OVERVIEW

This application note describes a KEELOQ® code hopping decoder implemented on a Microchip Mid-Range PICmicro microcontroller (PIC16CE624). The software has been designed as a group of independent modules (standard assembly include files "inc"). For clarity and ease of maintenance each module covers a single function. Each module can be modified to accommodate different behavior, support a different microcontroller (MCU), and/or a different set of peripherals (memories, timers, etc.).

KEY FEATURES

The set of modules presented in this application note implement the following features:

- Normal Learn mode
- Learn up to 16 transmitters, using internal EEPROM memory of a PIC16CE624
- Interrupt driven Radio Receiver (PWM) routine
- Compatible with all existing KEELOQ hopping code encoders with PWM transmission format selected, operating in “slow mode” (TE = 400 μs)
- Pinout compatible with HCS512 decoder (fits in KEELOQ Evaluation Kit demo board)
- RC oscillator (self-calibrating during receive)

Notice:

This is a non-restricted version of Application Note AN743 which is available under the KEELOQ License Agreement on the web site www.microchip.com.
DESIGN OBJECTIVES

Each module has been designed for maximum simplicity and maintainability. Whenever possible we favored clarity of design over efficiency, in order to show the basic concept of the design of a KEELOQ decoder without the complications that various constraints (limited RAM, STACK or other resources) could and did pose on (previous) other implementations.

To achieve the goal of maximum ease in maintenance, we also adopted “modern” assembly software design techniques, specifically:

- We applied the basic concepts of structured programming; all routines have a single point of entry and exit
- Inputs and output values are documented
- We made extensive use of the CBLOCK/ENDC pseudo-instruction of the MPASM™ assembler to automatically assign an address to RAM variables
- All pin assignments are mapped through #define directives to obtain nearly complete code independence from the specific pinout chosen
- Drivers to peripherals that are specific to a given processor type (i.e., PIC16CE624) have been encapsulated in more generic modules
- Whenever possible comments include pseudo-graphical representation of the data structures used and/or program flow.

MODULES OVERVIEW

The code presented in this application note is composed of the following basic modules:

- RXI.INC interrupt driven receiver
- KEYGEN.INC KEELOQ key generation routines implementing Normal mode
- FASTDEC.INC KEELOQ decrypt routine
- MEM-62X.INC encapsulates PIC16CE62X EEPROM drivers (FL62XINC.ASM)
- TABLE.INC transmitters table memory management (linear list)
- MID.ASM the actual initialization and main loop

FIGURE 2: MODULES OVERVIEW
RECEIVER MODULE

The receiver module has been developed around a fast and independent Interrupt Service Routine (ISR) that acts like a “virtual peripheral”. The whole receiving routine is implemented as a simple state machine that operates on a fixed time base (which can be used to produce a number of virtual timers). The working of this routine is completely transparent to the main program and similar to a UART. In fact, the interrupt routine consumes only 30% of the computational power of the MCU working in the background.

After a complete transmission code word of 66 bits has been received and stored in a 9 bytes buffer, a simple flag (RF_FULL) is set and the receiver becomes idle.

It is the responsibility of the main program to make use of the data in the buffer and to reset the flag to enable the receiving of a new transmission.

In order to obtain maximum compatibility with all KEELÖQ encoders, with or without oscillator tuning capabilities, the receiver routinely recalibrates itself by changing the time base period according to the length of the characteristic synchronization pause (TH = 10 x TE). This allows the decoder to operate from an inexpensive (uncalibrated) RC clock.

FIGURE 3: CODE WORD TRANSMISSION FORMAT

FIGURE 4: CODE WORD ORGANIZATION
The only peripheral used by this routine is Timer0 and its Overflow Interrupt, available on ANY mid-range PICmicro MCU. The timer is reloaded at any overflow creating a time base (of about 120 μs) and the same Interrupt Service Routine provides a virtual 16-bit timer derived from the same base period called XTMRH/XTMRL.

The receiving routine eventually modifies the period of this time base (only) during the reception of the 66 bits of a transmission (stretching or compressing it), in order to better synchronize and compensate the clock differences between the encoders and the decoder.

Since the radio input is polled only on multiples of the base period (N x 120 μs), the chance of a glitch (short noise pulse) to disturb the receiver is reduced.

Other implementations of the same receiver module can be obtained using other peripherals and detection techniques. For example:
- Using the INT pin and selectable edge interrupt source
- Using the Timer1 and CCP module in Capture mode (wherever available)
- Using comparator inputs interrupt (PIC16CE62X)

Any of these techniques pose different constraints on the pinout or the PICmicro microcontroller that can be used and leads to different performances in terms of achievable immunity from noise and CPU load.

**FAST DECRYPTION MODULE**

This module contains an implementation of the KEELOQ decryption algorithm that has been optimized for speed on a mid-range PICmicro MCU. It allows fast decryption times for maximum responsiveness of the system, even at 4 MHz clock.

The decryption function is also used in all learning schemes and represents the fundamental building block of all KEELOQ decoders.

**KEY GENERATION MODULE**

This module shows a simple and linear implementation of the Normal Learn Key Generation.

The KEELOQ Decrypt routine from the Fast Decryption module is used to generate the key at every received code word instead of generating it during the learn phase and storing it into memory. The advantage is a smaller Transmitter Record of 8 bytes instead of 16 bytes (see Table 2). That translates in a double number of transmitters that can be learned using the 128 byte internal EEPROM available inside the PIC16CE624. This space reduction comes at the expense of more computational power required to process every code word. When a new code word is received, the key generation algorithm is applied (Normal Learn) and the resulting Description Key is placed in the array DKEY [0..7]. During a continuous transmission, when the user is holding the button on the transmitter, the key generation is not repeated. To save time, the last computed Decryption Key value is used safely instead with the serial number being the same.

For an overview of some of the different security levels that can be obtained through the use of different key generation/management schemes, refer to the “Secure Data Products Handbook” (DS40168) (Section 1, KEELOQ Comparison Chart, Security Level Summary).

A detailed description of the Normal Learn key generation scheme can be found in Technical Brief TB003 “An Introduction to KEELOQ Code Hopping” (DS91002).

More advanced Key Generation Schemes can be implemented replacing this module with the techniques described in Technical Brief TB001 “Secure Learning RKE Systems using KEELOQ Encoders” (DS91000).

**TABLE MODULE**

One of the major tasks of a decoder is that of properly maintaining a database containing all the unique IDs (serial numbers) of the learned transmitters. In most cases, the database can be as simple as a single table, that associates those serial numbers with the synchronization counters (which are at the heart of the hopping code technology).

This module implements the easiest of all methods, a simple “linear list” of records.

Each transmitter learned is assigned a record of 8 bytes (shown in Table 2) where all the relevant information is stored and regularly updated.

**TABLE 2: TRANSMITTER RECORD**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0</td>
<td>XF</td>
<td>Function code (4 bits) and upper 4 Serial Number bits [24..28]</td>
</tr>
<tr>
<td>+1</td>
<td>IDLo</td>
<td>Serial Number bits [0..7]</td>
</tr>
<tr>
<td>+2</td>
<td>IDHi</td>
<td>Serial Number bits [8..15]</td>
</tr>
<tr>
<td>+3</td>
<td>IDMi</td>
<td>Serial Number bits [16..23]</td>
</tr>
<tr>
<td>+4</td>
<td>SYNCH</td>
<td>Sync Counter 8 MSb</td>
</tr>
<tr>
<td>+5</td>
<td>SYNCL</td>
<td>Sync Counter 81 Sb</td>
</tr>
<tr>
<td>+6</td>
<td>SYNCH2</td>
<td>Second copy of SyncH</td>
</tr>
<tr>
<td>+7</td>
<td>SYNCL2</td>
<td>Second copy of SyncL</td>
</tr>
</tbody>
</table>

The 16-bit synchronization counter value is stored in memory twice because it is the most valuable piece of information in this record. It is continuously updated at every button press on the remote. When reading the two stored synchronous values, the decoder should verify that two copies match. If not, it can adopt any safe resync or disable technique required, depending on the desired system security level.
The current implementation limits the maximum number of transmitters that can be learned to 16. This is due to the size of the internal EEPROM of the PIC16CE624.

This number can be changed to accommodate different PICmicro MCU models and memory sizes by modifying the constant `MAX_USER`.

The simple “linear list” method employed can be scaled up to some tens of users. Due to its simplicity, the time required to recognize a learned transmitter grows linearly with the length of the table.

It is possible to reach table sizes of thousands of transmitters by replacing this module with another that implements a more sophisticated data structure like a “Hash Table” or other indexing algorithm.

