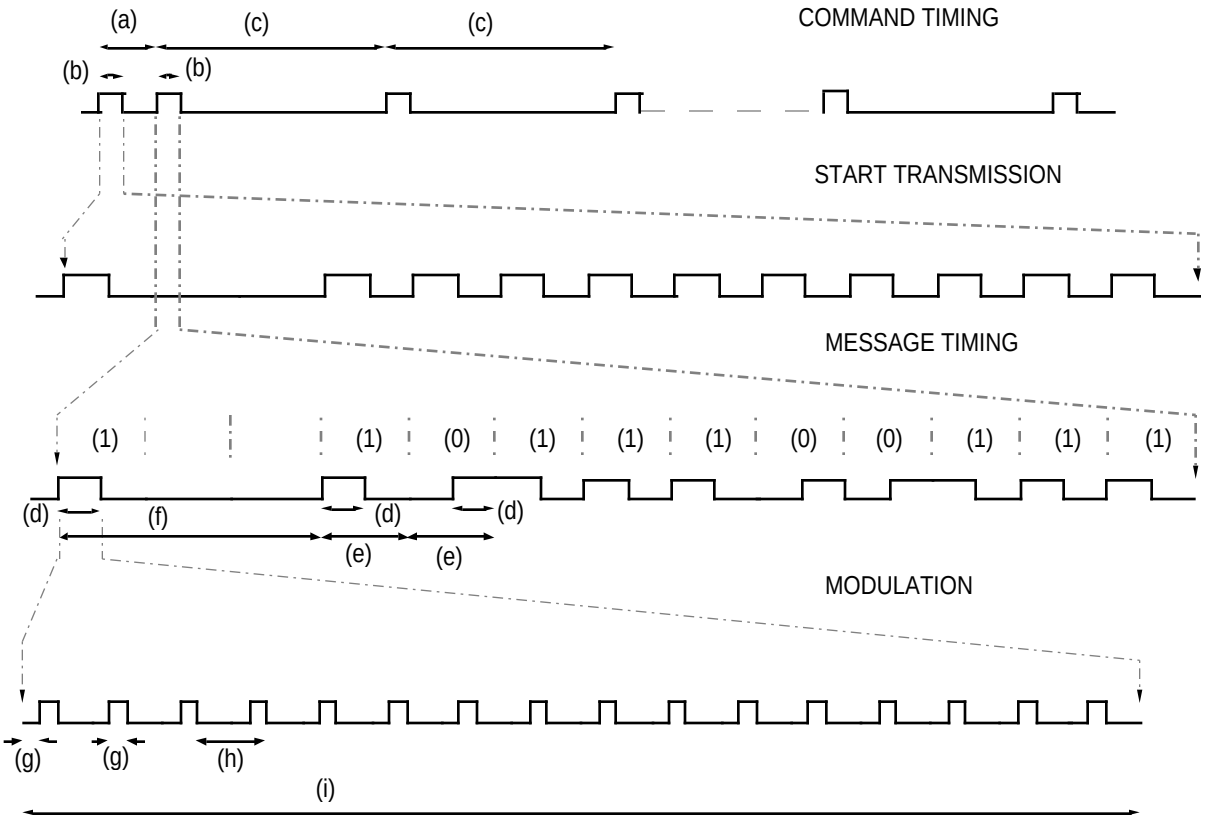
Figure 2 Bit coding of PCM signal

A complete transmission command consists of several messages. Each command begins with a start message of nine 1's followed by the message appropriate to the key pressed. This message is repeatedly transmitted until the key is released. The transmission is terminated after the key is released by a end message of nine 1's.

Every message consists of a pre-bit, a pre-bit pause, a start bit and nine data bits. The pre-bit and the start-bit are always logical '1'. The pre-bit allows for the set up of the automatic gain control in the receiving preamplifier. Figure 3 gives the exact timing relationships for the transmissions.

The command timing in Figure 3 shows that after the start transmission the message is continually re-transmitted at intervals of 131ms (approximately 8Hz) until the key is released. This is shown as time (c). The control timing shows the nine bit instruction 111001110 being transmitted starting with the LSB. The pre-bit pause is equal to two bit periods and is followed by a start-bit of logical '1'. The pulse train is continuous during the transition between transmitting a logic '0' and a logic '1'. The modulating pulse train has a frequency of approximately 32kHz with a mark-to-space ratio of one to three.

