Order this document
as AN465/D

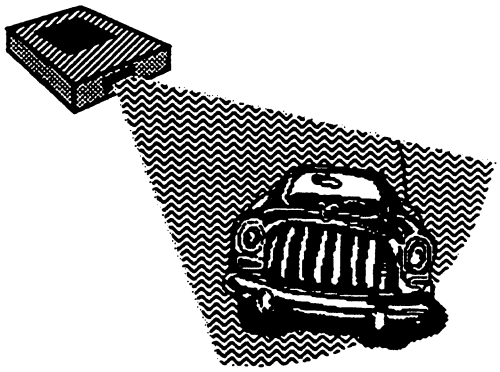
AN465

Secure Remote Control using the 68HC05K1 and the 68HC05P3

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

INTRODUCTION

This application note shows how the 68HC05K1 and the 68HC05P3 can be used together to form a multi-user secure remote control system. This system is especially suitable for enabling or disabling car alarms or for operating car central locking. It also has applications to alarm systems or secure entry systems in general. In the system described here, the transmitter will send an infrared signal to the receiver every time the single key on the transmitter is pressed. The receiver on receipt of this signal may then open or lock a door or enable/disable an alarm.



In order to make the system secure against attempts to capture the signal and retransmit it later for unauthorised purposes, the transmitter transmits a different signal every transmission. Two security problems arise here: how to encode the signal so that the next signal cannot be predetermined by anything other than the receiver; and how to keep the transmitter and receiver synchronous, given that the receiver may not receive every transmission, for example when the transmitter key is pressed accidentally.

The 68HC05K1 and the 68HC05P3 are members of the low-cost high-performance 68HC05 family of 8-bit microcontroller units (MCUs). The 68HC05K1 has 504 bytes of user ROM and 32 bytes of user RAM. It also has a 64-bit array of personality EPROM/OTPROM. The 68HC05K1 has a 15-bit multifunction timer with real-time interrupt circuit. It has 10 general purpose I/O pins and is available in a 16-pin package. The 68HC05K devices are currently the lowest pin count microcontroller units produced by Motorola.

The 68HC05P3 has 3328 bytes of user ROM and 128 bytes of user RAM. It also has 128 bytes of user EEPROM. In addition to the 15-bit core timer, it has a 16-bit programmable timer with timer capture and timer compare functions. The 68HC05P3 has 22 general purpose I/O lines and is available in a 28-pin package.

TRANSMITTER SPECIFICATIONS

The transmitter uses the 68HC05K1 because of the small size of the device and the low power consumption. This is very important because the transmitter will be battery powered. The transmitter will have a single button. When pressed, an infrared code is transmitted. This code will be different for every key press and will only repeat after several million key presses. The Personality EPROM (PEPROM) is used to store a 24-bit code which will be different for every transmitter. Using the PEPROM identifier code and a 24-bit count value in an encoding algorithm a transmission code is produced. As the count value increments for every key press the resultant output from the algorithm will change for every keypress. The 24-bit output is inserted into a transmission message which is transmitted through one of the I/O ports to an amplifier stage driving an IR diode.

A flow diagram detailing the operation of the transmitter is shown as figure 1. After the key is released the microcontroller enters STOP mode, which stops the oscillator and puts the device in its lowest power mode.

A learn routine is built into the transmitter to allow the receiver to recognise the transmitter and to allow for resynchronisation of the transmitter and receiver. If the key is pressed an encrypted code is transmitted. If the key is held down for five seconds then the identity code for the transmitter is inserted into the transmission message and transmitted eight times. The beginning of the transmission message is modified so that it is recognised as an identity code and not an encrypted code. The code is sent eight times so that the receiver can compare eight received signals which should be identical so any bit errors will show up before the code is programmed into the receiver.

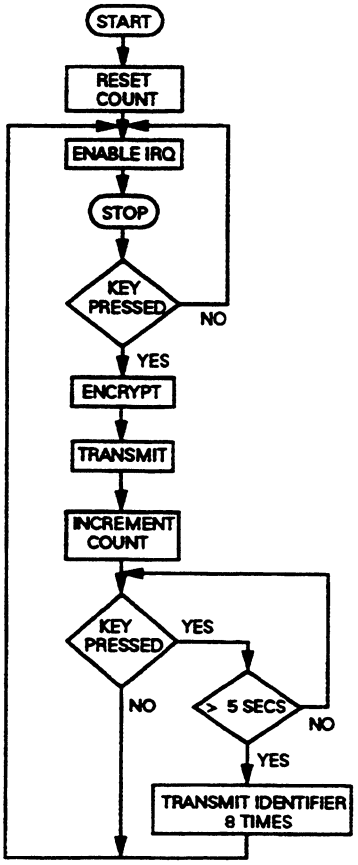


Figure 1 Flow diagram for transmitter operation

TRANSMISSION PROTOCOL

The method of transmission is a form of biphase pulse code modulation. A transmitted '1' consists of a 32kHz pulse train for 512µs followed by a 512µs pause. A transmitted '0' consists of a 512µs pause followed by a 32kHz pulse train. This gives a bit time of 1024µs for all bits. This is shown in figure 2.

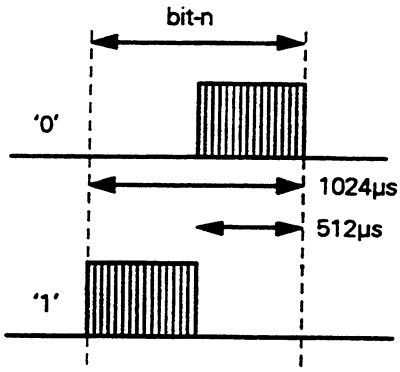
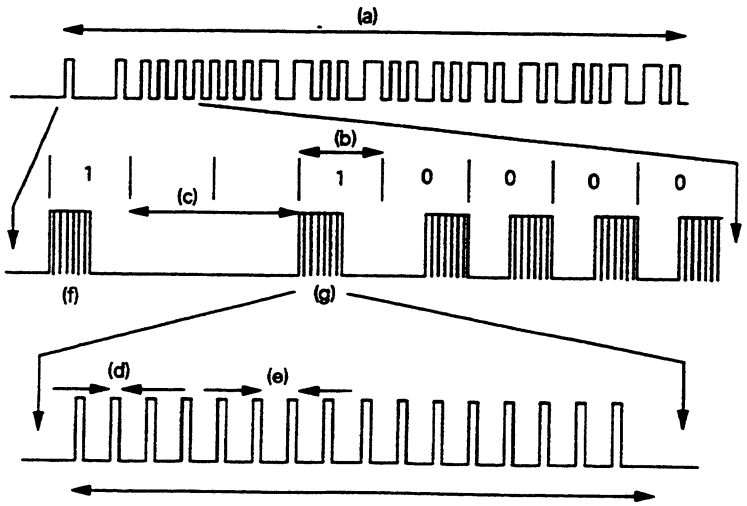


Figure 2 Transmission scheme

A complete transmission consists of a pre-bit, a pre-bit pause, a start bit, seven command bits and twenty-four bits for the encrypted code or identifier code. The pre-bit and start bit are always logic '1'. The pre-bit pause allows for the set-up of the automatic gain control of the receiver. The seven command bits are either all 1's if the following 24 bits are an encrypted code or all 0's if the following 24 bits are the identifier code. Figure 3 gives exact timing relationships for the transmissions. The upper waveform of figure 3 represents a complete transmission. As the seven command bits are all 0's, this transmission contains the identity code for the transmitter. The 24-bit code is sent least significant bit first. Below this is a waveform showing in more detail the first few bits transmitted. The third waveform shows the 32kHz pulse train or burst. In the 32kHz burst the mark to space ratio is 1 to 3. For every bit transmitted there is 512µs of the 32kHz burst, but if a 1 is transmitted after a 0 a burst is required for 1024µs. If a 0 is transmitted after a 1 then the time between bursts is 1024µs. This can be seen in the upper waveform.



(a) = 36.86 ms	Total transmission time	36 * bit time
(b) = 1024 µs	Bit time	
(c) = 2048 µs	Pre-bit pause	2 * bit time
(d) = 8 µs		
(e) = 32 µs	$1/f_{carrier}$	
(f) = 1024 µs	Pre-bit	
(g) = 1024 µs	Start bit	

Figure 3 Timing relationships for transmissions

ENCRYPTION

The method of encryption used is very effective but simple, thus saving processing time and code size. The algorithm used is a pseudo random generator. A flow diagram of it is shown as figure 4.

Held in RAM in the transmitter is a 24-bit count value which is incremented every time the transmitter key is pressed. If the least significant bit (LSB) of the count is 1 then the count is Exclusive-Ored with the 24-bit identifier code which is unique to every transmitter. The resultant is then barrel shifted with the LSB becoming the most significant bit (MSB). The previous bit1 becomes bit0. The algorithm loops 23 times checking the LSB and Exclusive-Oring the resultant of the previous loop with the identifier each time the LSB is 1. This method has the advantage that although the original count value may change by only 1 bit on successive presses of the key, the dependence of the Exclusive-Oring on the LSB ensures that the successive output values from the encryption algorithm are not similar.

On applying power to the transmitter the identifier is copied to RAM to become the initial count value. As 24 bits are used the number of different codes which can be produced is greater than 16 million. This will make the probability of someone guessing the code very low. There is no reason why this number cannot be extended to the full 64 bits of the PEPROM.