Again, due to the simplicity of the current solution, it is not possible to selectively delete a transmitter from memory. The only delete function available is a Bulk Erase (complete erase of all the memory contents). This happens when the user presses the Learn button for up to 10 seconds. The LED will switch off and at release of the button will flash once to acknowledge the delete command. To allow for selective transmitter removal from memory, more sophisticated techniques will be analyzed in future application notes.

**MEM-62X MODULE**

This module is an envelope built around an existing set of routines that are specifically optimized to drive the internal EEPROM of the PIC16CE62X device that is provided by Microchip as standard example code. Information can be downloaded from the Microchip web site “http://www.microchip.com” by following the links to Knowledge Base/Object Templates for Writing Code/I2C™ code for the PIC16CE62X Family with internal EEPROM.

The module makes the memory generically accessible by means of two routines, `RDword` and `WRword`, that read and write respectively, a 16-bit value out of an even address specified in `INDHI/INDLO`.

Replacing this module with the appropriate drivers and adapting the pinout, makes possible the use of any kind of nonvolatile memory. This includes internal and external serial EEPROM (Microwire, SPI or `I^2^C™` bus) of any size up to 64 Kbytes.

**THE MAIN PROGRAM**

The main program is reduced to a few pages of code. The behavior is designed to mimic the basic behavior of the HCS512 integrated decoder, although just the Stand-Alone mode of operation is functional (no Co-Processor mode).

Most of the time the main loop goes idle waiting for the receiver to signal complete reception of a full code word.

Double buffering of the receiver is done in RAM in order to immediately re-enable the reception of new codes and increase responsiveness and perceived range.

**CONCLUSION**

The basic principles of structured programming have been applied in this project to build a KEELOQ Hopping Code Decoder. The larger RAM memory available and deeper hardware stack of the PICmicro mid-range family allows us to make the code simpler and cleaner. Interrupts have been put to use to “virtualize” the receiving routine as a software peripheral and free the design of the hard real time constraint that it usually imposes.

We resisted introducing extra features/optimizations in favor of clarity among which:

- RAM space optimization, reuse of registers used as local variables to functions
- Speed optimizations, code compacting
- More complex key generation schemes
- Co-processor functionality
- Advanced user entry and deletion commands

These are left as exercises to the advanced reader/designer or as suggestions for future application notes.
MEMORY USAGE
Program Memory Words Used: 852 words
File Registers Used: 66 bytes

KEYWORDS
Mid-Range, KEELOQ, Decoder and PIC16CE62X

REFERENCES
- KEELOQ Code Hopping Decoder on a PIC16C56 AN642 DS00642
- Converting NTQ105/106 Designs to HCS200/300s AN644 DS00644
- Code Hopping Security System on a PIC16C57 AN645 DS00645
- Secure Learn Code Hopping Decoder on a PIC16C56 AN652 DS00652
- KEELOQ Simple Code Hopping Decoder AN659 DS00659
- KEELOQ Code Hopping Decoder on a PIC16C56 (public version) AN661 DS00661
- Secure Learn Code Hopping Decoder on a PIC16C56 (public version) AN662 DS00662
- KEELOQ Simple Code Hopping Decoder (public version) AN663 DS00663
- Using KEELOQ to Generate Hopping Passwords AN665 DS00665
- PICmicro Mid-Range MCU Code Hopping Decoder AN662 DS00672
- HCS410 Transponder Decoder using a PIC16C56 AN675 DS00675
- Modular Mid-Range PICmicro KEELOQ Decoder in C AN744 DS00744
- Secure Learning RKE Systems Using KEELOQ Encoders TB001 DS91000
- An Introduction to KEELOQ Code Hopping TB003 DS91002
- A Guide to Designing for EuroHomelink Compatibility TB021 DS91021
- KEELOQ Decryption and IFF Algorithms TB030 DS91030
- KEELOQ Decryption Routines in C TB041 DS91041
- Interfacing a KEELOQ Encoder to a PLL Circuit TB042 DS91042
- KEELOQ CRC Verification Routines TB043 DS91043
APPENDIX A: MID SOURCE CODE

; LIST n=0, c=132
PROCESSOR PIC16CE624
RADIX HEX

;**********************************************************************
;* Filename: MID.ASM
;**********************************************************************
;* Author: Lucio Di Jasio
;* Company: Microchip Technology
;* Revision: Rev 1.00
;* Date: 09/25/00
;*
;* Keeloq receiver and decoder for Mid-range PICmicro
;*
;* USES:
;* keygen.inc ; key generation, code hopping checking
;* fastdec.inc ; Keeloq decrypt routine
;* mem-62x.inc ; generic I2C routines
;* fl62xinc.asm ; specific internal memory drivers
;* rxi.inc ; interrupt receiver
;* table.inc ; table memory management
;*
;* Assembled using MPASM v02.40
;**********************************************************************

include "p16ce624.inc"
errorlevel -302 ; disable this message type

#define DEBUG 1 ; CP OFF for use with windowed devices

ifdef DEBUG
__CONFIG _RC_OSC & _PWRT_ON & _WDT_ON & _BODEN_ON & _CP_OFF
else
__CONFIG _RC_OSC & _PWRT_ON & _WDT_ON & _BODEN_ON & _CP_ALL
endif

__IDLOCS H'0100'

#define BANK1 bsf STATUS,RP0 ; select Bank 1
#define BANK0 bcf STATUS,RP0 ; select Bank 0

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the “Company”) is intended and supplied to you, the Company’s customer, for use solely and exclusively with products manufactured by the Company. The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
I/O definitions
(PIC16CE624 compatible with HCS512)

Learn - RA2  O  RA1 - RFIn
Led - RA3  RA0 - NU
NU - RA4/T0 OSC - XTAL
Reset - MCLR TST - XTAL
GND - Vss Vdd - +5V
S0 - RB0/INT RB7 - NU
S1 - RB1 RB6 - NU
S2 - RB2 RB5 - NU
S3 - RB3 RB4 - Vlow

#define RFIn PORTA,1 ; i radio signal input
#define Learn PORTA,2 ; i learn button
#define Led PORTA,3 ; o learn Led
#define Out0 PORTB,0 ; o S0 output
#define Out1 PORTB,1 ; o S1 output
#define Out2 PORTB,2 ; o S2 output
#define Out3 PORTB,3 ; o S3 output
#define Vlow PORTB,4 ; o low battery

MASKA equ b'11110111' ; port A I/O config
MASKB equ b'11100000' ; port B I/O config

OPTION_RS equ b'00001111' ; prescaler assigned to WDT, TMR0 clock/4, pull up

keeloq receive buffer map

Plain text                                | Encrypted
RV000000.KKKKKK.IIIIIIII.IIIIIIIIIIIIIIIIIIIIIII.KKKKODDD.DDDDDDD.SSSSSSSS.SSSSSSSS
8  7  6  5  4  3  2  1  0

I=S/N -> SERIAL NUMBER (28 BIT)
K=KEY -> buttons encoding (4 BIT)
S=Sync -> Sync counter (16 BIT)
D=Disc -> Discrimination bits (10 BIT)
R=Rept -> Repeat/first (1 BIT)
V=Vlow -> Low battery (1 BIT)

-- alias -------------------------------------------

#define HopLo CSR0 ; sync counter
#define HopHi CSR1 ;
#define DisLo CSR2 ; discrimination bits LSB
#define DOK CSR3 ; Disc. MSB + Ovf + Key
#define IDLo CSR4 ; S/N LSB
#define IDMi CSR5 ; S/N
#define IDHi CSR6 ; S/N MSB
#define S0 DOK,5 ;function codes
#define S1 DOK,6
#define S2 DOK,7
#define S3 DOK,4
#define VFlag CSR8,6 ; low battery flag

RAM allocation


; reserved temp. storage in common access bank
CBLOCK 070
    W_TEMP
    STATUS_TEMP
ENDC

; general use
CBLOCK 020
    FSR_TEMP
    PCLATH_TEMP

; receive/decode buffer
    CSR0
    CSR1
    CSR2
    CSR3
    CSR4
    CSR5
    CSR6
    CSR7
    CSR8

; flags
    Flags

; debouncing/input/output timings
    CFFlash ; flash counter x2
    CTFlash  ; flash period
    CLearn   ; debounce Learn button
    CTLearn  ; temp. Learn
    COut     ; temp. outputs
ENDC

;------------------------------------------------------------
; various Flags definitions
;
#define Flag_HopOK Flags,0   ; hopping code checked OK
#define Flag_2C Flags,1      ; allow a re-sync
#define Flag_Same Flags,2    ; received same code as previous
#define Flag_Learn Flags,3    ; learn mode
#define Flag_72 Flags,4      ; flips every 36ms