The signal for transmission is output through one port pin and is used to drive an IR diode amplifier circuit.



Command timing	(a) = 32.8ms (b) = 13.3ms (c) = 131ms	start transmission start command control transmission	13 * bit time
Message timing	(d) = 512µs (e) = 1.024ms (f) = 3.072ms	(half-bit time) (bit time) (pre-pulse time)	$16/f_{carrier}$
Modulation	(g) = 8µs (h) = 32µs (i) = 512µs	$(1/f_{carrier})$ (half-bit time)	$16/f_{carrier}$

Figure 3 Circuit timing

Remote Control Operation

Figure 4 is a flow diagram showing the operation of the remote control on power-up or reset. After the initial set-up of the ports as inputs or outputs the remote control goes into STOP mode. It will remain in STOP mode as long as the device is not reset or a key is not pressed. When a key is pressed an interrupt request is generated. A short time delay makes sure that it is a true key press and not noise and also allows time for any switching effects on the inputs to pass prior to checking the inputs.

The keyboard is then read to find which key has been pressed and the code for the key is decoded into an instruction and transmitted to the television. If the key is held down the instruction is re-transmitted until the key is released. This is useful for the instructions which count through the television channels or adjust the volume, colour or brightness controls.

When the key is released a terminating instruction is sent to the receiver to inform it that the next message received is a separate instruction. This is useful in the case of a one time instruction like sending a channel number. In this example the receiver will tune to a channel only once; to tune to another channel the key must be released and a new instruction sequence received.

After terminating the transmission the ports are reset ready for the next key press and the processor returns to the STOP mode.

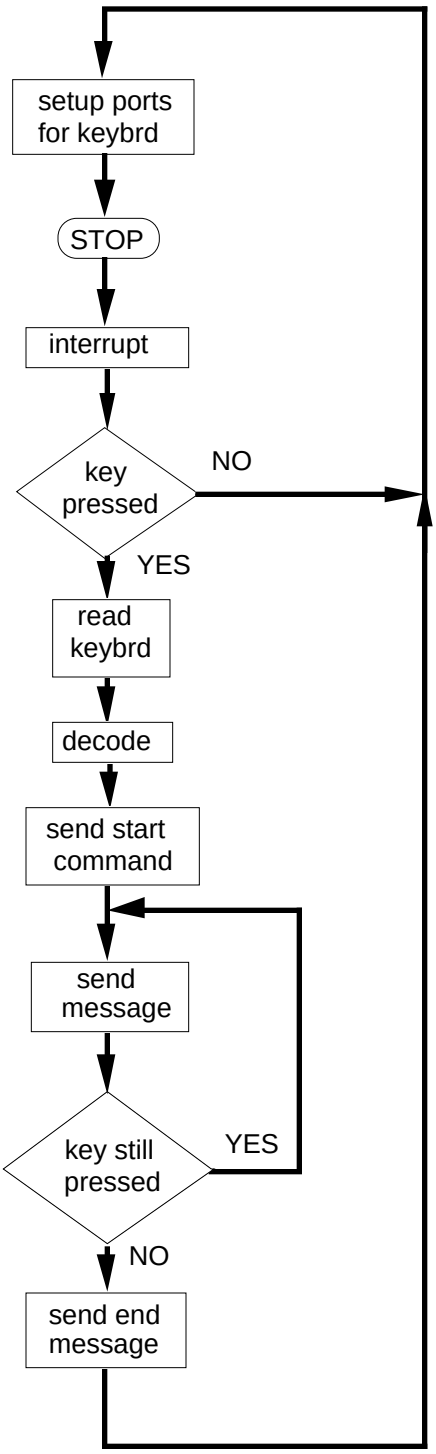


Figure 4 Flow diagram

Hardware

The remote control circuit is shown in Figure 5. The hardware consists of the keyboard, the oscillator and the infra-red amplifier. The oscillator can be a crystal or a ceramic resonator with a frequency of 2MHz. The oscillator frequency is important since the transmission timing is based around a 1MHz internal clock frequency.

The infra-red amplifier uses two transistors and two standard diodes to limit the current through the IR diodes to approximately 1A. There is a need for a large capacitor close to the IR diodes because of the high switching current of the circuit.