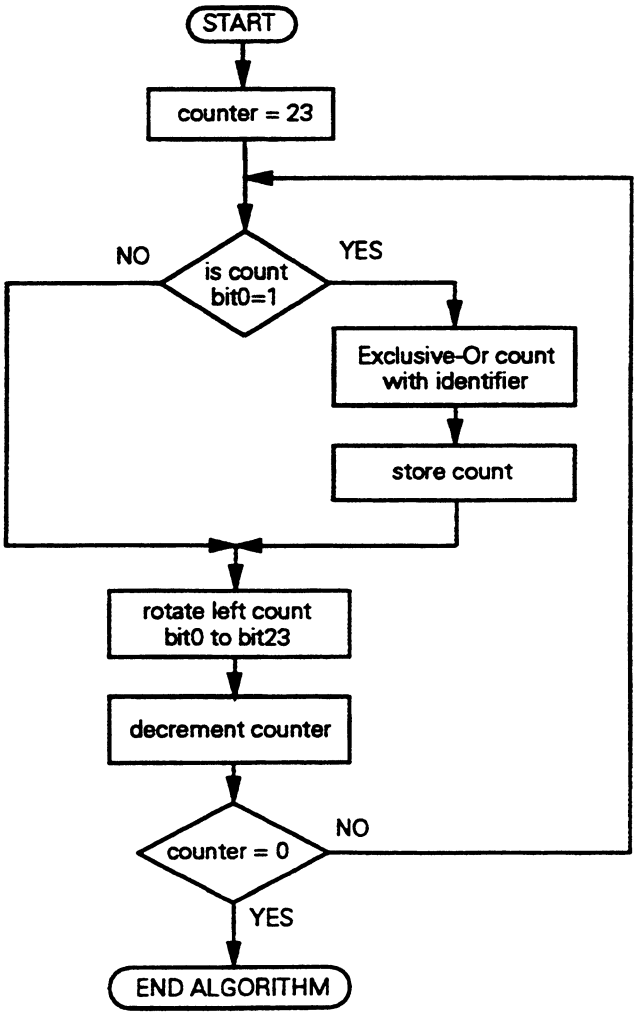


Figure 4 Encryption algorithm

RECEIVER SPECIFICATIONS

The receiver will receive and store 32 bits for every successful reception. The first bit received is the start bit which is always a 1. These bits are stored as 4 bytes with the first byte being the command instruction and the other 24 bits being either an encrypted code or the identifier of the transmitter. If the command instruction is all 1's, then the other 3 bytes are taken as an encrypted code. Otherwise, they are taken as an identity code. A flow diagram of the receiver is shown as figure 5.

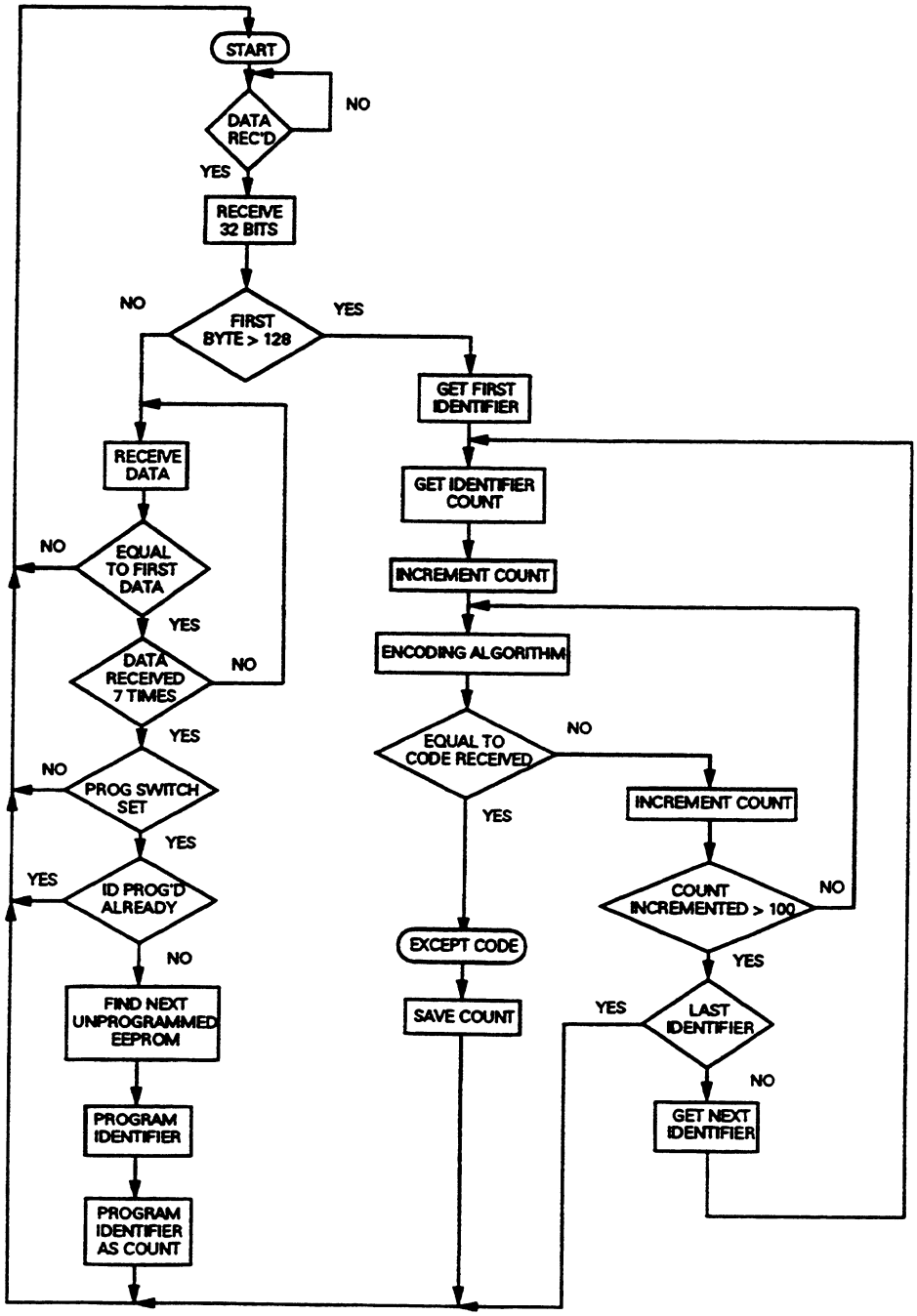


Figure 5 Flow diagram for receiver operation

If the code received is the identifier then the receiver will then wait and receive seven further identical signals before continuing. Each received signal is compared with the first to check for any bit errors. A check is made of the EEPROM to see if that identifier code has been programmed already. If it has not, then the code is programmed into EEPROM as a new transmitter. The code also becomes the initial count value of that transmitter identifier and is also stored in EEPROM. A count value also stored in EEPROM is incremented which contains the number of identifiers programmed already. Each receiver can accommodate up to 21 transmitters. If the identifier code is found to be already present then the count value for that identifier is reset to its original value, this being the value of the identifier. This is used to resynchronise transmitter and receiver. Before the identifier is programmed a switch on the receiver has to be pressed no more than ten seconds after the transmitter sends the eight identifier signals. A beeper will sound to signal that the identifiers have been received and will continue for up to 10 seconds or until the program enable switch is pressed.

If the code received is an encrypted code then the first identifier code in EEPROM and its count value are used in the same algorithm as the transmitter. If the resultant is identical to the code received then this is taken as a successful transmission. If it is not identical then the count value is incremented and the process is repeated. This continues for up to 100 increments of the count. This allows for accidental presses of the transmitter key. If a match is not found then the next identifier in EEPROM is tested in the same way. If no matches are found then the transmission is taken as illegal. If a successful match is found then the value of the count value which the algorithm used to produce the same code becomes the new count value of that transmitter. The transmitter and receiver have then identical count values. They are then resynchronised every time a successful transmission is made. A beeper will sound if the transmission was successful.

SYSTEM HARDWARE

Figure 6 shows the circuit for the transmitter. Very few components are required to build it which reduces the cost and size of the transmitter. Only one small signal transistor is used for the infrared transmitter stage. No start-up resistor is required between OSC1 and OSC2 as a 2MΩ resistance is connected internally. A 2MHz oscillator is used as this would allow for lower operating voltage than with a 4MHz crystal.

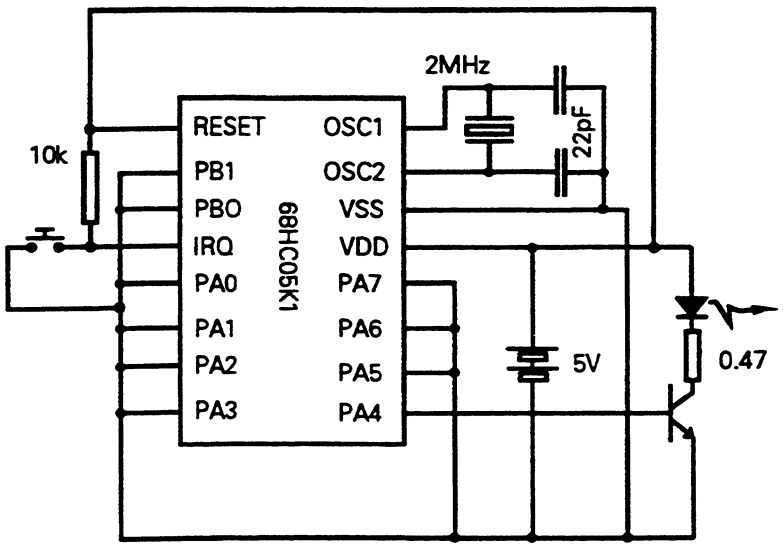


Figure 6 Transmitter circuit

The circuit diagram for the receiver is shown as figure 7. The IR receiver uses the TDA3048 but there are other devices which perform the same function. It is externally tuned to the 32kHz of the transmitted code. The output is converted to a 0 and 5V level and inverted with the 32kHz modulation removed for reception by the 68HC05P3. The 68HC05P3 receives the signal on the TCAP pin. Apart from the oscillator the only other circuitry is the switch to enable programming of the EEPROM after the identifier is received and the beeper. The beeper used here is controlled by supplying pulses to it. This application only concerns the communication system. It is intended that the 68HC05P3 could be used to form part of an alarm or secure entry system, as the receiver software only uses a fraction of the available memory.