;------------------------------------------------------------
; timings
;
TOUT  equ .5               ; 5 * 71ms = 350ms
TFLASH equ .2              ; 2 * 71ms = 140ms flashing period
TLEARN equ .255           ;
255 * 71ms = 18s learn time out

;------------------------------------------------------------
; org 00             ; reset vector
 goto Start

;------------------------------------------------------------
; ISR radio receiver
;
org 04               ; interrupt vector

#include "rxi.inc"

;------------------------------------------------------------
; Keeloq decoding

; include "keygen.inc" ; implements Normal Learn

; IIC bus EEPROM read/write routines

; include "mem-62x.inc" ; encapsulate specific EEPROM drivers

; table search/insert management

; include "table.inc" ; memory table management

; init all ports and timer

; InitPorts

BANK1
movlw MASKA
movwf PORTA
movlw MASKB
movwf PORTB
movlw OPTION_RS
movwf OPTION_REG

BANK0

movlw b'00000111' ; comparators off
movwf CMCON
return

; Start

CLRWDT

clr PORTA ; clear all outputs

cclr PORTB

call InitPorts ; init ports and timer

call InitRX

cclr Flags ; clear all flags

cclr CFlash ;
cclr COut ; reset all timers

cclr CLearn ; reset debouncing inputs

cclr CTLearn ; reset timer Learn

; I/O polling loop (every 72ms)

T72
btfssFlag_72

goto T720

T721
btfsc XTMRH,1 ; wait for falling edge 512 x Tbase = 72ms

goto Main

;
bcf Flag_72 ; clear

goto TLearn

T720

btfss XTMRH,1 ; wait for rising edge 512 x Tbase = 72ms

goto Main

; on the rising of Flags_72 execute once the polling loop

POLL

bsf Flag_72 ; set

;--------------------------------------------------------------------
; debounce Learn button
; (0,5 seconds minimum, active low)
; TLearn

btfsc Learn ; debounce button

goto NoLearn

incf CLearn,F ; counts 2 every 72ms

incf CLearn,F

btfsc STATUS,Z

goto TClearMem ; times out in 10s

; 4 counts = 288ms

TLenter

movlw .4 ; after 1/4 s

subwf CLearn,W ; enter learn

btfss STATUS,C

; goto TLearnE

TLearnON

bsf Flag_Learn ; enter learn mode

; activate learn timer

movlw TLEARN ; it is a count down

movwf CTLearn ; preload

; switch on the Learn Led

bsf Led

goto TLearnE

NoLearn

crfl CLearn ; reset counter

TLearnE

; Output timings

; TOut

movf COut,F ; check if timer running

btfsc STATUS,Z

goto TOutE ; no

; yes, we counting down

decfsz COut,F ; decrement

goto TOutE

; when times out

bcf Out0 ; switch off all outputs

bcf Out1

bcf Out2

bcf Out3

bcf Vlow ; low battery indication
bcf Led ; switch off Led

TOutE
;
;--------------------------------------------------------------------------
; Learn timing
;
TRLearn
    movf CTLearn,F ; is timer running
    btfsc STATUS,Z
    goto TRLearnE ;
    ; yes it is a count down
    decfsz CTLearn,F ; decrement
    goto TRLearnE
    bcf Flag_Learn ; end Learn
    bcf Led ; switch Led off
TRLearnE
;--------------------------------------------------------------------------
; Led Flashing
;
TFlash
    movf CFlash,F
    btfsc STATUS,Z
    goto MainE ; no flashing
    decfsz CTFlash,F ; flashing timer decrement
    goto MainE
    movlw TFLASH ; period timer reload
    movwf CTFlash
    bcf Led
    btfss CFlash,0 ; toggle on/off
    bsf Led ; Led == ~(CFlash.1)
    decf CFlash,F ;
MainE
    goto Main

;--------------------------------------------------------------------------
; Clearing Memory
;
TClearMem
    bcf Led ; first turn Led OFF
    CLRWDT
    btfss Learn ; loop until button released
    goto TClearMem
    bsf Led ; Led ON
    call ClearMem ; erase all
    bcf Flag_Learn ; no Learn
    clrf CTLearn ; reset all timers
    clrf CTFlash ; and counters
    clrf CFlash ;
    bsf Led ; turn Led ON
    movlw 2*TOUT ; single long Flash
    movwf COut
    goto Succed

;--------------------------------------------------------------------------
; decode a received message
Remote
; double buffering B0..7 -> CSR0..7

movf B0,W ; copy receive buffer
movwf CSR0 ; in decode buffer
movf B0+1,W
movwf CSR1
movf B0+2,W
movwf CSR2
movf B0+3,W
movwf CSR3
movf B0+4,W
movwf CSR4
movf B0+5,W
movwf CSR5
movf B0+6,W
movwf CSR6
movf B0+7,W
movwf CSR7
movf B0+8,W
movwf CSR8

bcf RF_Full ; make the receive buffer immediately available
            ; to increase the receiver performance

Decode
    call NormalKeyGen; Key generation algorithm (normal)
        call Decrypt ; Keeloq decryption
    call DecCHK ; test successful decryption
    btfss STATUS,Z
    goto Fail ; discard if failed

; code passed first decryption test

TestLearn
    btfss Flag_Learn ; if we are not in learn mode
    goto NormalMode ; discard

LearnMode

RemSearch
    call Find ; look into EEPROM for the Serial Number
    btfsc Flag_Found ;
    goto ReLearn ; known transmitter update

; not found this is a possible new transmitter to learn
    call Insert ; look for space in EEPROM to store the new TX
    btfsc Flag_MFull ; if memory is full
    goto Fail ; discard

ReLearn
; ASSERT Ind is pointing to a valid memory location
; memorize the function code used for learning (button pressed)
    movf DOK,W ; save function code
    movwf XF ; and upper ID in XF

; memorize Serial Number and Function keys
LearnID
    call IDWrite ; save XF, ed ID

; update hopping code
LearnHop
    bsf Flag_HopOK ; guard check
    call HopUpdate ; memorize sync counter first value
    bcf Flag_HopOK ;
; learn successful, flash Led 4 times
;
    movlw  32          ; 16 flashes
    movwf  CFlash      ; init the counter
    movlw  TFLASH      ; flashing period
    movwf  CTFlash     ;
    bsf   Led          ; start with Led ON
    bcf    Flag_Learn  ; learning finished
Succed
    goto   Main
Fail
    goto   Main
;
;---------------------------------------------------------------
;
NormalMode
    call    Find       ; look into EEPROM for the Serial Number
    btfss   Flag_Found ; unknown transmitter
    goto    Fail       ;
RecNormal
    ; ASSERT Ind is pointing to the location where Serial Number was found
APPENDIX B: RXI SOURCE CODE

; Filename: RXI.INC
;**********************************************************************
; Author: Lucio Di Jasio
; Company: Microchip Technology
; Revision: Rev 1.00
; Date: 06/07/00
;*
; Assembled using MPASM v02.40
;**********************************************************************

Interrupt based Radio Receiver
; designed for Te = 400us (slow mode) with 3x oversampling
; this version uses only Timer0 (suitable for any mid-range PICmicro)
; no Pin Out constraints
; designed for low sensitivity to noise
; self calibrating adjusting on Tsync pause after preamble
; with very high oscillator/encoder freq. tolerance (close to +/- 50%)
; 4MHz RC oscillator (does not require crystals or resonators)
; Timer0 generates a time base with 120us period
; period is adjusted during receiving of a Keeloq transmission
; to sync at best with the encoder's internal oscillator
; CPU load due to the interrupt service is limited to 30% at 4MHz
; and is almost independent from noise and in general from receiver activity
;**********************************************************************

#define XTAL.4000000
#define RF_OVERS3 ; 3 * 120 = 360us + autocalib
#define RF_NBITS.66
#define STD_TIME .120 ; us Tsync (standard sampling period)

CBLOCK
; second RF CSR
B0:4 ; 4 locations encrypted
B4:5 ; 5 locations plain text

RFP ; puntatore al byte corrente
RFbitc ; contatore bit ricevuti
RFsamp ; contatore sample di allineamento e sync
RFState ; stato macchina di ricezione
RFSkip ; contatore di skip
RFXflags ; flag di ricezione
RFtime ; autocalibrazione HCS

XTMRL ; timer base a 16 bit
XTMRH

ENDC

#define RXFflags,0 ; receive buffer full
#define RXFflags,1 ; sampled input value