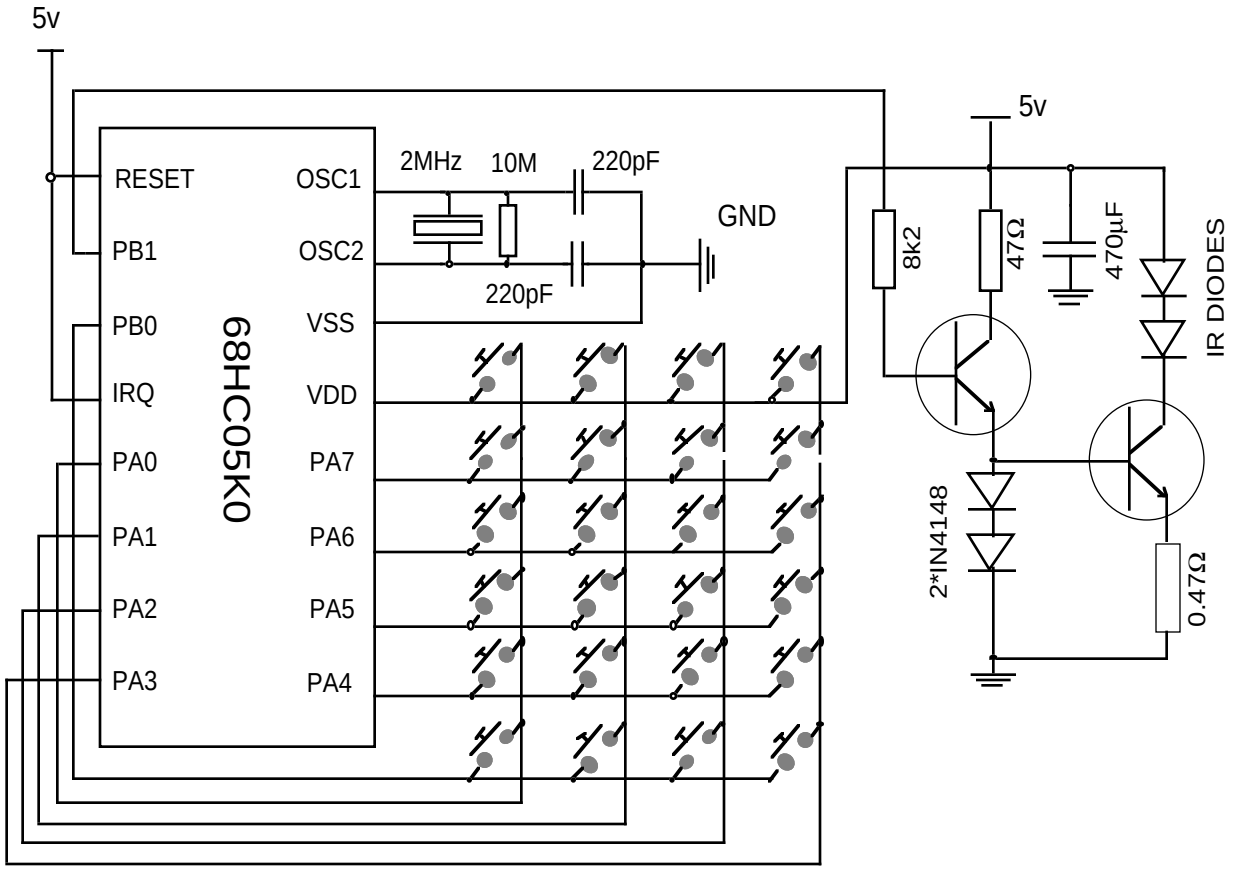


Figure 5 Infra-red remote control

Software

The listing of the remote control assembler code is contained at the end of this application note. The first section of the listing sets up the ports prior to going into STOP mode and waiting for a key to be pressed. PortA bits 0-3 are set up as inputs with the pull-downs enabled. Bits 4-7 are set up as outputs logic '1' as is PortB bit 0. PortB bit 1 is set-up as output logic '0' to switch off the IR amplifier before going into STOP mode.

The next section of code named 'presd' is the routine pointed to by the interrupt vector and is entered when a key is pressed. This routine first calls the keyboard scanning routine to determine which key has been pressed. It then calls the decoding routine to convert the code from the keyboard to a code that will be accepted by the television. The start message is then transmitted and is followed by the instruction message. There is then a check to see if the same key is still being pressed. If it is then the instruction message is re-transmitted until the key is released and the end message is transmitted.