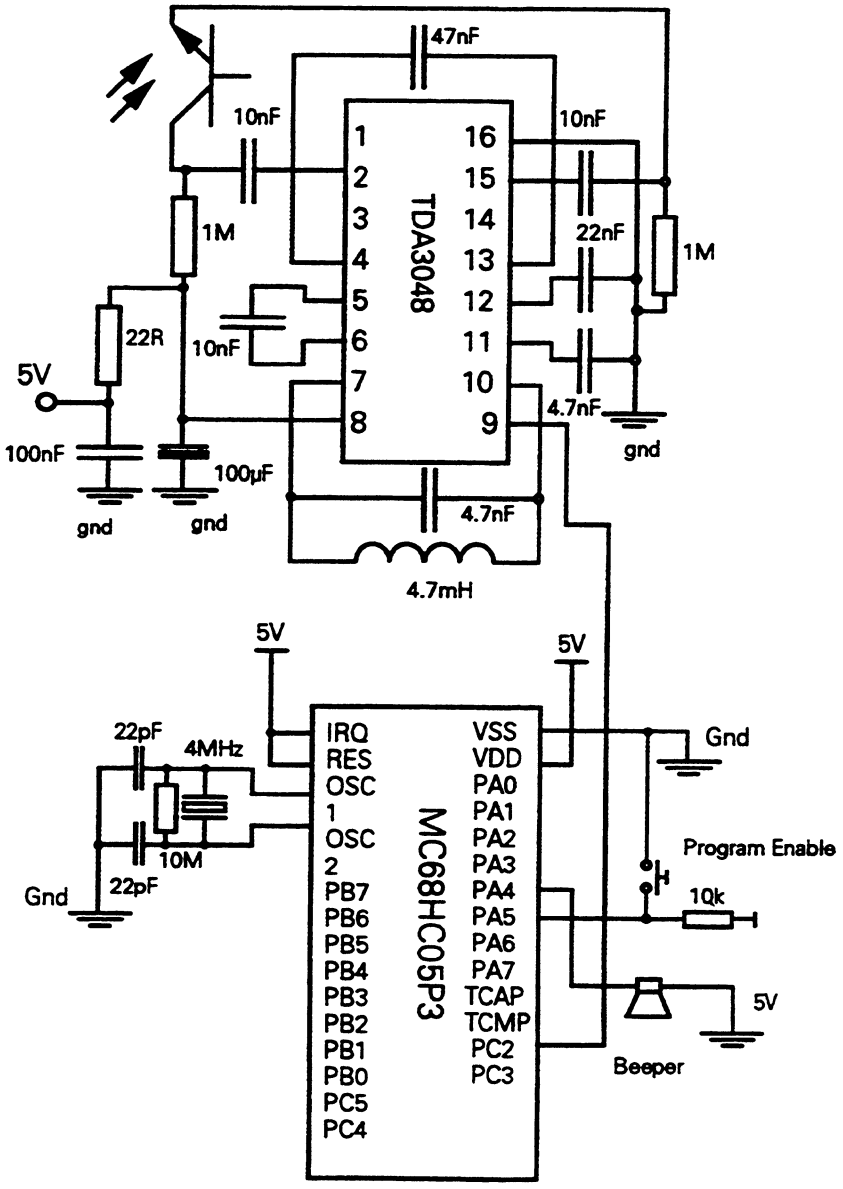


Figure 7 Receiver circuit

SYSTEM SOFTWARE DESCRIPTION

If the key is pressed on the transmitter a single transmission is made containing an encrypted code and instruction to the receiver to tell it is an encrypted code. If this code is received correctly the receiver will make a beep sound. If the code is decoded correctly by the receiver then the transmitter will produce another beep sound. If the key is held down for approximately 5 seconds then the identifier code is transmitted. If the code is received correctly by the receiver but that identifier is already programmed then it will signify this by another beeping sound. If that identifier is not already programmed then the receiver will beep for approximately 10 seconds until the user presses the program enable switch before the EEPROM is programmed with the code. If the key is not pressed then the identifier is not programmed and the receiver will not successively receive encrypted codes until the learn routine is executed again.

TRANSMITTER SOFTWARE

An assembled listing of the transmitter software can be found at the end of this application note. On power being applied to the transmitter the identifier is copied from the PEPROM into RAM. The count value also stored in RAM is set to the value of the identifier. After enabling the IRQ line the 68HC05K1 enters STOP mode. If the key is pressed an interrupt is generated which will awaken the device. The microcontroller then encrypts the identifier and transmits it.

The encryption algorithm subroutine 'algo' sets up a count of 24 for the number of encryption cycles it goes through. The 3 byte count value is then barrel shifted with the LSB becoming the MSB via the carry. If the carry is cleared, that is the LSB was a 0 then the count is not Exclusive-Ored with the identifier. This process is repeated with the count being decremented each time until it is zero.

The transmission subroutine 'trans' transmits the pre-bit and the start bit and then four bytes of data. The first byte is the instruction command followed by three bytes of code. The first of these three bytes is set to all 1's to inform the receiver that the next 3 bytes are an encrypted code and all 0's for an identifier code. Two subroutines are used to produce the 32kHz burst and the 512µs pause. After every bit transmission the four bytes are rotated so that the LSB of the four bytes is the last bit transmitted, Bit1 is the current bit being transmitted and Bit2 is the next bit to be transmitted. This arrangement is required because if a 1 follows a 0 then the 32kHz burst is required for 2 half bit periods (1024µs). To avoid breaks in the 32kHz burst the next bit to be transmitted has to be checked to see if the burst is to be for 1 or 2 half bit periods. If this is done then a check has to be made of the last bit transmitted to see if a double burst was transmitted. In this case only the 512µs pause subroutine has to be called as the 32kHz burst was produced during the transmission of the previous bit.

After every transmission the 24-bit count value is incremented so that a different code will be produced by the encryption algorithm for every key press. A check is then made to see if the key is held down for approximately five seconds to either enter the learning routines or return to STOP mode to await the next key press.

The learn routine 'learnt' sets the first of the 4 bytes transmitted to all 0's to inform the receiver that the next 3 bytes received is the identifier code of the transmitted. This code is transmitted eight times with the pre and start bits as normal. After this routine the program returns to the very start of the code so resetting the count value to the value of the identifier.

RECEIVER SOFTWARE

The receiver software uses the timer capture function of the 68HC05P3. The interrupt routine 'timer' is executed every time there is a falling edge on the TCAP pin. Using this function the number of half bit periods between successive falling edges can be calculated.

The first check made is to look for 6 half bit periods. This occurs between the pre and start bits. If this occurs then the system is set to receive the next 32 data bits. If the time between successive edges is 2 half bit periods the data bit received is the same as the previous bit received. If the time is 3 half bit periods the data is a 0 if the last bit was a 1. If the last bit was a 0 then two 1's have been received. If the time is 4 half bit periods then the data received was a 1 followed by a 0 if the last bit received was a 0. If the last bit received was a 1 then a 4-bit period gap is illegal. After each bit is received they are rotated onto 4 bytes. After all 32 bits have been received, the 4 bytes contain the instruction command and the data code.

After a complete transmission is received the command byte is checked to proceed in the decoding routines or the learning routines.

In the decoding routines 'decode', a check is made of the count value stored in EEPROM which holds the number of identifiers programmed into the EEPROM. The first identifier is taken and used with its corresponding count value which is also stored in EEPROM in the same algorithm routine as the transmitter. If the output of the algorithm is the same as the data received the transmission is taken as successful. If the output is different then the count in incremented and tried again in the algorithm to see if a match can be found. The count is incremented by up to 100 times to look for a match. If a match is found then the value of the count at which the match was found replaces the original count value for that identifier in EEPROM. If no matches are found after 100 increments then the next identifier is tested in the same way. The transmission is a failure if all identifiers are tried and no matches found. A successful transmission is signified to the user by a sound from the beeper.

In the learn routines the transmitter waits to receive the other 7 transmissions comparing each one with the original signal to check for bit errors. If all transmissions are identical the identifier count in EEPROM is first checked to see if there is room in EEPROM for the new transmitter identifier. Up to 21 identifiers can be programmed. The identifier is then compared with each programmed identifier in turn to see if that identifier is already programmed. If the identifier has not been programmed already the beeper will then sound and a check is made of the enable programming switch to see if it is pressed in the next 10 seconds before the identifier is programmed into the EEPROM in the next free space. An initial count value is also programmed and has the same value as the identifier. If the identifier is found to be already programmed the count value for that identifier is reset to the initial value. This is used to resynchronise with the transmitter.