; Async_ISR
;
; this routine must be included at the interrupt vector address

if ($ != 4)
  error "verify receiver routine located at interrupt vector"
endif
if ((W_TEMP & 0xfff0) != 0x70)
  error "Verify W_TEMP in common bank 70..7f"
endif
if ((STATUS_TEMP & 0xfff0) != 0x70)
  error "Verify STATUS_TEMP in common bank 70..7f"
endif

IntVector
movf  W_TEMP, W ; must have been placed in common bank
swapf STATUS, W
movf  STATUS_TEMP, W ; sempre accessibile
BANK0
;   bsf Led ; to measure INT overhead

;------------------------- old PIC16C62x context saving -------------------
;   bcf STATUS, RP1 ; save bank
;   IFS STATUS, RP0
;   bcf STATUS, RP1
;   BANK0 ; change to bank 0
;   movf  W_TEMP ; save context
;   swapf STATUS, W
;   movf  STATUS_TEMP

; assuming only Timer0 ovflw interrupt enabled
; there is no need to check interrupt source
; if more interrupts enabled ... 
; ADD switch to ISR here

Async_ISR  movf  RFtime, W ; non disruptive timer reload
           subw  Timer0, F
           bcf INTCON, T0IF ; interrupt served

; sample RF input pin
           bcf RFBit ; read input pin in RFBit
           btfsc RFIn
           bsf RFBit

; maintain a 16 bit extended TIMER (XTMRH/L)
;\clocked at +/-120us
;\incXMRH,F ; update 16 bit timer
\incfsz XTMRL,F
\deCF XMRH,F

; receiver state machine starts here
;
AsyncRF  \decf RFSkip, F ; skip if delay required
BNZ ExitIntShort ;
\btfsc RF_Full ; check to avoid overrun
\goto ExitIntShort ; if buffer still full do not touch it
\movf PCLATH, W ; save PCLATH since we will use tables here
\movw PCLATH_TEMP

AsyncStateM
\clRF; we assume this routine has been placed in page 0
\movf RFState, W ; switch ( RFState) {
\andlw 07 ; reduce to 0..7 range
\addw PCL, F ; table offset
RFTable
  goto TRFSYNC ; 0 sync pulse measurement
  goto TRFHALF ; 1 half start bit sync
  goto TRFBIT  ; 2 receive a bit
  goto TRFZERO ; 3 check zero point
  goto TRFCLOCK; 4 sync with next bit start
  goto RFRestore; 5 reset receiver
  goto RFRestore; 6 reset receiver
  goto RFRestore; 7 reset receiver

RFTableEnd

IF HIGH(RFTable) != HIGH(RFTableEnd)
  error "RFTable crosses page border"
ENDIF

;------------------------------------
; state 0 sync pulse measurement
;
; ..|  |__|  |_______________________|  |__|__...
;             |                       |
; preamble -->|<-- Tsync ------------>|<- first bit ...
;
TRFSYNC
  btfsc RFBit ; waiting for a rising edge
  goto TRFRise
  incf RFsamp,F ; try measure Tsync (4 ms)
  incf RFSkip,F ; continua con skip = 1
  goto AsyncRFE

; check boundaries (min, max)
TRFRise
  movlw .21 ; .21 * 120us >2.52 ms min
  subwf RFsamp,W
  btfss STATUS,C
  goto RFRestore ; too short (just a preamble?)
  movlw .56 ; .56 * 120us <6.72 ms max
  subwf RFsamp,W
  btfsc STATUS,C
  goto RFRestore ; too long (transmission start?)

; use measured value of Tsync to calibrate the time base
  CLRRC ; 10Te/120 x 4 = 1/3Te
  rlfrf RFsamp,F ; x2
  rlfrf RFsamp,W ; x4
  movwf RFtime ; 1/3Te optimal time base period

; make ready for receiving the first bit
  clrf RFbitc ; bit counter
  movlw B0 ; init buffer pointer
  movwf RFP
  movlw RF_OVERS/2 ; set half a bit delay to sync with first bit
  movwf RFSkip ;
  incf RFState,F ; move to state 1
  goto AsyncRFE

;------------------------------------
; state 1 start bit sync
;
; | must be high
; V
; +-----+........+
; |   |
; | | | |
; +-----------------......

TRFHALF
btfss RFBit
  goto RFRestore1 ; mid start bit check fails

TRFNEXT
  movlw RF_OVERS ; set a delay of a full bit

TRFSKIP
  movwf RFSkip ;
  incf RFState,F ; move on to next state
  goto AsyncRFE

;------------------------------------
; state 2
; receive a bit
; |
; +---*---+.......+
; |       |       |
; +-------+-------......

TRFBIT
  movf FSR,W ; save FSR
  movwf FSR_TEMP
  movf RFP,W
  movwf FSR ; point to current buffer
  CLRC
  btfss RFBit ; copy in bit (inverted)
  SETC
  rrf INDF,F ; rotate in buffer (Lsb first-> rotate right)
  incf RFbitc,F ; count bits
  movf FSR_TEMP,W ; restore FSR
  movwf FSR
  goto TRFNEXT ; move on next state

;------------------------------------
; state 3
;
; check zero point
;
; +---*---+.......+ must be low
; |
; +-------+-------......
;

TRFZERO
  btfsc RFBit ; end bit check fails
  goto RFRestore3 ;
  movlw 7
  andwf RFbitc,W ; 8 bit read in?
  SKPNZ
  incf RFP,F ; next byte
  movlw RF_NBITS
  subwf RFbitc,W ; received them all?
  BZTRFFULL

; not yet finished, resync on next rising edge

TRFZN
  movlw 1 ; next state without delays (skip=1)
  movwf RFSkip ;
  incf RFState,F ; move on to CLOCK
  clr RFsamp ; init resync counter
goto AsyncRFE

;---------------------------------------------------------
;
TRFFULL      bsf     RF_Full         ; buffer full and ready
goto    RFRestore

RFRestore4
;     nop                     debugging point
RFRestore3
;     nop                     debugging point
RFRestore2
;     nop                     debugging point
RFRestore1
;     nop                     debugging point

;---------------------------------------------------------
;
RFRestore
  clrf    RFState         ; reset state machine
  incf    RFSkip,F        ; preload skip 1
  clrf    RFsamp
  movlw   STD_TIME        ; reset to default speed
  movwf   RFtime
  goto    AsyncRFE

;-----------------------------
; state 4
;
; resync with next bit         waiting this edge
;                     v
; +---*---+.....+        +---
; |       |       |        |
; |       |       |        |
;          +-------+--------+......
;
;-----------------------------
TRFCLOCK
  btfsc   RFBit           ; waiting for rising edge
  goto    TRFCRise        
  incf    RFsamp,F        ; measure length
  incf    RFSkip,F        ; keep skip=1 (sample every 120us)
  goto    AsyncRFE

; edge detected
TRFCRise
  movlw  .3                ; if (RFsamp >=3)
  subwf  RFsamp,W
  btfsc  STATUS, C
  goto    RFRestore4       ; too long delay
  movlw  RF_OVERS/2        ; else
  movwf  RFSkip            ; set delay 1/2Te
  movlw  1                 ; move on to start bit state (1)
  movwf  RFState
  AsyncRFE

;---------------------------------
;
ExitInt
  movf    PCLATH_TEMP, W   ; restore PCLATH_TEMP
  movwf   PCLATH
ExitIntShort
;
    bcf    Led            ; to measure INT overhead
    swapf  STATUS_TEMP,W  ; restore context
    movwf   STATUS
    swapf  W_TEMP,F
    swapf  W_TEMP,W

;--------------------- old restore context ---------------------------
;    btfsc  STATUS,RP1
;    bsf    STATUS,RP0
    retfie                  ; exit re-enable interrupts

;----------------------------------------------------------------
; InitRX
;
; receiver state machine init
; clear 16 bit extended timer
;
; InitRX
    clrf    RFState     ; init receiver
    clrf    RXFlags
    movlw   1
    movwf   RFSkip     ; no delays
    clrf    RFsamp
    movlw   STD_TIME   ; init sampling period
    movwf   RFtime
    clrf    XTMRL      ; clear timer
    clrf    XTMRH
    return
APPENDIX C: TABLE SOURCE CODE

;**********************************************************************
;*  Filename:   TABLE.INC
;**********************************************************************
;*  Author:     Lucio Di Jasio
;*  Company:    Microchip Technology
;*  Revision:   Rev 1.00
;*  Date:       06/07/00
;*
;*  EEPROM TABLE Management routines
;*     simple "linear list" management method
;*
;*  Assembled using MPASM v02.40
;**********************************************************************

#define MAX_USER    .16         ; max number of TX that can be learned
#define EL_SIZE      .8         ; single record size in bytes

CBLOCK
    XF          ; function codes and 4 msb of serial number
    EHopHi      ; last value of sync counter (from EEPROM)
    EHopLo
    LastHop     ; last code for resync
    MFlags
ENDC