As the transmission protocol requires nine data bits and only one byte instructions are being decoded a flag has to be set for the ninth bit of the transmission routine. For the start and end transmissions this flag is set to 1 to give the nine 1's message. For all instructions the ninth bit is 0 so the flag is cleared.

The decoding routine compares the code from the keyboard scan routine with data array 'keydat'. On a match it takes the corresponding element from the array 'tvdat' as the instruction code for transmission.

The values of the instruction codes shown in the right-hand side of Figure 1 are specific for the receiver application. Each receiver using the same communications protocol will receive the same nine bit instruction but what the instruction does is dependent upon the receiver software. In this example the eight bit instruction '14' changes the channel to number four. In another receiver application the receiver may interpret the instruction code '14' as increase volume.

The transmission routine is entered with the instruction for transmission in 'keyst3'. After the pre-bit and the start-bit are transmitted the instruction byte is rotated (LSB first) into the carry flag. A logic '1' is sent for transmission if the flag is set after rotation and a logic '0' is sent for transmission if the flag is cleared. Each bit is transmitted as shown in Figure 1. The routines 'send0' and 'send1' send a pause of 512µs followed by a 32kHz pulse train for 512µs and a 32kHz pulse train for 512µs followed by a 512µs pause respectively. In the situation when a '1' follows a '0' then a pulse train of 1024µs is required. To avoid breaks in this pulse train the 'send0' routine checks the next bit to be transmitted to see if a double length pulse train must be transmitted. The 'send1' routine then has to check that a double length pulse train has not been sent in the previous one and a half bit periods before sending a pulse train.

The routine 'burst' produces the 32kHz pulse train for a duration set by a count in the accumulator. As the instruction time for setting the PortB bit 1 pin high or low is five clock cycles then the minimum processor clock period is derived by dividing the minimum output state time, which is 8µs when the output is high, by the minimum number of clock cycles to change this state. This gives an internal clock period of 8µs/5 equalling 1.6µs. Adding a three cycle delay will require an internal clock period of 8µs/8 = 1µs, allowing a 2MHz oscillator to be used.

The code size is approximately 300 bytes, leaving memory space for more features to be added to the controller.

Debug

On applying power to the circuit the RESET vector will initialise the program counter at the beginning of the software. When examining the output at PortB bit 1 with an oscilloscope or logic analyser it should be noted that when trying to capture the signal by pressing a key the first signal out will be the start message of nine 1's. To capture the instruction the key should be held down and as the instruction will be continually re-transmitted then the capture can be initiated at this point.