The software used by both the receiver and transmitter is very modular with general purpose routines for erasing and programming EEPROM, moving data blocks in memory and rotating blocks of data.

Code for transmitter

```
0001                    ********************************************************************************
0002                    * Secure Remote Control IR Transmitter Using 68HC05K1                          *
0003                    * A Breslin                                                                    *
0004                    * MCU Applications                                                             *
0005                    * 24.6.92.                                                                     *
0006                    ********************************************************************************
0007
0008                    ********************************************************************************
0009                    *                                                                              *
0010                    * Motorola reserves the right to make changes without further notice to any     *
0011                    * products herein to improve reliability, function or design. Motorola does    *
0012                    * not assume any liability arising out of the application or any use of any     *
0013                    * product or circuit described herein  neither does it convey any license      *
0014                    * under its patent rights nor the rights of others. Motorola products are      *
0015                    * not designed ,intended, or authorised for use as components in systems       *
0016                    * intended for surgical implant into the body, or other applications           *
0017                    * intended to support or sustain life, or for any other application in which   *
0018                    * the failure of the Motorola product could create a situation where personal  *
0019                    * injury or death may occur. Should Buyer purchase or use Motorola products    *
0020                    * for any such unintended or unauthorised application, Buyer shall indemnify   *
0021                    * and hold Motorola and its officers,employees,subsidiaries,affiliates,and     *
0022                    * distributers harmless against all claims,costs,damages,and expenses,and      *
0023                    * reasonable attorney fees arising out of,directly or indirectly, any claim    *
0024                    * to peronal injury or death associated with such unintended or unauthorised   *
0025                    * use, even if such claim alleges that Motorola was negligent regarding the    *
0026                    * design or manufacture of the part. Motorla and "Motorla sign" areregisterd   *
0027                    * trademarks of Motorola,Inc. Motorola Inc. is an Equal                        *
0028                    * Oppurtunity/Affirmative Employer.                                            *
0029                    *                                                                              *
0030                    ********************************************************************************
0031
0032                    ********************************************************************************
0033                    * This software is used to transmit a encrypted code on receipt of a key       *
0034                    * press. Each transmitter must have a unique identifier code stored in the*
0035                    * first 24 bits of the PEPROM. This identifier is used with a count value *
0036                    * in a encrytion algorithm to produce a different code evry time the key       *
0037                    * is pressed. Every time the key is pressed the count is incremented.          *
0038                    ********************************************************************************
0039
0040 0000               porta   equ     0
0041 0001               portb   equ     1
0042 0004               ddra    equ     4
0043 0005               ddrb    equ     5
0044 0008               tscr    equ     8
0045 0009               tcntr   equ     9
0046 000a               irqcr   equ     $0a             ; irq register
0047 000e               pebsr   equ     $0e             ; PEPROM bit select register
0048 000f               pescr   equ     $0f             ; PEPROM control register
0049
0050                    ********************************************************************************
0051                    *  This section reserves RAM for the count, peprom identifier code and         *
0052                    *  four location for storing the data to be transmitted                        *
0053                    ********************************************************************************
0054
0055 00e0                       org     $e0
0056
0057
0058 00e0               count   rmb     3               ; three byte count
0059 00e3               comdat  rmb     4               ; four bytes for transmission
0060 00e7               pepid   rmb     3               ; three bytes for identifier
0061 00ea               datsto  rmb     3               ; three bytes for gp store
0062 00ed               gpcnt   rmb     1               ; one general purpose store
0063 00ee               gp2     rmb     1               ; one general purpose store
0064 0200                       org     $200
0065
```

```
0066                    ********************************************************************************
0067                    * This is the main section for transmission. The value of the identifier       *
0068                    * stored in PEPROM is loaded into the count store. This is done when the        *
0069                    * is reset and is used to resynchronise the transmitter and receiver           *
0070                    ********************************************************************************
0071
0072
0073 0200 9c            start   rsp                     ; reset stack
0074 0201 cd 02 b3              jsr     repep           ; read identifier from peprom
0075 0204 a6 e7                 lda     #pepid          ; point to start
0076 0206 ae e0                 ldx     #count          ; point to destination
0077 0208 cd 02 a3              jsr     mvdat           ; move data
0078 020b 3f 05         nxtkey  clr     ddrb
0079 020d 1e 0a                 bset    7,irqcr         ; enable irq
0080 020f 9a                    cli                     ; clear interrupts
0081 0210 8e                    stop                    ; low-power until key-press
0082 0211 20 f8                 bra     nxtkey
0083
0084
0085                    ********************************************************************************
0086                    * This section is called when a key is pressed. The identifier is read         *
0087                    * and combined with the current count value in the encrypting algorithm.       *
0088                    * The resultant code is transmitted. If the key is held down for more          *
0089                    * five seconds the learn routine is enterred. Otherwise STOP mode is           *
0090                    * re-entered.                                                                  *
0091                    ********************************************************************************
0092
0093 0213 9c            go      rsp                     ; reset stack pointer
0094 0214 a6 ff                 lda     #$ff            ; code sent for noraml command
0095 0216 b7 e3                 sta     comdat          ; sent as first byte
0096 0218 cd 02 c3              jsr     algo            ; encrypt code
0097 021b ad 38                 bsr     trans           ; transmit codes
0098 021d a6 06                 lda     #$06            ; timeout if key held down
0099 021f b7 ed                 sta     gpcnt           ;
0100 0221 a6 ff         wait    lda     #$ff            ;
0101 0223 2f 2b         loopa   bih     finish          ; check if key released
0102 0225 ae ff                 ldx     #$ff
0103 0227 ad 65                 bsr     delay
0104 0229 4a                    deca                    ;
0105 022a 26 f7                 bne     loopa           ;
0106 022c 3a ed                 dec     gpcnt           ;
0107 022e 26 f1                 bne     wait            ;
0108
0109                    ********************************************************************************
0110                    * This routine is used for teaching the receiver the identifier number.        *
0111                    * The identifier is sent eight times to make sure the code is learned.         *
0112                    ********************************************************************************
0113
0114 0230 cd 02 b3      learnt  jsr     repep           ; read identifier
0115 0233 a6 08                 lda     #$08            ; send ident
0116 0235 b7 ee                 sta     gp2             ; eight times
0117 0237 3f e3         send8   clr     comdat          ; learn command
0118 0239 a6 e7                 lda     #pepid          ; send identifier
0119 023b ae e4                 ldx     #comdat+1
0120 023d ad 64                 bsr     mvdat
0121 023f b6 e7                 lda     pepid
0122 0241 ad 12                 bsr     trans           ; one transmission
0123 0243 a6 08                 lda     #$08            ; every 40ms
0124 0245 ad 45         fourm   bsr     pause
0125 0247 4a                    deca
0126 0248 26 fb                 bne     fourm
0127 024a 3a ee                 dec     gp2
0128 024c 26 e9                 bne     send8
0129 024e 20 b0                 bra     start           ; effectively reset
0130 0250 cd 02 eb      finish  jsr     incnt           ; increment counter
0131 0253 20 b6                 bra     nxtkey          ; else next key
0132
```

```
0133                    ********************************************************************************
0134                    * This is the transmission routine. It transmits the four bytes stored         *
0135                    * in comdat. The bit transmitted is in bit1 of comdat, the previous bit        *
0136                    * is bit0 and the next bit is bit2. After each bit transmission the 32         *
