AN827

Using KEELOQ® to Validate Subsystem Compatibility

Author: *Lucio Di Jasio*
Microchip Technology Inc.

OVERVIEW

Proper modular design can make a system more flexible, easier to maintain and update, and most of all, can help get the design completed faster. In fact, modules or subsystems design can happen in parallel, and can be out-sourced when convenient. Subsystems can also be conveniently replaced during the life of an application to simplify maintenance and allow upgrades. In many situations though, it can be extremely important to verify compatibility of the replacement module in order to ensure the system performs to specifications, maintains quality standards and/or allows safe operation of the whole system. As an example, we offer the case of a mobile phone with its rechargeable battery module. Batteries can be based on different technologies (NiCd, NiMH, Li Ion, etc.) that, in turn, require different charging algorithms. A mismatch would lead to potentially dangerous situations (including as an extreme, fire and explosion of the battery pack) severely compromising the safety of use of the whole product. The situation is different in the case of security applications (such as modular Home Alarm systems) where module validation is required to prevent tampering of the system, theft, and intrusion.

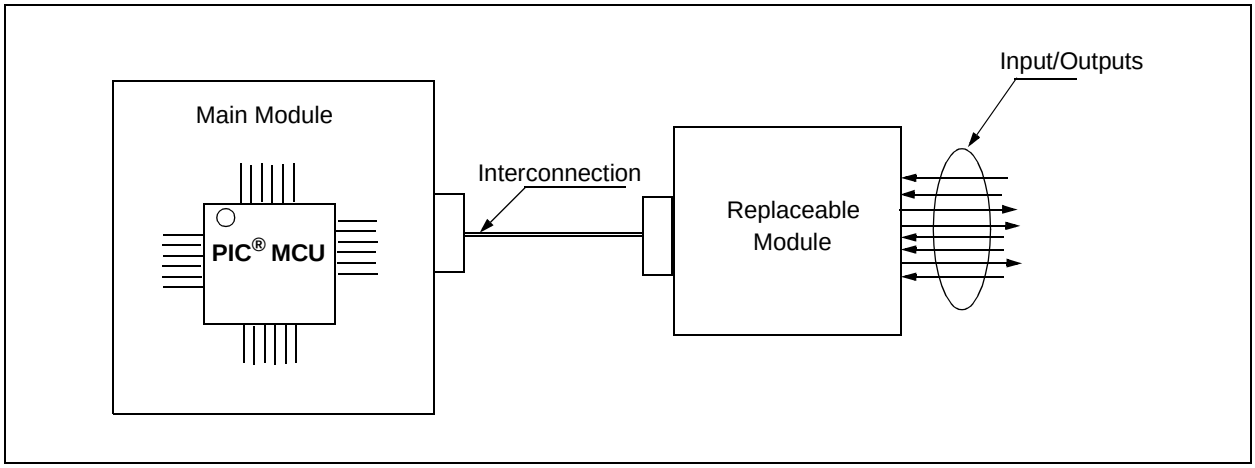
This Application Note concentrates on the use of the HCS410 transcoder to implement a reliable (yet low cost) method for module authentication using the KEELOQ® technology.

A MODULAR SYSTEM

In the following sections, we will consider a simple system composed of two modules: a main module containing a microprocessor, and a replaceable module containing some kind of I/O circuitry. As a design constraint we will assume the replaceable I/O module to be very cost-sensitive. The main module will be able to recognize I/O module replacements and determine if the new modules are compatible. Compatibility will be considered granted if the new modules are "proved" to be manufactured from the same company that manufactures the main module or a licensed third party.

The reader will further note how the Interconnection represents in all respects a third important subsystem, with cost and reliability as the main constraints on its specification. For simplicity, we will consider it to be based on low cost cabling with as few contacts (wires) as possible.

FIGURE 1: A SIMPLE MODULAR SYSTEM



Notice:

This is a non-restricted version of Application Note AN825 which is available under the KEELOQ License Agreement. All Application Notes under the KEELOQ License Agreement are contained in the KEELOQ CD DS51773.