Listing

```
0026                        ************************************************************
0027                        * INFRA RED REMOTE CONTROL FOR K0,K1                       *
0028                        ************************************************************
0029                        * WRITTEN BY A.BRESLIN    13.1.92                          *
0030                        ************************************************************
0031                        * THIS PROGRAM READS AND ENCODES A KEY FROM A 24 KEY KEYBOARD *
0032                        * TO A FORM OF BIPHASE PULSE CODE MODULATION (PCM) FOR INFRA  *
0033                        * RED TRANSMISSION. IT USES THE TRANSMISSION PROTOCOL OF THE  *
0034                        * MC144105 IR REMOTE CONTROL TRANSMITTER                      *
0035                        ************************************************************
0036
0037
0038 0000                  porta   equ     00
0039 0001                  portb   equ     01
0040 0004                  ddra    equ     04
0041 0005                  ddrb    equ     05
0042 0008                  tcsr    equ     $08
0043 0010                  papd    equ     $10
0044
0045 00e0                          org     $e0
0046
0047 00e0                  keyst1  rmb     1               ; initial code from keyboard
0048 00e1                  keyst2  rmb     1               ; keycode
0049 00e2                  keyst3  rmb     1               ; code transmitted
0050 00e3                  dflag   rmb     1               ; flag for last and 9th bits
0051
0052
0053                        ************************************************************
0054                        * THE PORTS ARE SET UP USING PORTA 0-3 AS INPUTS MAKING USE  *
0055                        * OF THE INTERNAL INTERUPT GENERATION ON THESE I/0 LINES.    *
0056                        * STOP MODE IS ENTERED UNTIL A KEY IS PRESSED                *
0057                        ************************************************************
0058
0059 0200                          org     $200
0060
0061 0200 9a               start   cli
0062 0201 ad 04            wpres   bsr     setup
0063 0203 9c                       rsp
0064 0204 8e                       stop
0065 0205 20 fa                    bra     wpres
0066
0067 0207 a6 f0            setup   lda     #$f0            ; porta 0-3 inputs
0068 0209 b7 04                    sta     ddra            ; 4-7 as outputs
0069 020b b7 00                    sta     porta           ; set outputs high
0070 020d b7 10                    sta     papd            ; 0-3 pulldown
0071 020f a6 03                    lda     #$03            ; portb 0-1 outputs
0072 0211 b7 05                    sta     ddrb
0073 0213 a6 01                    lda     #$01            ; set portb 0 high
0074 0215 b7 01                    sta     portb
0075 0217 81                       rts
0076
0077
```

```
0078                        ************************************************************
0079                        * THE KEY READ IS DECODED FOR TRANSMISSION.                 *
0080                        * THE TRANSMISSION PROTOCOL REQUIRES A START MESSAGE OF 9   *
0081                        * ONES FOLLOWED BY THE KEYPRESSED CODE. THIS CODE IS        *
0082                        * CONTINUALLY RETRANSMITTED IF THE KEY IS HELD DOWN. AN END  *
0083                        * CODE OF 9 ONES TERMINATES THE TRANSMISSION AND THE DEVICE  *
0084                        * RETURNS TO STOP MODE.                                     *
0085                        ************************************************************
0086
0087 0218 ad 34             presd   bsr     keyscn          ; get key pressed
0088 021a b6 e1                     lda     keyst2          ; save key to check
0089 021c b7 e0                     sta     keyst1          ; if key held down
0090 021e ad 67                     bsr     decode          ; decode key pressed
0091 0220 12 e3                     bset    1,dflag         ; set nineth bit to 1
0092 0222 a6 ff                     lda     #$ff            ; send start data
0093 0224 b7 e2                     sta     keyst3          ; to transmission routine
0094 0226 ad 71                     bsr     trnmit          ; nine one's
0095 0228 b6 e1             sndagn  lda     keyst2          ; send key press message
0096 022a b7 e2                     sta     keyst3          ; byte
0097 022c 13 e3                     bclr    1,dflag         ; set nineth bit to 0
0098 022e ad 69                     bsr     trnmit
0099 0230 b6 00                     lda     porta           ; check if key still pressed
0100 0232 a4 0f                     and     #$0f            ; end if no key pressed
0101 0234 26 0f                     bne     endtrn
0102 0236 ad 16                     bsr     keyscn          ; else check if same
0103 0238 b6 e0                     lda     keyst1          ; key pressed
0104 023a b1 e1                     cmp     keyst2
0105 023c 26 07                     bne     endtrn          ; end if not
0106 023e ae c8                     ldx     #$c8            ; delay
0107 0240 5a                tloop   decx                    ; before next
0108 0241 26 fd                     bne     tloop           ; transmission
0109 0243 20 e3                     bra     sndagn
0110 0245 12 e3             endtrn  bset    1,dflag         ; send end message
0111 0247 a6 ff                     lda     #$ff            ; of nine ones