0137                    * bits in comdat are rotated. The present bit becomes the last bit and        *
0138                    * the next bit becomes the present bit for transmission                       *
0139                    ********************************************************************************
0140
0141 0255 a6 10          trans   lda     #$10            ; porta bit 4 output
0142 0257 b7 04                 sta     ddra
0143 0259 ae 10                 ldx     #$10
0144 025b ad 37                 bsr     burst           ; pre-bit
0145 025d ae f5                 ldx     #$f5            ; pre-bit pause
0146 025f ad 2d                 bsr     delay
0147 0261 ae 10                 ldx     #$10
0148 0263 ad 2f                 bsr     burst           ; start bit
0149 0265 ad 25                 bsr     pause
0150 0267 a6 1f                 lda     #$1f            ; send thirty one bits
0151 0269 02 e3 0f      sendat  brset   1,comdat,send1  ; test if bit = 1
0152 026c ad 1e                 bsr     pause           ; else transmit = 0
0153 026e 04 e3 04              brset   2,comdat,send01 ; test if next bit = 1
0154 0271 ae 10                 ldx     #$10            ; else send code for 00
0155 0273 20 02                 bra     send0           ;  burst for 0.5 bit period
0156 0275 ae 20         send01  ldx     #$20            ;  burst for 1 bit period
0157 0277 ad 1b         send0   bsr     burst           ; burst routine for modulation
0158 0279 20 09                 bra     endtrn          ; end of bit transmission
0159 027b 01 e3 04      send1   brclr   0,comdat,send10 ; test if double burst transmitted
0160 027e ae 10                 ldx     #$10            ; else send single burst
0161 0280 ad 12                 bsr     burst           ; burst routine for modulation
0162 0282 ad 08         send10  bsr     pause           ; pause for biphase
0163 0284 ae e3         endtrn  ldx     #comdat         ; rotate all thirty two bits
0164 0286 ad 23                 bsr     rotate          ; to transmit next bit
0165 0288 4a                    deca                    ; check if all bits sent
0166 0289 26 de                 bne     sendat          ; else send next bit
0167 028b 81                    rts
0168
0169                    ********************************************************************************
0170                    * This routine is called by the transmission routine to provide the 512us      *
0171                    * pause in the biphase coding.                                                 *
0172                    ********************************************************************************
0173
0174 028c ae 31          pause   ldx     #$31
0175 028e 9d             delay   nop
0176 028f 9d                     nop
0177 0290 5a                     decx
0178 0291 26 fb                  bne     delay
0179 0293 81                     rts
0180
0181                    ********************************************************************************
0182                    * This routine is called by the transmission routine to provide the burst      *
0183                    * of modulation. It produces a 32kHz pulse train with a mark to space          *
0184                    * ratio of 1 to 3.                                                            *
0185                    ********************************************************************************
0186
0187 0294 19 00          burst   bclr    4,porta         ; clear port
0188 0296 21 fe                  brn     *               ; delay
0189 0298 18 00                  bset    4,porta         ; set port
0190 029a 21 fe                  brn     *               ; delay to give 8us
0191 029c 19 00                  bclr    4,porta         ; clear port
0192 029e 9d                     nop                     ; delay
0193 029f 5a                     decx                    ; burst for count
0194 02a0 26 f2                  bne     burst           ;
0195 02a2 81             endbur  rts
0196
0197                    ********************************************************************************
0198                    * This is a general purpose routine for moving data pointed to by the          *
0199                    * X reg to the location pointed to by the accumulator. Three bytes are         *
0200                    * moved.                                                                       *
0201                    ********************************************************************************
0202
0203 02a3 bf ed          mvdat   stx     gpcnt           ; save destination
0204 02a5 83                     swi                     ; move first byte
0205 02a6 4c                     inca                    ; point to second byte
0206 02a7 83                     swi                     ; move second byte
0207 02a8 4c                     inca                    ; point to third byte
0208 02a9 83                     swi                     ; move third byte
0209 02aa 81                     rts
```

```
0210                    ********************************************************************************
0211                    * This is a general purpose rotate routine used for transmitting data or *
0212                    * reading the peprom. It can be entered to rotate 4 or 3 bytes. The      *
0213                    * bits are rotated one place. The 8th bit of the 4th byte is not loaded  *
0214                    * with the carry.                                                        *
0215                    ********************************************************************************
0216
0217 02ab 64 03         rotate  lsr     3,x             ; rotate right 4
0218 02ad 66 02         rot3    ror     2,x             ; or 3 bytes
0219 02af 66 01                 ror     1,x             ; pointed to by X reg
0220 02b1 76                    ror     0,x             ; lowest bit into carry
0221 02b2 81                    rts
0222
0223
0224                    ********************************************************************************
0225                    * This routine reads the identifier from the PEPROM.                     *
0226                    ********************************************************************************
0227
0228 02b3 3f 0e         repep   clr     pebsr           ; point to bottom of peprom
0229 02b5 a6 18                 lda     #$18            ; counter for 24 bits
0230 02b7 38 0f         nextp   lsl     pescr           ; bit by bit
0231 02b9 ae e7                 ldx     #pepid          ; store in RAM
0232 02bb ad f0                 bsr     rot3            ; rotate bits 3 RAM bytes
0233 02bd 3c 0e                 inc     pebsr           ; while counter != 0
0234 02bf 4a                    deca                    ; read next bit
0235 02c0 26 f5                 bne     nextp
0236 02c2 81                    rts
0237
0238
0239                    ********************************************************************************
0240                    * This routine encrypts the current count and the identifier into a code *
0241                    * for transmission. The count is rotated and is Exclusive-Ored with the  *
0242                    * the identifier if the LSB is logic 1. The process is repeated for all  *
0243                    * 24 bits.                                                               *
0244                    ********************************************************************************
0245
0246 02c3 a6 e0         algo    lda     #count          ; point to start
0247 02c5 ae e4                 ldx     #comdat+1       ; point to destination
0248 02c7 ad da                 bsr     mvdat           ; move data
0249 02c9 ae 20                 ldx     #$20            ; 32 bytes
0250 02cb 9f            algbit  txa
0251 02cc ae e4                 ldx     #comdat+1       ; point to first byte
0252 02ce ad dd                 bsr     rot3            ; rotate
0253 02d0 97                    tax
0254 02d1 24 14                 bcc     noxor           ; test if first byte=0
0255 02d3 1e e6                 bset    7,comdat+3      ; else exclusive-Or
0256 02d5 b6 e7                 lda     pepid           ; with counter
0257 02d7 b8 e4                 eor     comdat+1
0258 02d9 b7 e4                 sta     comdat+1
0259 02db b6 e8                 lda     pepid+1
0260 02dd b8 e5                 eor     comdat+2
0261 02df b7 e5                 sta     comdat+2
0262 02e1 b6 e9                 lda     pepid+2
0263 02e3 b8 e6                 eor     comdat+3
0264 02e5 b7 e6                 sta     comdat+3
0265 02e7 5a            noxor   decx                    ; if not all bits rotated
0266 02e8 26 e1                 bne     algbit          ; test next bit
0267 02ea 81                    rts
0268
```

```
0269                    ********************************************************************************
0270                    * This routine increments the count value after each key press                 *
0271                    ********************************************************************************
0272
0273 02eb 3c e0        incnt   inc     count           ; inc low byte
0274 02ed 26 0a                 bne     nover           ; if overflow
0275 02ef 3c e1                 inc     count+1         ; inc mid byte
0276 02f1 26 06                 bne     nover           ; if overflow
0277 02f3 3c e2                 inc     count+2         ; inc high byte
0278 02f5 26 02                 bne     nover           ; if overflow
0279 02f7 3c e0                 inc     count           ; inc low byte
0280 02f9 81            nover   rts
0281
0282
0283 02fa 97            soft    tax                     ; pointer for source
0284 02fb f6                    lda     ,x              ; load source byte
0285 02fc be ed                 ldx     gpcnt           ; point to destination
0286 02fe f7                    sta     ,x              ; move byte
0287 02ff 3c ed                 inc     gpcnt           ; point to next byte
0288 0301 80                    rti
0289
0290 03f8                       org     $3f8
0291
0292 03f8 02 00                 fdb     start           ; timer int
0293 03fa 02 13                 fdb     go              ; irq
0294 03fc 02 fa                 fdb     soft            ; swi
0295 03fe 02 00                 fdb     start           ; reset
0296
0297                            end
0298
```