#define     Flag_MFull  MFlags,0    ; no empty space left in memory
#define     Flag_Found  MFlags,1    ; search was successful

.gdx

;------------------------------------------------------------
; Table structure definition:
; the EEPROM is filled with an array of MAX_USER user records
; starting at address 0000
; each record is EL_SIZE byte large and contains the following fields:
; EEPROM access is in 16 bit words for efficiency
;
;   DatoHi  DatoLo  offset
; +-------+-------+-------+
; | XF    | IDLo  |  0    XF contains the function codes (buttons) used during learning
; | IDHi  | IDMi  |  +2   IDHi IDMi IDLo contain the 24 LSB of the Serial Number
; | HopHi | HopLo |  +4   sync counter
; | HopHi2| HopLo2|  +6   second copy of sync counter for integrity checking
; +-------+-------+

; NOTE a function code of 0f0 (seed transmission) is considered
; invalid during learning and is used here to a mark location free
;
;------------------------------------------------------------

; FIND Routine
;
; search through the whole table the given a record whose ID match
;
; INPUT:
;     IDHi, IDMi, IDLo,   serial number to search
;
; OUTPUT:
;     IndHi, IndLo        address of record (if found)
;     Flag_Found          set if matching record found
;
; USES:
; W, Count,
; Find
bcf Flag_Found ; init flag
clr IndHi ; init pointer
clr IndLo

FindL
call RDword ; read first Word
movf DatoHi,W
movwf XF ; function code
xorlw 0f0 ; check if 1111xxxx
andlw 0f0
btfsc STATUS,Z
goto FindNext ; unused location

Comp2
movf DatoLo,W ; compare IDLo
xorwf IDLo,W
btfss STATUS,Z
goto FindNext ; fail

call RDnext ; read next word
movf DatoLo,W ; compare IDHi,IDMi
xorwf IDMi,W
btfss STATUS,Z
goto FindNext ; fail
movf DatoHi,W
xorwf IDHi,W ; fail
btfsc STATUS,Z
goto FoundMatch ; match

FindNext
movlw EL_SIZE ; skip to next entry
addwf IndLo,F
btfsc STATUS,C ; carry
incf IndHi,F

; check if end of table reached
movlw HIGH(EL_SIZE * MAX_USER)
xorwf IndHi,W
btfss STATUS,Z
goto FindL

movlw LOW(EL_SIZE * MAX_USER)
xorwf IndLo,W
btfss STATUS,Z
goto FindL

goto FindExit ; table end reached

FoundMatch
bsf Flag_Found ; success
call RDnext ; read HopHi/Lo
movf DatoHi,W
movwf EHopHi ; into EHopHi/Lo
movf DatoLo,W
movwf EHopLo

FindExit
return

;------------------------------------------------------------
; INSERT Routine
; search through the whole table for an empty space
;
; INPUT:
; none
;
; OUTPUT:
; IndHi, IndLo address of empty record
; Flag_MFull set if no empty space found
;
; USES:
; W, Count
;
Insert
    bsf Flag_MFull ; init flag
    clrf IndHi ; init pointer
    clrf IndLo

InsertL
    call EDword ; read first Word
    movf DatoHi,W
    movwf XF ; function code
    xorlw 0f0 ; check if 1111xxxx
    andlw 0f0
    btfsc STATUS,Z
    goto InsertFound

InsertNext
    movlw EL_SIZE ; skip to next entry
    addwf IndLo,F
    btfsc STATUS,C ; carry
    incf IndHi,F

; check if end of table reached
    movlw HIGH(EL_SIZE * MAX_USER)
    xorwf IndHi,W
    btfss STATUS,Z
    goto InsertL
    movlw LOW(EL_SIZE * MAX_USER)
    xorwf IndLo,W
    btfss STATUS,Z
    goto InsertL
    goto InsertExit ; table end reached

InsertFound
    bcf Flag_MFull ; success

InsertExit
    return

;------------------------------------------------------------
; Function IDWrite
; store IDHi,Mi,Lo + XF at current address
; INPUT:
; IndHi, IndLo point to record + offset 0
; IDHi, IDMi, IDLo Serial Number
; XF function code
; OUTPUT:
; IndHi, IndLo point to record + offset 0
; USES:
; as per I2C
;
IDWrite
    btfss Flag_Learn ; Guard statement: check if we are
    goto IDWriteE ; in Learn Mode
movf XF,W ; copy XF and IDLo
movwf DatoHi
movf IDLo,W
movwf DatoLo
call WRword ; write at Ind+00
movlw 2 ; move to offset +2
addwf IndLo,F
btfsc STATUS,Z ; carry
incf IndHi,F ;
movf IDHi,W ; copy IDHi IDMl
movwf DatoHi
movf IDMi,W
movwf DatoLo
call WRword ; write ad Ind+2
movlw 2 ; move back to offset 0
subwf IndLo,F
btfss STATUS,C ; borrow
decf IndHi,F ;
IDWriteE
return

;------------------------------------------------------------
; Function HopUpdate
; update sync counter of user record at current location
; INPUT:
;   IndHi, IndLo record + offset 0
;   HopHi, HopLo current sync counter
; OUTPUT:
;   none
; USES:
;   as per I2C
;
HopUpdate
btfss Flag_HopOK ; Guard statement
goto HopUpdateE ; new valid sync value received
movlw .4 ; move to offset +4
addwf IndLo,F
btfsc STATUS,Z ; carry
incf IndHi,F ;
movf HopHi,W ;
movwf DatoHi
movf HopLo,W
movwf DatoLo ; write at offset +4
call WRword ; update
movlw 2 ; move to offset +6
addwf IndLo,F
btfsc STATUS,Z ; carry
incf IndHi,F ;
movf HopHi,W ;
movwf DatoHi
movf HopLo,W
movwf DatoLo ; write at offset +4
call WRword ; update
movlw 6 ; move back to offset 0
subwf IndLo,F
btfss STATUS,C ; borrow
decf IndHi,F ;
HopUpdateE
return

;------------------------------------------------------------
; Function ClearMem
; mark all records free
; INPUT:
; OUTPUT:
; USES:
;
ClearMem
clrf IndHi       ; start at address 0000
clrf IndLo

ClearMemL
movlw 0ff       ; XF = 0FF
movwf DatoHi    ; restore ad 0fff
movwf DatoLo
call WRword

ClearNext
movlw EL_SIZE    ; goto NEXT record
addwf IndLo,F
btfsc STATUS,Z  ; carry
incf IndHi,F

; check if end of table reached
movlw HIGH(EL_SIZE * MAX_USER)
xorwf IndHi,W
btfsc STATUS,Z ; not yet continue
movlw LOW(EL_SIZE * MAX_USER)
xorwf IndLo,W
btfss STATUS,Z
go to ClearMemL ; not yet continue

ClearMemEx
return
APPENDIX D: MEM-62X SOURCE CODE

;******************************************************************************
;*  Filename:   mem-62x.INC
;******************************************************************************
;*  Author:     Lucio Di Jasio
;*  Company:    Microchip Technology
;*  Revision:   Rev 1.00
;*  Date:       06/07/00
;*
;*  Assembled using MPASM v02.40
;******************************************************************************
;*  PIC16CE62x mid-range (14 bit core) version
;*  NOTE:
;*          2) All timing is based on a reference crystal frequency of 4 MHz
;*              which is equivalent to an instruction cycle time of 1 µs.
;*          3) Address and literal values are read in hexadecimal unless
;*              otherwise specified.
;******************************************************************************

errorlevel -302

include "fl62xinc.asm" ; standard library for PIC16CE62X

CBLOCK
   IndHi       ; memory address pointer
   IndLo
   DatoHi     ; read/write buffer to EEPROM
   DatoLo
ENDC

;******************************************************************************
;*      RDword
;*      read one word from serial EEPROM device
;*
;*      Input   :       IndHi/LO
;*      Output  :       DatoLo/Hi = data read from serial EEPROM
;******************************************************************************

RDword
   call    ENABLE_EEPROM
   movf    IndLo,W
   movwf    READDR
   call    READ_RANDOM
   BANK0
   movf    REDATA,W
   movwf    DatoHi
   goto    RDnextbyte
RDnextbyte
   call    READ_CURRENT
   BANK0
   movf    REDATA,W
   movwf    DatoHi
   goto    RDnextbyte
RDnext
   call    READ_CURRENT
   BANK0
   movf    REDATA,W
   movwf    DatoHi
   goto    RDnextbyte

;******************************************************************************
;*      WRword
;*      write one word to EEPROM device
;******************************************************************************
AN742