SERIALIZATION OF MODULES

The first step in module production control involves the concept of Serialization. That is, every module produced must contain a **unique** Serial Number to identify it from among the entire production. Serialization is also obviously beneficial for quality control, maintenance programs, etc., although here we specify the Serial Number must be machine-readable. In other words, we require the replaceable module to be able to provide a unique number, in a digital form, upon request over the interconnection system. At any point in time (especially before making use of any of the replaceable module functions) the main module will request this unique number and compare it against a value in the main controller non-volatile memory. The main controller can use this Serial Number to implement various types of policies, such as:

- identifying the valid replacements
- logging its value through the application life
- using it for any sort of documentation of the system history for maintenance and/or warranty checking purposes

An optimal system would also require all replaceable modules to have some read/write non-volatile memory accessible through the same communication channel. Such non-volatile memory could then be used by the manufacturer to supply calibration data for the main processor to adjust the input/output signal conditioning (correcting gains and offsets for example), as well as allow the main board to log usage information.

AUTHENTICATION OF MODULES

The problem with Serialization is in its simplicity. If the serial number is readable by the main module and its value is checked multiple times, then tapping into the interconnection system will easily provide this information. Production of multiple replacement modules with the same Serial Number is trivial. All the modules would appear the same to the main board. It would actually be impossible to detect any change or replacement.

In everyday life, we use IDs and signatures to authenticate individuals with the assumption that they cannot be easily counterfeited or duplicated. But in the digital world, duplicating a Serial Number is almost too easy. It is similar to the instance in a bank where, to get access to our accounts, we were requested to supply a password and we had to say it aloud in front of other people. If the password is fixed, then as soon as we use it, it is compromised.

A solution to the authentication problem, consists in making it possible to **deduce** whether something or somebody knows (or contains) a number (a password or a key) without ever communicating it, or exposing it to cloning.

IDENTIFY FRIEND OR FOE (IFF)

The IFF anti-cloning technique that is used in the following section is simple, yet very effective.

The term IFF is believed to be traced to its first use in the aviation industry, where it was employed in the radar systems to distinguish approaching Friend planes from the enemies (or Foes).

The main IFF concept is based on the use of formulas instead of fixed values during the authentication process. In our simple system, the main processor sends a large random number (usually referred to as the “challenge”) to the replaceable module. The module would apply a convened formula to the challenge value, compute a “response” value and send it back to the main processor. The main processor would compare the returned value with the locally computed value. If the formula used by the replaceable module is the correct one, the main processor finds a match and accepts the new module as compatible or ‘authentic’, as shown in Figure 2.

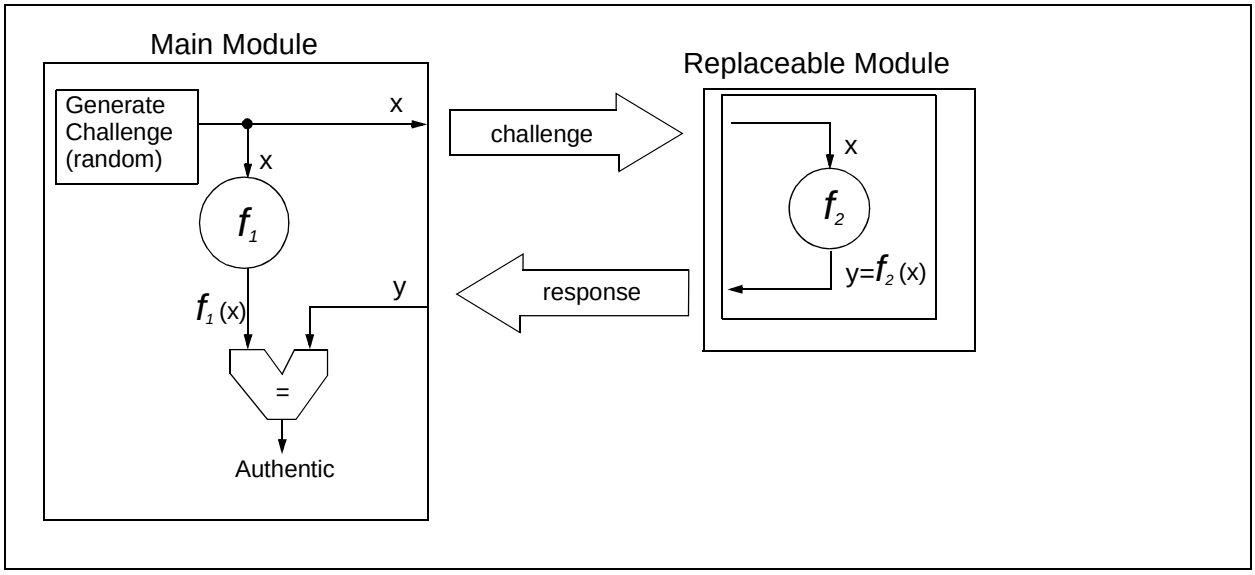
The reader will note how, during the entire transaction, no detail of the formula is ever communicated between the two modules. Tapping into the interconnection system will not reveal any information that could be used in the creation of a duplicate module, since the challenge value is always different, possibly even random.