Code for receiver

```
0001
0002                    ********************************************************************************
0003                    * Secure remote control receiver.
0004                    * A Breslin
0005                    * MCU Applications EKB
0006                    * 5/10/92
0007                    ********************************************************************************
0008                    ********************************************************************************
0009                    * This program receives and decodes the receiver IR transmissions of the *
0010                    * 68HC05K1 secure remote control transmitter. The data is received on the*
0011                    * TCAP pin. The data received is compared with all the possible expected *
0012                    * data transmissions to decide if the data received is valid.            *
0013                    *    The system allows for up to 21 different users with each user having*
0014                    * a unique identifier which is used in the encrypting algorithm. The     *
0015                    * system can learn new identifiers.                                      *
0016                    ********************************************************************************
0017
0018 00f0               tol     equ     240             ; tolerance 120us
0019 0100               eeprom0 equ     $100            ; eeprom location 0
0020 0040               eepcnt  equ     64              ; eeprom count store offset
0021 0080               ram     equ     $80             ; RAM location
0022 0300               rom     equ     $300            : ROM location
0023 0ff6               vector  equ     $ff6            ; vector table
0024
0025 0000               porta   equ     0
0026 0001               portb   equ     1
0027 0002               portc   equ     2
0028
0029 0004               ddra    equ     4
0030 0005               ddrb    equ     5
0031 0006               ddrc    equ     6
0032 000a               tim16   equ     $0a
0033 001c               prog    equ     $1c               ; eeprom programming reg
0034 0012               tcr     equ     $12
0035 0013               tsr     equ     $13
0036 0014               icrh    equ     $14
0037 0015               icrl    equ     $15
0038 0019               timcl   equ     $19
0039 001d               icrl2   equ     $1d
0040                    ********************************************************************************
0041                    *  This section reserves RAM for the count, peprom identifier code and     *
0042                    *  four location for storing the data to be transmitted                    *
0043                    ********************************************************************************
0044
0045 0080                       org     ram
0046
0047
0048 0080               count   rmb     3               ; three byte count
0049 0083               comdat  rmb     4               ; four bytes for transmission
0050 0087               pepid   rmb     3               ; three bytes for identifier
0051 008a               datsto  rmb     3               ; three bytes for gp store
0052 008d               gpcnt   rmb     1               ; one general purpose store
0053 008e               datin   rmb     1               ; data rec flags
0054 008f               diffl   rmb     1               ; last time (low byte)
0055 0090               diffh   rmb     1               ; last time (high byte)
0056 0091               irra1   rmb     1
0057 0092               irra2   rmb     1
0058 0093               irra3   rmb     1
0059 0094               irra4   rmb     1
0060 0095               stat    rmb     1
0061 0096               tcnt    rmb     1
0062 0097               idcnt   rmb     1
0063 0098               gpcnt2  rmb     1
0064 0099               gpcnt3  rmb     1
0065
0066 0300                       org     rom
0067
```

```
0068                    ********************************************************************************
0069                    * This is the receiver routine. On receipt of a transmission a check is       *
0070                    * made to see if the received data is a learn command or an encoded           *
0071                    * data. A learn command instructs the receiver to learn the received          *
0072                    * identifier. A comparison is made between the encoded data and the data      *
0073                    * expected to decide if the data is excepted.                                 *
0074                    ********************************************************************************
0075
0076 0300 a6 7f       start   lda     #$7f
0077 0302 b7 00             sta     porta
0078 0304 3f 04             clr     ddra
0079 0306 9c                rsp
0080 0307 cd 04 ee          jsr     rec             ; receive data
0081 030a 9c                rsp
0082 030b a6 84             lda     #comdat+1       ; point to start
0083 030d ae 8a             ldx     #datsto         ; point to destination
0084 030f cd 04 a7          jsr     mvdat           ; move data
0085 0312 b6 83             lda     comdat          ; load command
0086 0314 a1 81             cmp     #$81            ; check if learn
0087 0316 25 03             blo     learnr          ; branch if decode
0088 0318 cc 04 20          jmp     decode
0089 031b a6 06       learnr  lda     #$06            ; store count
0090 031d b7 98             sta     gpcnt2
0091 031f cd 04 ee          jsr     rec
0092 0322 a6 84             lda     #comdat+1       ; point to start
0093 0324 ae 8a             ldx     #datsto         ; point to destination
0094 0326 cd 04 a7          jsr     mvdat           ; move data
0095 0329 cd 04 ee    nxtrec  jsr     rec             ; load next codes
0096 032c cd 05 09          jsr     cmpdat          ; compare data received
0097 032f 27 03             beq     datcmp
0098 0331 cc 05 a8          jmp     badend          ; end if different
0099 0334 3a 98       datcmp  dec     gpcnt2          ; count rec code
0100 0336 26 f1             bne     nxtrec
0101
0102
0103
0104                    ********************************************************************************
0105                    * At this stage eight identical transmissions of the identifier have been*
0106                    * received. A search is made to see if the identifier has been prog'd     *
0107                    * already and to find the next unprogrammed bytes                         *
0108                    ********************************************************************************
0109
0110 0338 cd 05 d5     save    jsr     buzz
0111 033b 5f                clrx
0112 033c 5c                incx
0113 033d c6 01 00          lda     eeprom0
0114 0340 a1 ff             cmp     #$ff            ; check if clear
0115 0342 26 05             bne     notclr
0116 0344 5f                clrx                    ; no offset
0117 0345 4f                clra                    ; clear counter
0118 0346 cd 04 06          jsr     eeprog          ; program 00
0119 0349 5f          notclr  clrx
0120 034a c6 01 00          lda     eeprom0         ; load count
0121 034d 27 33             beq     enable
0122 034f a1 15             cmp     #21             ; maximum 21 locations
0123 0351 25 06             blo     space           ; space left
0124 0353 cd 03 e0          jsr     erase
0125 0356 cc 05 a8          jmp     badend          ; ram full
0126 0359 b7 8d       space   sta     gpcnt           ; number of ids prog'd
0127 035b b6 8a       eeptst  lda     datsto
0128 035d d1 01 01          cmp     eeprom0+1,x
0129 0360 26 19             bne     diffid
0130 0362 b6 8b             lda     datsto+1
0131 0364 d1 01 02          cmp     eeprom0+2,x
0132 0367 26 12             bne     diffid
0133 0369 b6 8c             lda     datsto+2
0134 036b d1 01 03          cmp     eeprom0+3,x
0135 036e 26 0b             bne     diffid
0136 0370 5c                incx
0137 0371 5c                incx
0138 0372 5c                incx
0139 0373 ad 51             bsr     stocnt
0140 0375 cd 05 ad          jsr     wend            ; delay
0141 0378 cc 03 00          jmp     start           ; identifier prog'd already
0142
0143 037b 5c          diffid  incx                    ; inc for first byte offset
0144 037c 5c                incx                    ; inc for second byte offset
0145 037d 5c                incx                    ; inc for third byte offset
0146 037e 3a 8d             dec     gpcnt           ; test if more ids programmed
0147 0380 26 d9             bne     eeptst          ; search if
```

```
0148                    ********************************************************************************
0149                    * The next three bytes are unprogrammed. They are first erased and then        *
0150                    * programmed.                                                                  *
0151                    ********************************************************************************
0152 0382 19 04         enable  bclr    4,ddra          ; PA4 input
0153 0384 a6 18                 lda     #$18            ; ten seconds
0154 0386 b7 98                 sta     gpcnt2          ; to press switch
0155 0388 4f                    clra                    ; or code is not
0156 0389 3f 8d                 clr     gpcnt           ; programmed
0157 038b 09 00 11      eloop   brclr   4,porta,progb
0158 038e 4a                    deca
0159 038f 26 fa                 bne     eloop
0160 0391 3a 8d                 dec     gpcnt
0161 0393 26 f6                 bne     eloop
0162 0395 cd 05 d5              jsr     buzz
0163 0398 3a 98                 dec     gpcnt2
0164 039a 26 ef                 bne     eloop
0165 039c cc 03 00              jmp     start
0166 039f 5c            progb   incx                    ; offset for next byte
0167 03a0 ad 3e                 bsr     erase           ; erase byte
0168 03a2 b6 8a                 lda     datsto          ; load byte
0169 03a4 ad 60                 bsr     eeprog          ; program byte
0170 03a6 5c                    incx                    ; point to next byte
0171 03a7 ad 37                 bsr     erase           ; erase byte
0172 03a9 b6 8b                 lda     datsto+1        ; load byte
0173 03ab ad 59                 bsr     eeprog          ; program byte
0174 03ad 5c                    incx                    ; point to next byte
0175 03ae ad 30                 bsr     erase           ; erase byte
0176 03b0 b6 8c                 lda     datsto+2        ; load byte
0177 03b2 ad 52                 bsr     eeprog          ; program byte
0178 03b4 ad 10                 bsr     stocnt
0179
0180 03b6 c6 01 00              lda     eeprom0         ;
0181 03b9 4c                    inca                    ; increment id count
0182 03ba b7 8d                 sta     gpcnt           ; store count
0183 03bc 5f                    clrx                    ; offset 0
0184 03bd ad 21                 bsr     erase           ; erase count
0185 03bf b6 8d                 lda     gpcnt           ; load count
0186 03c1 ad 43                 bsr     eeprog          ; program new count
0187 03c3 cc 03 00              jmp     start
0188
0189
0190                    ********************************************************************************
0191                    * The identifier becomes the initial count value                               *
0192                    ********************************************************************************
0193
0194 03c6 9f             stocnt  txa
0195 03c7 ab 3d                 add     #eepcnt-3       ; point to eeprom
0196 03c9 97                    tax                     ; counter store
0197 03ca ad 14                 bsr     erase           ; erase byte
0198 03cc b6 8a                 lda     datsto          ; load byte
0199 03ce ad 36                 bsr     eeprog          ; program byte
0200 03d0 5c                    incx                    ; point to next byte
0201 03d1 ad 0d                 bsr     erase           ; erase byte
0202 03d3 b6 8b                 lda     datsto+1        ; load byte
0203 03d5 ad 2f                 bsr     eeprog          ; program byte
0204 03d7 5c                    incx                    ; point to next byte
0205 03d8 ad 06                 bsr     erase           ; erase byte