0112 0249 b7 e2                     sta     keyst3
0113 024b ad 4c                     bsr     trnmit
0114 024d 80                        rti                     ; re-enter stop mode
0115
0116                        ************************************************************
0117                        * WHEN A KEY IS PRESSED THE DEVICE COMES OUT OF STOP MODE    *
0118                        * THE KEYBOARD IS SCANNED TO SEE WHICH KEY IS PRESSED        *
0119                        ************************************************************
0120
0121 024e cd 02 fc          keyscn  jsr     datwt           ; wait for debounce
0122 0251 b6 00                     lda     porta           ; check if key press
0123 0253 b7 e0                     sta     keyst1          ; store inputs
0124 0255 a4 0f                     and     #$0f            ; mask outputs
0125 0257 27 a7                     beq     start           ; stop if no key pressed
0126 0259 ae ef                     ldx     #$ef            ; set one row low
0127 025b 9f                nxtrow  txa                     ; read ouput lines
0128 025c b4 e0                     and     keyst1          ; combine with inputs
0129 025e b7 e1                     sta     keyst2          ; store key code
0130 0260 bf 00                     stx     porta           ; to find row which clears inputs
0131 0262 b6 00                     lda     porta           ; check for inputs cleared
0132 0264 a4 0f                     and     #$0f            ; mask outputs
0133 0266 27 1c                     beq     gotit           ; zero in key-press row clears inputs
0134 0268 58                        lslx                    ; check if last row
0135 0269 5c                        incx                    ; set lsb to 1
0136 026a 24 02                     bcc     tryb            ; try portb output if not porta
0137 026c 20 ed                     bra     nxtrow          ; try next porta output row
0138
0139 026e b6 e0             tryb    lda     keyst1
0140 0270 b7 e1                     sta     keyst2
0141 0272 ae f0                     ldx     #$f0
0142 0274 bf 00                     stx     porta           ; set all porta outputs high
0143 0276 11 01                     bclr    0,portb         ; set portb 0 output low
0144 0278 b6 00                     lda     porta           ; check for inputs cleared
0145 027a a4 0f                     and     #$0f            ; mask outputs
0146 027c 27 06                     beq     gotit           ; zero in key-press row clears inputs
0147 027e b6 e1                     lda     keyst2          ;
0148 0280 a4 3f                     and     #$3f            ; set individual code since last row
0149 0282 b7 e1                     sta     keyst2          ; store code
0150 0284 10 01             gotit   bset    0,portb         ; set portb column high again
0151 0286 81                        rts
0152
```

```
0153                        ************************************************************
0154                        * THE DECODE ROUTINE USES TWO ARRAYS. IT COMPARES THE KEY    *
0155                        * VALUE WITH THE ARRAY KEYDAT AND WHEN A MATCH IS FOUND THE  *
0156                        * CORRESPONDING ELEMENT IN THE ARRAY TVDAT BECOMES THE       *
0157                        * TRANSMITTED CODE.                                          *
0158                        ************************************************************
0159
0160 0287 ae 18             decode  ldx     #$18            ; data array offset to zero
0161 0289 d6 03 02          nxtel   lda     keydat,x        ; look at each element of array
0162 028c b1 e1                     cmp     keyst2          ; compare with key read
0163 028e 27 03                     beq     match           ; decode if match
0164 0290 5a                        decx                    ; else try next element
0165 0291 26 f6                     bne     nxtel           ; norm if no match found
0166 0293 d6 03 1a          match   lda     tvdat,x         ; get key code
0167 0296 b7 e1                     sta     keyst2          ; store code to transmit
0168 0298 81                        rts
0169
0170                        ************************************************************
0171                        * THE TRANSMISSION PROTOCOL REQUIRES A PRE-BIT, A PRE-BIT    *
0172                        * PAUSE, A START BIT AND NINE DATA BITS, WHERE THE PRE-BIT   *
0173                        * AND THE START BIT ARE LOGIC '1'.                          *
0174                        ************************************************************
0175
0176 0299 10 e3             trnmit  bset    0,dflag         ; initialise for first bit
0177 029b ad 32                     bsr     send1           ; send pre-bit
0178 029d cd 02 fc                  jsr     datwt           ; pre-bit pause
0179 02a0 cd 02 fc                  jsr     datwt           ; equalling four half data periods
0180 02a3 cd 02 fc                  jsr     datwt           ;
0181 02a6 cd 02 fc                  jsr     datwt           ;
0182 02a9 ad 24                     bsr     send1           ; send start bit
0183 02ab ae 08                     ldx     #$08            ; transmit 8 data bits
0184 02ad 34 e2             nxtbit  lsr     keyst3          ; get next bit
0185 02af 25 04                     bcs     data1           ; send 1 if carry set
0186 02b1 ad 28                     bsr     send0           ; send 0 if carry clear