;*      Input :  DatoLo/Hi = data to be written
;*      Output :  IndHi/LO= EEPROM data address
;*****************************************************************************

WRword
    call    ENABLE_EEPROM
    movf    IndLo,W
    movwf    EEADDR
    movf    DatoHi,W
    movwf    EEDATA
    call    WriteB    ; write DatoHi first (big endian)
    incf    IndLo,W
    movwf    EEADDR
    movf    DatoLo,W    ; write DatoLo second
    movwf    EEDATA

WriteB
    call    WRITE_BYTE
    BANK0

; 5 ms wait loop for writing time
    clr    DatoHi    ; use it as wait loop counter

WriteWL
    call    delay20    ; wait 20 cycles = 20 µs
    decfsz  DatoHi,F
    goto    WriteWL
    return

;*****************************************************************************
;*      ENABLE_EEPROM
;*      switch on Vdd for internal EEPROM
;*****************************************************************************

ENABLE_EEPROM
    BANK1
    BSF    EEINTF,EEVDD    ; turn on EE data module
    return

errorlevel +302
APPENDIX E: FL62XINC SOURCE CODE

#define TWENTYMHZ

Program: FL62xINC.ASM
Revision Date: V1.00 30 June 1998 Adapted to 16CE62x parts

PIC16CE62x EEPROM communication code. This code should be linked in
with the application. While this code is very similar to the FLASH62X
code, this file assumes the file registers are in page 1 and hence
it doesn’t need to keep switching between register page 0 and 1. This
saves 19 EEPROM locations.

These routines provide the following functionality:
write byte random address
read byte random address
read byte next address
read sequential is not supported.

If the operation is successful, bit 7 of PC_OFFSET will be set, and
the functions will return W=1. If the memory is busy with a write
cycle, it will not ACK the command. The functions will return with
bit 7 of PC_OFFSET cleared and and W will be set to 0.

Based on Franco code.

VERY IMPORTANT! This code must reside on the lower half of
code page (address 0-FF).

This provides users with highly compressed assembly code for
communication between the EEPROM and the Microcontroller, which
leaves a maximum amount of code space for the core application.

Conditional assembly delays are included to meet standard mode timing
specs. For 4Mhz, define FOURMHZ at top of file. For 10 Mhz, define TENMHZ.

and low voltage. Applications running at slower clock rates and those
operating within 4.5-5.5V may be able to remove some of the NOPs/Delay calls.

This code is specifically written for the interface hardware of the
16C623/624/625 parts. See AN571 for the unmodified routines.

***************************************************************************
***************************  EEPROM Subroutines  **************************
***************************************************************************
Communication for EEPROM based on I2C protocol, with Acknowledge.

Byte_Write: Byte write routine
Inputs: EEPROM Address EEADDR
EEPROM Data EEDATA
Outputs: Return 01 in W if OK, else return 00 in W

Write_Page: Page write routine - writes up to 8 bytes at a time
Inputs: FSR points to beginning of RAM buffer.
        W number of bytes to write
        EEPROM Address EEADDR
EEPROM Data EEDATA
Outputs: Return 01 in W if OK, else return 00 in W

Read_Current: Read EEPROM at address currently held by EE device.
Inputs: NONE
Outputs: EEPROM Data EEDATA
Return 01 in W if OK, else return 00 in W
; Read Random: Read EEPROM byte at supplied address
; Inputs: EEPROM Address EEADDR
; Outputs: EEPROM Data EEDATA
; Return 01 in W if OK, else return 00 in W
;
; Note: EEPROM subroutines will set bit 7 in PC_OFFSET register if the
; EEPROM acknowledged OK, else that bit will be cleared. This bit
; can be checked instead of referring to the value returned in W
;
; EEinterface file registers (EEAddress, EEDATA) are in common ram.
; EEINTF file register is on Register Page 1. Upon exit, Register
; page is set to 0.

***************************************************************************

OPERATION:

Byte Write:

load EEADDR and EEDATA
then CALL WRITE_BYTE

Page Write:

Load EEADDR
Load FSR with address of 1st byte to transfer
Load W with number of bytes to transfer (8 bytes max)
then CALL WRITE_PAGE

Read Random:

Load EEADDR
then CALL READ_RANDOM
data read returned in EEDATA

Read Current
no setup necessary
CALL READ_CURRENT
data read returned in EEDATA

Page Read:

Load EEADDR with address within EE to read
Load FSR with address of buffer
Load W with number of bytes to transfer
then CALL READ_PAGE

***************************************************************************

***************************  Variable Listing ****************************
***************************************************************************

OK EQU 01H
NO EQU 00H

EE_OK EQU 07H ; Bit 7 in PC_OFFSET used as OK flag for EE

; These file registers can be moved, however they need to reside within
; the shared memory in the last 16 bytes of the register page. This
; provides access to the variables and the EEINTF register on page 1
; without constantly shifting register pages.

EEADDR EQU 0x78 ; EEPROM Address
EEDATA EQU 0x79 ; EEPROM Data
EEBYTE EQU 0x7A ; Byte sent to or received from
                 ; EEPROM (control, address, or data)
bytecount EQU 0x7B ; # of bytes to write
COUNTER EQU 0x7C ; Bit counter for serial transfer
PC_OFFSET EQU 0x7D; PC offset register (low order 4 bits),
                 ; value based on operating mode of
                 ; Also, bit 7 used for EE_OK flag
;*************** Set up EEPROM control bytes ***********************
;****************************************************************************
READ_CURRENT
   MOVLW   B’10000100’   ; PC offset for read current addr.  EE_OK bit7='1'
   MOVWF   PC_OFFSET     ; Load PC offset
   BSF STATUS,RP0 ; set register page 1
   GOTO    INIT_READ_CONTROL

WRITE_BYTE
   MOVLW   B’10000000’   ; PC offset for write byte.  EE_OK: bit7 = '1'
   GOTO    INIT_WRITE_CONTROL

WRITE_PAGE
   movwf   bytecount ; save off number of bytes to send
   MOVLW   B’10000000’   ; PC offset for read page.  EE_OK bit = 1
   goto INIT_WRITE_CONTROL

READ_PAGE
   movwf   bytecount ; save off number of bytes to send
   MOVLW   B’10001010’   ; PC offset for read page.  EE_OK bit = 1
   goto INIT_WRITE_CONTROL

READ_RANDOM
   MOVLW   B’10000011’   ; PC offset for read random.  EE_OK: bit7 = '1'

INIT_WRITE_CONTROL
   MOVWF   PC_OFFSET     ; Load PC offset register, value preset in W
   MOVLW   B’10100000’   ; Control byte with write bit, bit 0 = '0'

START_BIT
   BSF STATUS,RP0 ; set register page 1
   BCF     EEINTF,EESDA  ; Start bit, EESDA and EESCL preset to '1'

;****** Set up output data (control, address, or data) and counter ******
;***************************************************************************
PREP_TRANSFER_BYTE
   MOVWF   EEBYTE        ; Byte to transfer to EEPROM already in W
   MOVLW   .8            ; Counter to transfer 8 bits
   MOVWF   COUNTER

;************  Clock out data (control, address, or data) byte ************
;***************************************************************************
OUTPUT_BYTE
 ifdef  FOURMHZ
   NOP
 endif
 ifdef  TENMHZ
   call    delay8     ; Tsu:sta, Thigh: 4700 nS (add 6 cycles at 10 Mhz)
 endif
 ifdef  TWENTYMHZ
   call    delay16    ; Tsu:sta, Thigh: 4700 nS (add 6 cycles at 10 Mhz)
 endif
 RLF     EEBYTE, F     ; Rotate left, high order bit into carry bit
 BCF     EEINTF, EESCL  ; Set clock low during data set-up
 BCF     EEINTF, EESDA  ; Set data low, if rotated carry bit is
 SKPNC    ; a '1', then:
 BSF     EEINTF, EESDA  ; reset data pin to a one, otherwise leave low
 ifdef  FOURMHZ
   NOP
 endif
 ifdef  TENMHZ
   call    delay8     ; Tlow 4700 nS (add 6 cycles at 10 Mhz)
 endif
ifdef TWENTYMHZ
call delay16 ; Tlow 4700 nS (add 6 cycles at 10 Mhz)
endif
BSF EEINTF, EESCL ; clock data into EEPROM
DECFSZ COUNTER, F ; Repeat until entire byte is sent
GOTO OUTPUT_BYTE
ifdef FOURMHZ
NOP ; Needed to meet Timing (Thigh=4000nS)
endif
ifdef TENMHZ
call delay8
endif
ifdef TWENTYMHZ
call delay16 ; Tlow 4700 nS (add 6 cycles at 10 Mhz)
endif