0206 03da b6 8c                 lda     datsto+2        ; load byte
0207 03dc ad 28                 bsr     eeprog          ; program byte
0208 03de 5c                    incx                    ; point to next byte
0209 03df 81                    rts
0210
0211                    ********************************************************************************
0212                    * EEPROM erase routine                                                         *
0213                    ********************************************************************************
0214
0215 03e0 1c 1c          erase   bset    6,prog          ; turn on charge pump
0216 03e2 10 1c                 bset    0,prog
0217 03e4 14 1c                 bset    2,prog          ; enable latch bit
0218 03e6 16 1c                 bset    3,prog
0219 03e8 19 1c                 bclr    4,prog          ; byte erase
0220 03ea d7 01 00              sta     eeprom0,x       ; write to location
0221 03ed 10 1c                 bset    0,prog          ; enable power
0222 03ef ad 09                 bsr     eepdly          ; erase time delay
0223 03f1 13 1c                 bclr    1,prog          ; disable power
0224 03f3 11 1c                 bclr    0,prog          ; clear latch
0225 03f5 ad 03                 bsr     eepdly          ; wait for voltage to drop
0226 03f7 3f 1c                 clr     prog
0227 03f9 81                    rts                     ;
0228
```

```
0229                    ************************************************************************
0230                    * Routine for 15ms delay (2MHz bus) for programming.                    *
0231                    ************************************************************************
0232 03fa a6 0f         eepdly  lda     #$0f            ;
0233 03fc 3f 99                 clr     gpcnt3          ;
0234 03fe 3a 99         timlp   dec     gpcnt3          ;
0235 0400 26 fc                 bne     timlp           ;
0236 0402 4a                    deca                    ;
0237 0403 26 f9                 bne     timlp           ;
0238 0405 81                    rts
0239
0240                    ************************************************************************
0241                    * EEPROM programming routine                                           *
0242                    ************************************************************************
0243
0244 0406 1c 1c         eeprog  bset    6,prog          ; turn on charge pump
0245 0408 10 1c                 bset    0,prog
0246 040a 14 1c                 bset    2,prog          ; enable latch
0247 040c 17 1c                 bclr    3,prog
0248 040e 19 1c                 bclr    4,prog          ; byte program
0249 0410 d7 01 00              sta     eeprom0,x       ;
0250 0413 10 1c                 bset    0,prog          ; enable power
0251 0415 ad e3                 bsr     eepdly          ; program time delay
0252 0417 13 1c                 bclr    1,prog          ; clear latch
0253 0419 11 1c                 bclr    0,prog          ; disable power
0254 041b ad dd                 bsr     eepdly          ; wait for volt drop
0255 041d 3f 1c                 clr     prog
0256 041f 81                    rts
0257
0258
0259
0260                    ************************************************************************
0261                    * The section tries to find a match between the received data and the   *
0262                    * output of the encrypting algorithm. It first check to see if the      *
0263                    * received data is a                                                    *
0264                    ************************************************************************
0265
0266 0420 cd 05 d5      decode  jsr     buzz
0267 0423 c6 01 00              lda     eeprom0         ; load number of ids prog'd
0268 0426 b7 97                 sta     idcnt           ; store
0269 0428 27 25                 beq     decend          ; test if unprogrammed
0270 042a a1 14                 cmp     #$14            ; test if out of range
0271 042c 22 21                 bhi     decend
0272 042e 3f 8e                 clr     datin           ; clear offset count
0273 0430 ad 54         nxtid   bsr     eepmv           ; move id and count to RAM
0274 0432 a6 64                 lda     #$64            ; count for 100 tries
0275 0434 b7 96                 sta     tcnt
0276 0436 cd 04 b7      nxtdec  jsr     algo            ; decode
0277 0439 cd 05 09              jsr     cmpdat          ; compare data
0278 043c 27 14                 beq     open            ; except if equal
0279 043e cd 04 df              jsr     incnt           ; increase counter
0280 0441 3a 96                 dec     tcnt            ; dec counter
0281 0443 26 f1                 bne     nxtdec          ; try again
0282 0445 3c 8e                 inc     datin
0283 0447 3c 8e                 inc     datin
0284 0449 3c 8e                 inc     datin           ; offset for next identifier
0285 044b 3a 97                 dec     idcnt           ; test number of ids
0286 044d 26 e1                 bne     nxtid           ; check next id
0287 044f cc 05 a8      decend  jmp     badend          ; else end
0288
```

```
0289                    ********************************************************************************
0290                    * count value is saved in EEPROM                                               *
0291                    ********************************************************************************
0292
0293 0452 be 8e         open    ldx     datin           ; count location
0294 0454 9f                    txa
0295 0455 ab 40                 add     #eepcnt
0296 0457 97                    tax
0297 0458 ad 86                 bsr     erase           ; erase byte
0298 045a b6 80                 lda     count           ; count value
0299 045c ad a8                 bsr     eeprog          ; program byte
0300
0301 045e 5c                    incx
0302 045f cd 03 e0              jsr     erase           ; erase byte
0303 0462 b6 81                 lda     count+1         ; count value
0304 0464 cd 04 06              jsr     eeprog          ; program byte
0305 0467 5c                    incx
0306 0468 cd 03 e0              jsr     erase           ; erase byte
0307 046b b6 82                 lda     count+2         ; count value
0308 046d ad 97                 bsr     eeprog          ; program byte
0309 046f a6 06                 lda     #%00000110
0310 0471 b7 00                 sta     porta           ; light green led
0311 0473 cd 05 d5              jsr     buzz
0312 0476 a6 ff                 lda     #$ff
0313 0478 ae ff         adec    ldx     #$ff
0314 047a 5a            xdec    decx
0315 047b 26 fd                 bne     xdec
0316 047d 4a                    deca
0317 047e 26 f8                 bne     adec
0318 0480 cd 05 d5              jsr     buzz
0319 0483 cc 03 00              jmp     start
0320
0321                    ********************************************************************************
0322                    * This subroutine moves the count and identifiers into RAM                     *
0323                    ********************************************************************************
0324
0325 0486 be 8e         eepmv   ldx     datin                   ; load offset
0326 0488 d6 01 01              lda     eeprom0+1,x             ; load three
0327 048b b7 87                 sta     pepid                   ; identifier
0328 048d d6 01 02              lda     eeprom0+2,x             ; bytes
0329 0490 b7 88                 sta     pepid+1
0330 0492 d6 01 03              lda     eeprom0+3,x
0331 0495 b7 89                 sta     pepid+2
0332 0497 d6 01 40              lda     eeprom0+eepcnt,x        ; load three
0333 049a b7 80                 sta     count                   ; count bytes
0334 049c d6 01 41              lda     eeprom0+eepcnt+1,x
0335 049f b7 81                 sta     count+1
0336 04a1 d6 01 42              lda     eeprom0+eepcnt+2,x
0337 04a4 b7 82                 sta     count+2
0338 04a6 81                    rts
0339
0340                    ********************************************************************************
0341                    * This is a general purpose routine for moving data pointed to by the          *
0342                    * X reg to the location pointed to by the accumulator. Three bytes are         *
0343                    * moved.                                                                       *
0344                    ********************************************************************************
0345
0346 04a7 bf 8d         mvdat   stx     gpcnt           ; save destination
0347 04a9 83                    swi                     ; move first byte
0348 04aa 4c                    inca                    ; point to second byte
0349 04ab 83                    swi                     ; move second byte
0350 04ac 4c                    inca                    ; point to third byte
0351 04ad 83                    swi                     ; move third byte
0352 04ae 81                    rts
0353                    ********************************************************************************
0354                    * This is a general purpose rotate routine used for transmitting data or       *
0355                    * reading the peprom. It can be entered to rotate 4 or 3 bytes. The            *
0356                    * bits are rotated one place. The 8th bit of the 4th byte is not loaded        *
0357                    * with the carry.                                                              *
0358                    ********************************************************************************
0359
0360 04af 66 03         rotate  ror     3,x             ; rotate right 4
0361 04b1 66 02         rot3    ror     2,x             ; or 3 bytes
0362 04b3 66 01                 ror     1,x             ; pointed to by X reg
0363 04b5 76                    ror     0,x             ; lowest bit into carry
0364 04b6 81                    rts
0365
0366
0367
0368
```

```
0369                    ********************************************************************************
0370                    * This routine encrypts the current count and the identifier into a code       *
0371                    * for transmission. The count is rotated and is Exclusive-Ored with the        *
0372                    * the identifier if the LSB is logic 1. The process is repeated for all        *
0373                    * 24 bits.                                                                     *
0374                    ********************************************************************************
0375
0376 04b7 a6 80          algo    lda     #count          ; point to start
0377 04b9 ae 84                  ldx     #comdat+1       ; point to destination
0378 04bb ad ea                  bsr     mvdat           ; move data
0379 04bd ae 20                  ldx     #$20            ; 32 bytes
0380 04bf 9f            algbit  txa
0381 04c0 ae 84                  ldx     #comdat+1       ; point to first byte
0382 04c2 ad ed                  bsr     rot3            ; rotate
0383 04c4 97                     tax
0384 04c5 24 14                  bcc     noxor           ; test if first byte=0
0385 04c7 1e 86                  bset    7,comdat+3       ; else exclusive-Or
0386 04c9 b6 87                  lda     pepid           ; with counter
0387 04cb b8 84                  eor     comdat+1
0388 04cd b7 84                  sta     comdat+1
0389 04cf b6 88                  lda     pepid+1
0390 04d1 b8 85                  eor     comdat+2
0391 04d3 b7 85                  sta     comdat+2
0392 04d5 b6 89                  lda     pepid+2
0393 04d7 b8 86                  eor     comdat+3
0394 04d9 b7 86                  sta     comdat+3
0395 04db 5a            noxor   decx                    ; if not all bits rotated
0396 04dc 26 e1                  bne     algbit          ; test next bit
0397 04de 81                     rts
0398
0399                    ********************************************************************************
0400                    * This routine increments the count value after each key press                 *
0401                    ********************************************************************************
0402