0187 02b3 20 02                     bra     bitsnt
0188 02b5 ad 18             data1   bsr     send1
0189 02b7 5a                bitsnt  decx                    ; countdown bits sent
0190 02b8 26 f3                     bne     nxtbit          ; send next bit if count not zero
0191 02ba 03 e3 04                  brclr   1,dflag,send00  ; if flag set
0192 02bd ad 10                     bsr     send1           ; send 1 as nineth bit
0193 02bf 20 02                     bra     endend          ;
0194 02c1 ad 18             send00  bsr     send0           ; else send 0
0195 02c3 ae 18             endend  ldx     #$18
0196 02c5 ad 35             loopw   bsr     datwt           ; delay between successive
0197 02c7 ad 33                     bsr     datwt           ; transmissions
0198 02c9 ad 31                     bsr     datwt
0199 02cb 5a                        decx
0200 02cc 26 f7                     bne     loopw
0201 02ce 81                        rts
0202
0203
0204                        ************************************************************
0205                        * TO TRANSMIT A LOGIC '1' A 32kHz PULSE TRAIN FOR 512us IS   *
0206                        * FOLLOWED BY A 512us PAUSE.                                 *
0207                        ************************************************************
0208
0209 02cf 01 e3 04          send1   brclr   0,dflag,last0   ; check if last bit was zero
0210 02d2 a6 10                     lda     #$10            ; burst if last bit was 1
0211 02d4 ad 15                     bsr     burst           ; 32kHz pulse for 512us
0212 02d6 ad 24             last0   bsr     datwt           ; wait 512us
0213 02d8 10 e3                     bset    0,dflag         ; set flag as 1 sent
0214 02da 81                        rts
0215
```

```
0216                        ************************************************************
0217                        * TO TRANSMIT A LOGIC '0' A 512us PAUSE IS FOLLOWED BY A     *
0218                        * 32kHz PULSE TRAIN FOR 512us. IF A LOGIC '1' FOLLOWS A '0'  *
0219                        * THE 32kHz IS CONTINUED FOR 1024us TO AVOID A PROCESSING    *
0220                        * DELAY                                                      *
0221                        ************************************************************
0222
0223 02db ad 1f             send0   bsr     datwt           ; wait 512us
0224 02dd 00 e2 04                  brset   0,keyst3,next1  ; check if next bit is 1
0225 02e0 a6 10                     lda     #$10            ; single burst if 1
0226 02e2 20 02                     bra     datset          ; data set
0227 02e4 a6 20             next1   lda     #$20            ; double burst required
0228 02e6 ad 03             datset  bsr     burst           ; 32kHz pulse for 512us
0229 02e8 11 e3                     bclr    0,dflag         ; clear flag as 0 sent
0230 02ea 81                        rts
0231
0232                        ************************************************************
0233                        * THE 32kHz PULSE TRAIN HAS A MARK TO SPACE RATIO OF 1 TO 3  *
0234                        ************************************************************
0235
0236 02eb 13 01             burst   bclr    1,portb         ; portb 1 low
0237 02ed 21 fe                     brn     *
0238 02ef 12 01                     bset    1,portb         ; portb 1 high
0239 02f1 21 fe                     brn     *
0240 02f3 13 01                     bclr    1,portb         ; portb 1 low
0241 02f5 9d                        nop
0242 02f6 4a                        deca                    ; decrement count
0243 02f7 27 02                     beq     endbur          ; end of burst ?
0244 02f9 20 f0                     bra     burst
0245 02fb 81                endbur  rts
0246
0247
0248 02fc a6 52             datwt   lda     #$52            ; count
0249 02fe 4a                loop    deca                    ; to provide 512us delay
0250 02ff 26 fd                     bne     loop            ; after instruction times
0251 0301 81                        rts
0252
0253 0302 31 f1 e1 d1 b1 71 keydat  fcb     $31,$f1,$e1,$d1,$b1,$71
0254 0308 32 f2 e2 d2 b2 72         fcb     $32,$f2,$e2,$d2,$b2,$72
0255 030e 34 f4 e4 d4 b4 74         fcb     $34,$f4,$e4,$d4,$b4,$74
0256 0314 38 f8 e8 d8 b8 78         fcb     $38,$f8,$e8,$d8,$b8,$78
0257
0258 031a 11 3e 39 10 17 14 tvdat   fcb     $11,$3e,$39,$10,$17,$14
0259 0320 12 3d 3b 2c 18 15         fcb     $12,$3d,$3b,$2c,$18,$15
0260 0326 13 3c 3a 2d 19 16         fcb     $13,$3c,$3a,$2d,$19,$16
0261 032c 00 0d 0c 07 06 01         fcb     $00,$0d,$0c,$07,$06,$01
0262
0263
0264 0332 80                softin  rti
0265
0266 03fa                           org     $3fa
0267
0268 03fa 02 18                     fdb     presd           ; scan keybrd on int
0269 03fc 03 32                     fdb     softin          ; software interrupt
0270 03fe 02 00                     fdb     start           ; resett
```

Literature Distribution Centres:
EUROPE: Motorola Ltd., European Literature Centre, 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.
ASIA PACIFIC: Motorola Semiconductors (H.K.) Ltd., Silicon Harbour Center, No. 2, Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.
JAPAN: Nippon Motorola Ltd., 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141, Japan.
USA: Motorola Literature Distribution, P.O. Box 20912, Phoenix, Arizona 85036.