;**************************  Acknowledge Check ****************************
;***************************************************************************
BCF EEINTF, EESCL ; Set EESCL low, 0.5us < ack valid < 3us
ifdef FOURMHZ
NOP ; Needed to meet Timing (Tlow= 4700nS)
endif
ifdef TENMHZ
goto $+1
endif
ifdef TWENTYMHZ
call delay4
endif
BSF EEINTF, EESDA ; set data line high to check for acknowledge
ifdef FOURMHZ
GOTO $+1
endif
ifdef TENMHZ
call delay6 ; Necessary for EESCL Tlow at low voltage, (4.7us)
endif
ifdef TWENTYMHZ
call delay12
endif

BSF EEINTF, EESCL ; Raise EESCL, EEPROM acknowledge still valid
ifdef FOURMHZ
NOP ; Tsu:dat (allow time for ack setup)
endif
ifdef TENMHZ
call delay4
endif
ifdef TWENTYMHZ
call delay8
endif
BTFSC EEINTF, EESDA ; Check EESDA for acknowledge (low)
BCF PC_OFFSET, EE_OK ; If EESDA not low (no ack), set error flag
ifdef TENMHZ
call delay4
endif
ifdef TWENTYMHZ
call delay8
endif
BCF EEINTF, EESCL ; Lower EESCL, EEPROM release bus
BTFSS PC_OFFSET, EE_OK ; If no error continue, else stop bit
GOTO STOP_BIT

;***** Set up program counter offset, based on EEPROM operating mode *****
;***************************************************************************
STATEMACHINE
movlw   HIGH(GTABLE)
movwf   PCLATH
MOVF PC_OFFSET,W
ANDLW B'00001111'
ADDWF PCL, F

;跳转表
GTABLE
GOTO INIT_ADDRESS   ;PC offset=0, write control done, send address
GOTO INIT_WRITE_DATA ;PC offset=1, write address done, send data
GOTO STOP_BIT       ;PC offset=2, write done, send stop bit
GOTO INIT_ADDRESS   ;PC offset=3, write control done, send address
GOTO INIT_READ_CONTROL ;PC offset=4, send read control
GOTO READ_BIT_COUNTER ;PC offset=5, set counter and read byte
GOTO STOP_BIT       ;PC offset=6, read done, send stop bit
GOTO INIT_ADDRESS   ;PC offset=7, write control done, send address
GOTO INIT_WRITE_PAGE_DATA ;PC offset=8, write address done, send data
GOTO STOP_BIT       ;PC offset=9, write done, send stop bit
GOTO INIT_ADDRESS   ;PC offset=A, write control done, send address
GOTO INIT_READ_PAGE_CONTROL ;PC offset=B, write address done, send data
GOTO READ_PAGE_BIT_COUNTER ;PC offset=C, set counter and read byte

GTABLE_END

if HIGH(GTABLE) != HIGH(GTABLE_END)
    error "jump table must fit all in the same page"
endif

;**********  Initalize EEPROM data (address, data, or control) bytes  *****
;***************************************************************************
INIT_ADDRESS
INCF PC_OFFSET, F ; Increment PC offset to 2 (write) or to 4 (read)
MOVF EEADDR,W     ; Put EEPROM address in W, ready to send to EEPROM
GOTO PREP_TRANSFER_BYTE

INIT_WRITE_DATA
INCF PC_OFFSET, F ; Increment PC offset to go to STOP_BIT next
MOVF EEDATA,W     ; Put EEPROM data in W, ready to send to EEPROM
GOTO PREP_TRANSFER_BYTE

INIT_WRITE_PAGE_DATA
DECFSZ bytecount,f ; count byte tx'd
GOTO $+2 ;
INCF PC_OFFSET, F ; Increment PC offset to go to STOP_BIT next
MOVF INDF,W          ; Put EEPROM data in W, ready to send to EEPROM
INCF FSR,P          ; bump pointer
GOTO PREP_TRANSFER_BYTE

INIT_READ_CONTROL
BSF EEINTF,EESCL ; Raise EESCL
BSF EEINTF,EESDA ; raise EESDA
INCF PC_OFFSET, F ; Increment PC offset to go to READ_BIT_COUNTER next
MOVLW B'10100001' ; Set up read control byte, ready to send to EEPROM
GOTO START_BIT ; bit 0 = '1' for read operation

INIT_READ_PAGE_CONTROL
BSF EEINTF,EESCL ; Raise EESCL
BSF EEINTF,EESDA ; raise EESDA
INCF PC_OFFSET, F ; Increment PC offset to go to READ_BIT_COUNTER next
MOVLW B'10100001' ; Set up read control byte, ready to send to EEPROM
GOTO START_BIT ; bit 0 = '1' for read operation

;***************************************************************************
; Read EEPROM data  ****************************************************
READ_PAGE_BIT_COUNTER
BSF EEINTF,EESDA ; set data bit to 1 so we're not pulling bus down.
NOP
BSF   EEINTF,EESCL
MOVLW .8           ; Set counter so 8 bits will be read into EEDATA
MOVF   COUNTER

READ_BYTE_RPC
#ifdef  TENMHZ
call    delay6
#endif
#ifdef  TWENTYMHZ
call    delay12
#endif
BSF   EEINTF,EESCL ; Raise EESCL, EESDA valid.  EESDA still input from ack
#ifdef  TENMHZ
call    delay6
#endif
#ifdef  TWENTYMHZ
call    delay12
#endif
BTFSS   EEINTF,EESDA ; Check if EESDA = 1
CLRC                 ; if EESDA not = 1 then clear carry bit
RLF     EEDATA, F    ; rotate carry bit (=EESDA) into EEDATA;
BCF     EEINTF,EESCL ; Lower EESCL
BSF     EEINTF,EESDA ; reset EESDA
DECFSZ  COUNTER, F   ; Decrement counter
GOTO    READ_BYTE_RPC    ; Read next bit if not finished reading byte

movf     EEDATA,w
movwf    INDF          ; write data to buffer
incf     FSR,f         ; increment buffer pointer
declfsz   bytecount,f
goto     SEND_ACK      ; skip next 2 instructions
SEND_ACK
BCF     EEINTF,EESDA ; Send an ACK (More reads to come)
BSF     EEINTF,EESCL;
NOP
BSF     EEINTF,EESCL
GOTO    READ_PAGE_BIT_COUNTER
SEND_NAK
BSF     EEINTF,EESDA ; Send an ACK (More reads to come)
BSF     EEINTF,EESCL;
NOP
BSF     EEINTF,EESCL
GOTO    STOP_BIT       ; skip next 2 instructions

; end read page bit control

READ_BIT_COUNTER
BSF     EEINTF,EESDA ; set data bit to 1 so we’re not pulling bus down.
NOP
BSF     EEINTF,EESCL
MOVLW .8           ; Set counter so 8 bits will be read into EEDATA
MOVF   COUNTER

READ_BYTE_RBC
#ifdef  TENMHZ
call    delay6
#endif
#ifdef  TWENTYMHZ
call    delay12
#endif
BSF     EEINTF,EESCL ; Raise EESCL, EESDA valid.  EESDA still input from ack
SETC                 ; Assume bit to be read = 1
#ifdef TENMHZ
    call delay6
#endif
#ifdef TWENTYMHZ
    call delay12
#endif
ifdef
    BTFSS EEINTF, EESDA ; Check if EESDA = 1
    CLR C ; if EESDA not = 1 then clear carry bit
    RLF EEDATA, F ; rotate carry bit (=EESDA) into EEDATA;
    BCF EEINTF, EESCL ; Lower EESCL
    BSF EEINTF, EESDA ; reset EESDA
    DECFSZ COUNTER, F ; Decrement counter
    GOTO READ_BYTE_RBC ; Read next bit if not finished reading byte
    BSF EEINTF, EESCL
    NOP
    BCF EEINTF, EESCL
#endif
;******************  Generate a STOP bit and RETURN  **********************
;***************************************************************************
STOP_BIT
    BCF EEINTF, EESDA ; EESDA=0, on TRIS, to prepare for transition to ‘1’
    BSF EEINTF, EESCL ; EESCL = 1 to prepare for STOP bit
#ifdef FOURMHZ
    call delay4 ; wait 4 cycles Tsu:sto (4.7 us)
#endif
#ifdef TENMHZ
    call delay10
#endif
ifdef TWENTYMHZ
    call delay20
endif
    BSF EEINTF, EESDA ; Stop bit, EESDA transition to ‘1’ while EESCL high
    BCF STATUS, RP0
    BTFSS PC_OFFSET, EE_OK ; Check for error
    RETLW NO ; if error, send back NO
    RETLW OK ; if no error, send back OK
#endif
ifdef TWENTYMHZ
    delay20 goto delay18
    delay18 goto delay16
    delay16 goto delay14
    delay14 goto delay12
    delay12 goto delay10
    delay10 goto delay8
    delay8 goto delay6
    delay6 goto delay4
    delay4 return
#endif
ifdef TENMHZ
    ; delay function. Wait a number of cycles.
    delay10 goto delay8
    delay8 goto delay6
    delay6 goto delay4
    delay4 return
#endif
ifdef FOURMHZ
    delay4 return
#endif
;***************************************************************************
;************************  End EEPROM Subroutines  **************************
APPENDIX F: KEYGEN SOURCE CODE