0403 04df 3c 80          incnt   inc     count           ; inc low byte
0404 04e1 26 0a                  bne     nover           ; if overflow
0405 04e3 3c 81                  inc     count+1         ; inc mid byte
0406 04e5 26 06                  bne     nover           ; if overflow
0407 04e7 3c 82                  inc     count+2         ; inc high byte
0408 04e9 26 02                  bne     nover           ; if overflow
0409 04eb 3c 80                  inc     count           ; inc low byte
0410 04ed 81            nover   rts
0411
0412
0413                    ********************************************************************************
0414                    * This is the data reception routine. It receives and verifies the             *
0415                    * initialisation data before receiving the next 32 data bits.                  *
0416                    ********************************************************************************
0417
0418 04ee 14 0a          rec     bset    2,tim16         ; enable timer
0419 04f0 3f 95                  clr     stat            ; clear flags
0420 04f2 b6 13                  lda     tsr
0421 04f4 b6 15                  lda     icrl
0422 04f6 a6 80                  lda     #%10000000      ; enable timer
0423 04f8 b7 12                  sta     tcr             ; capture on -ve edge
0424 04fa 9a                     cli
0425 04fb 0f 95 fd               brclr   7,stat,*        ; wait for edge
0426 04fe 3f 12                  clr     tcr             ; disable timer ints
0427 0500 a6 07                  lda     #$07
0428 0502 b7 04                  sta     ddra            ; PA0-2 outputs
0429 0504 a6 f9                  lda     #%11111001
0430 0506 b7 00                  sta     porta
0431 0508 81                     rts
0432
0433
0434
0435 0509 b6 84          cmpdat  lda     comdat+1        ; check if 1st byte
0436 050b b1 8a                  cmp     datsto          ; decodes correct
0437 050d 26 0a                  bne     nodec           ; else end
0438 050f b6 85                  lda     comdat+2        ; check if 2nd byte
0439 0511 b1 8b                  cmp     datsto+1        ; decodes correct
0440 0513 26 04                  bne     nodec           ; else end
0441 0515 b6 86                  lda     comdat+3        ; check if 3rd byte
0442 0517 b1 8c                  cmp     datsto+2        ; decodes correct
0443 0519 81            nodec   rts
0444
0445
0446
```

```
0447                    ********************************************************
0448                    *                                                      *
0449                    *       TCAP IR receiver routine (MC144105).           *
0450                    *                                                      *
0451                    ********************************************************
0452
0453 051a a6 03        timer   lda     #%00000011
0454 051c b7 00                sta     porta
0455 051e a6 ff                lda     #$ff
0456 0520 b7 04                sta     ddra
0457 0522 b6 14                lda     icrh            ;  save
0458 0524 b7 93                sta     irra3           ;  msb
0459 0526 b6 13                lda     tsr             ;  clear icf bit
0460 0528 b6 15                lda     icrl            ;  save
0461 052a b7 94                sta     irra4           ;  lsb
0462 052c b0 92                sub     irra2           ;  take away old lsb
0463 052e b7 8f                sta     diffl           ;  and save
0464 0530 b6 93                lda     irra3           ;  subtract
0465 0532 b2 91                sbc     irra1           ;  msbs
0466 0534 b7 90                sta     diffh           ;  and save difference
0467 0536 b6 93                lda     irra3           ;  transfer
0468 0538 b7 91                sta     irra1           ;  latest
0469 053a b6 94                lda     irra4           ;  values
0470 053c b7 92                sta     irra2           ;  to old valyes
0471
0472 053e b6 8f                lda     diffl           ;  add tolerance
0473 0540 ab 78                add     #tol/2          ;  to make sure msb
0474 0542 b7 8f                sta     diffl           ;  is ok
0475 0544 b6 90                lda     diffh           ;  adjust msb
0476 0546 a9 00                adc     #$00            ;  with carry
0477 0548 b7 90                sta     diffh           ;  from adjustment
0478
0479 054a b6 8f                lda     diffl            ; if lsb is more than
0480 054c a1 f0                cmp     #tol             ; twice the tolerance
0481 054e 22 55                bhi     ill              ; delay was not legal
0482
0483                    ********************************************************
0484                    *                                                      *
0485                    *       6 half-bit delta: pre-amble & start.           *
0486                    *       2 half-bit delta: repeat last bit.             *
0487                    *                                                      *
0488                    ********************************************************
0489
0490 0550 b6 90        try6    lda     diffh
0491 0552 a1 06                cmp     #6              ; delta: 6 half bits ?
0492 0554 22 4f                bhi     ill             ; if more then can't be legal
0493 0556 26 07                bne     not6
0494 0558 a6 01                lda     #1              ; pre-amble has happened
0495 055a b7 96                sta     tcnt            ; so ready to start
0496 055c 1e 86                bset    7,comdat+3           start bit a one
0497 055e 80                   rti
0498
0499 055f b6 96        not6    lda     tcnt            ; if zero the correct
0500 0561 27 44                beq     irab            ; pre-amble has not occured
0501
0502 0563 b6 90        try2    lda     diffh
0503 0565 a1 02                cmp     #2              ; delta: 2 half bits ?
0504 0567 26 05                bne     try3
0505 0569 0f 86 24             brclr   7,comdat+3,shft ;     same again
0506 056c 20 22                bra     shft
0507
0508                    ********************************************************
0509                    *                                                      *
0510                    *       3 half-bit delta, if last bit was a            *
0511                    *       0 then two 1s, if it was a 1 then one 0.       *
0512                    *                                                      *
0513                    ********************************************************
0514
0515 056e a1 03        try3    cmp     #3              ;delta: 3 half bits ?
0516 0570 26 0e                bne     try4
0517
0518 0572 0e 86 1a             brset   7,comdat+3,shft0   ;  if last a 1 then one 0
0519 0575 99                   sec                       ;if a 0 then two 1s
0520 0576 ae 83                ldx     #comdat
0521 0578 cd 04 af             jsr     rotate
0522 057b 3c 96                inc     tcnt
0523 057d 99                   sec
0524 057e 20 10                bra     shft            ;second
0525
```

```
0526                    ********************************************************
0527                    *                                                      *
0528                    *       4 half-bit delta: if last bit was 0 then a     *
0529                    *       1 followed by a 0, illegal if it was a 1.      *
0530                    *                                                      *
0531                    ********************************************************
0532
0533
0534 0580 a1 04         try4    cmp     #4              ;delta should be 4
0535 0582 26 21                 bne     ill
0536 0584 0e 86 1e              brset   7,comdat+3,ill
0537 0587 99                    sec
0538 0588 ae 83                 ldx     #comdat
0539 058a cd 04 af              jsr     rotate
0540 058d 3c 96                 inc     tcnt
0541 058f 98            shft0   clc
0542
0543 0590 ae 83         shft    ldx     #comdat
0544 0592 cd 04 af              jsr     rotate
0545 0595 3c 96                 inc     tcnt
0546 0597 b6 96                 lda     tcnt
0547 0599 a1 20                 cmp     #$20            ;command finished ?
0548 059b 25 04                 blo     notfin          ;if less, not finished
0549 059d 22 06                 bhi     ill             ;if more, not legal
0550 059f 1e 95                 bset    7,stat
0551 05a1 80            notfin  rti
0552
0553                    ********************************************************************************
0554                    * This routine is puts a delay between receptions. The reception error   *
0555                    * LED is also enabled.                                                   *
0556                    ********************************************************************************
0557
0558 05a2 cc 03 00      thend   jmp     start
0559
0560 05a5 3f 96         ill     clr     tcnt
0561 05a7 80            irab    rti
0562
0563 05a8 ad 03         badend  bsr     wend
0564 05aa cc 03 00              jmp     start           ; delay
0565
0566 05ad 81            wend    rts
0567 05ae 3f 0a                 clr     tim16
0568 05b0 a6 25                 lda     #$25
0569 05b2 b7 8d                 sta     gpcnt
0570 05b4 4f                    clra
0571 05b5 5f                    clrx
0572 05b6 5a            dlay    decx                    ; 3us * 256
0573 05b7 26 fd                 bne     dlay            ;  = 786uS
0574 05b9 4a                    deca                    ; + 3us * 256
0575 05ba 26 fa                 bne     dlay            ;  = 197ms
0576 05bc 3a 8d                 dec     gpcnt           ; + 4us * 25
0577 05be 26 f6                 bne     dlay            ;  = 4.9s
0578 05c0 81                    rts
0579
0580
0581 05c1 97            soft    tax                     ; pointer for source
0582 05c2 f6                    lda     ,x              ; load source byte
0583 05c3 be 8d                 ldx     gpcnt           ; point to destination
0584 05c5 f7                    sta     ,x              ; move byte
0585 05c6 3c 8d                 inc     gpcnt           ; point to next byte
0586 05c8 80                    rti
0587 05c9 a6 64         beep    lda     #$64
0588 05cb b7 8d                 sta     gpcnt
0589 05cd ad 06         lb      bsr     buzz
0590 05cf 3a 8d                 dec     gpcnt
0591 05d1 26 fa                 bne     lb
0592 05d3 20 fe                 bra     *
0593
0594 05d5 16 04         buzz    bset    3,ddra
0595 05d7 a6 60                 lda     #$60
0596 05d9 16 00         buz2    bset    3,porta
0597
0598
0599 05db ae 80                 ldx     #$80
0600 05dd ad 0c                 bsr     buzlp
0601 05df 17 00                 bclr    3,porta
0602 05e1 ae 80                 ldx     #$80
0603 05e3 ad 06                 bsr     buzlp
0604 05e5 4a                    deca
0605 05e6 26 f1                 bne     buz2
0606 05e8 17 04                 bclr    3,ddra
```

```
0607 05ea 81                        rts
0608
0609 05eb 5a                buzlp   decx
0610 05ec 26 fd                     bne     buzlp
0611 05ee 81                        rts
0612
0613 05ef 5f                bulk2   clrx
0614 05f0 cd 03 e0          bulkl2  jsr     erase
0615 05f3 a6 54                     lda     #$54
0616 05f5 cd 04 06                  jsr     eeprog
0617 05f8 5c                        incx
0618 05f9 a3 80                     cpx     #$80
0619 05fb 26 f3                     bne     bulkl2
0620 05fd cc 03 00                  jmp     start
0621
0622 0600 5f                bulk    clrx
0623 0601 cd 03 e0          bulkl   jsr     erase
0624 0604 5c                        incx
0625 0605 a3 80                     cpx     #$80
0626 0607 26 f8                     bne     bulkl
0627 0609 cc 03 00                  jmp     start
0628
0629
0630 0ff6                           org     vector
0631
0632 0ff6 05 1a                     fdb     timer           ; tcap
0633 0ff8 03 00                     fdb     start           ; core timer
0634 0ffa 03 00                     fdb     start           ; irq
0635 0ffc 05 c1                     fdb     soft            ; swi
0636 0ffe 03 00                     fdb     start           ; reset
0637
0638                                end
0639
```

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Freescale Semiconductor, Inc.

freescale™
semiconductor