;******************************************************************************
;* Filename: KEYGEN.INC
;******************************************************************************
;* Author: Lucio Di Jasio
;* Company: Microchip Technology
;* Revision: Rev 1.00
;* Date: 06/07/00
;*
;* Normal Key generation Algorithm
;* refer to Secure Data Products Handbook TB003
;* for an introduction to KmsLo® and Key generation Algorithms
;*
;* Assembled using MPASM v02.40
;******************************************************************************

CBLOCK
    DKEY0       ; decryption key LSB first
    DKEY1
    DKEY2
    DKEY3
    DKEY4
    DKEY5
    DKEY6
    DKEY7
    SEED0       ; SEED temp for Serial Number (Normal Learn)
    SEED1
    SEED2
    SEED3
    HOPT0       ; temp for encrypted message during Key construction
    HOPT1
    HOPT2
    HOPT3
    SKEY0       ; temp for half key during key generation
    SKEY1
    SKEY2
    SKEY3
ENDC

#include "fastdec.inc"      ; Keeloq decrypt routine

;-------------------------------------------------------------------------------
; NormalKeyGen
; first check if output is active and Serial Number is the same
movf    COut,F          ; test if Output timer is still going
btfsc   STATUS,Z
    goto    Generate
movf    IDLo,W
xorwf   SEED0,W         ; compare LSB of Serial Number IDLo
btfss   STATUS,Z
    goto    Generate
movf    IDMi,W
xorwf   SEED0,W
    goto    Generate
movf    IDHi,W
indf};
xorwf SEED0,W
btfss STATUS,Z
  goto Generate
movf CSR7,W; compare lower nibble of MSB
xorwf SEED0,W
andlw 0f
btfss STATUS,Z
  goto Generate
; key generation is not required, last computed key (DKEY) is still valid!
goto NormalKeyGenE; exit

; key generation is required
Generate
  call SaveHOP; save received hopping code during key gen
  call SaveSEED; prepare the SEED (= Serial Number)

; generate low half of the key
  call LoadSEED; SEED value + 020
  movlw 020
  iorwf CSR3,F
  call LoadManufacturerCode
  call Decrypt; generate

; save first half of the key for later
  movf CSR0,W
  movwf SKEY0
  movf CSR1,W
  movwf SKEY1
  movf CSR2,W
  movwf SKEY2
  movf CSR3,W
  movwf SKEY3

; generate most significant half (32bits) of the Key
  call LoadSEED; SEED value + 060
  movlw 060
  iorwf CSR3,F
  call LoadManufacturerCode
  call Decrypt; generate

; join the two half of the key
  movf SKEY0,W
  movwf DKEY0
  movf SKEY1,W
  movwf DKEY1
  movf SKEY2,W
  movwf DKEY2
  movf SKEY3,W
  movwf DKEY3
  movf CSR0,W
  movwf DKEY4
  movf CSR1,W
  movwf DKEY5
  movf CSR2,W
  movwf DKEY6
  movf CSR3,W
  movwf DKEY7
  call LoadHOP; reload the encrypted message

NormalKeyGenE
  return

;----------------------------------------------------------------------
; SaveHOP
;
; saves the received Hopping Code in a temp during Key Generation phase
;
SaveHOP
    movf CSR0,W
    movwf HOPT0
    movf CSR1,W
    movwf HOPT1
    movf CSR2,W
    movwf HOPT2
    movf CSR3,W
    movwf HOPT3
    return
;
; LoadHOP
;
; restores Hopping Code in decryption buffer
;
LoadHOP
    movf HOPT0,W
    movwf CSR0
    movf HOPT1,W
    movwf CSR1
    movf HOPT2,W
    movwf CSR2
    movf HOPT3,W
    movwf CSR3
    return

; SaveSEED
;
; Serial Number is used as SEED in Normal Learn
;
SaveSEED
    movf IDLo,W ; LSB
    movwf SEED0
    movf IDMi,W
    movwf SEED1
    movf IDHi,W
    movwf SEED2
    movf CSR7,W ; MSB (only lower nibble)
    andlw 0f
    movwf SEED3
    return
;
; LoadSEED
;
; Loads the SEED value into the decryption buffer CSR0..3
;
LoadSEED
    movf SEED0,W
    movwf CSR0
    movf SEED1,W
    movwf CSR1
    movf SEED2,W
    movwf CSR2
    movf SEED3,W
    movwf CSR3
    return

; Load Manufacturer Code

LoadManufacturerCode

movlw   001     ; MC = 0123456789ABCDEF
movwf   DKEY7   ; MSB
movlw   023
movwf   DKEY6
movlw   045
movwf   DKEY5
movlw   067
movwf   DKEY4
movlw   089
movwf   DKEY3
movlw   0AB
movwf   DKEY2
movlw   0CD
movwf   DKEY1
movlw   0EF
movwf   DKEY0   ; LSB
return

;------------------------------------------------------------
; verification of decryption
;
; INPUT:
;          DOK                 discrimination bits and function codes after decrypt
;          IDHi,IDM1,IDLo      24 bit of serial number from plain text
;          S0..S3              function codes from plain text
; OUTPUT:
;          Z   set if decrypt check OK
;
 DecCHK

movf    DisLo,W         ; compare discrimination bits
xorwf   IDLo,W          ; with 10 lsb from serial number
btfss   STATUS,Z        ; NZ if bad
return

movf    DOK,W           ; 2 MSB of discrimination word
xorwf   IDM1,W
andlw   3
btfss   STATUS,Z        ; NZ if bad
return

movf    DOK,W           ; check function codes
xorwf   CSR7,W          ; against plain text copy
andlw   0f0
return                  ; Z if OK

;-----------------------------------------------------------------
; HopCHK
; verification of sync counter
; N.B. sync counter is store in EEPROM twice for safety
; should the two copies not match (corrupted memory)
; activate a 2^ chance for resync
;
HopCHK

bcf     Flag_HopOK      ; clear flags
bcf     Flag_Same       ;

btfss   Flag_2C         ; 2^ chance (resync) already set
goto    HopCHK2         ;

; 2^ chance, verify new code is just previous one +1
;
movf LastHop,W          ; compare store value
xorwf HopLo,W           ; with the new one
BZ HopOK                ; if match ... resync

HopCHK2

movf EHopHi,W           ; check EEPROM integrity
xorwf DatoHi,W
BNZ ReqResync           ; give a chance to resync
movf EHopLo,W
xorwf DatoLo,W
BNZ ReqResync

; memory read fine, make a 16 bit comparison of Sync counter
; with previous counter value stored in EEPROM
;
; verify that new > old
; specifically if the difference is:
; 0     -> Flag_HopOK + Flag_Same
; 1..15 -> Flag_HopOK, open window
; 16..32768 -> Flag_2C, require resync
; > 32768 (negative values) discard!

VerSync

movf EHopLo,W           ; DatoHi/Lo = HopHi/Lo-EHopHi/Lo
subwf HopLo,W           ; 16 bit subtraction
movwf DatoLo
btfss STATUS,C
incf EHopHi,F           ; borrow
movf EHopHi,W
subwf HopHi,W
movwf DatoHi
btfss STATUS,C
goto Fail               ; result is <0 -> discard
btfss STATUS,Z
goto ReqResync          ; if >256 req. resync 2^ chance

; verify in open window 1..16
movlw .16
subwf DatoLo,W
btfsc STATUS,C
goto ReqResync          ; if >=16 req resync 2^ chance

; verify if 0 : same code as previous
movf DatoLo,F
btfsc STATUS,Z
bsf Flag_Same           ; signal diff is 0 means same code as previous
goto HopOK               ; 0<X<16 open window

ReqResync

bsf Flag_2C             ; signal req. resync
incf HopLo,W
movwf LastHop           ; store pre inc. sync value
return                   ; return failure

HopOK

bcf Flag_2C             ; no 2nd chance needed
bsf Flag_HopOK          ; signal code is in sync
return                   ;
Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.