For increased security, the process can be repeated as many times as required to achieve the requested level of confidence.

Of course there are many factors to be considered that influence the overall security of the system. Among these factors are:

1. the computational complexity of the formula
2. the bit length of the challenge and response
3. the encryption/decryption keys are never exchanged between the two modules

FIGURE 2: IFF OR CHALLENGE AND RESPONSE



KEELOQ - IFF

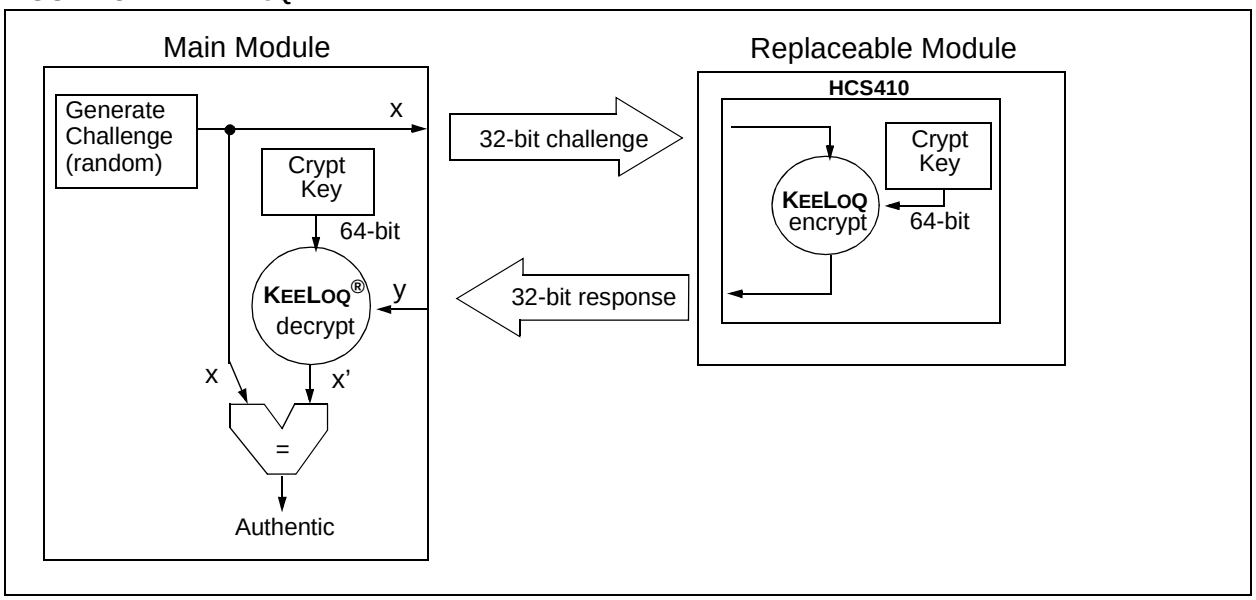
A KEELOQ IFF system adds a little twist to the standard IFF mechanism. The process takes advantage of the symmetrical nature of the KEELOQ block cipher. KEELOQ is a robust, field proven, 64-bit key encryption engine (block cipher) operating on 32-bit (data blocks): challenge and response. The method is illustrated by the following sequence of steps:

1. A 32-bit challenge (x) is generated by the Main Module (optimally as a random value).
2. The challenge is sent to the Replaceable Module.
3. The Replaceable Module authentication device (HCS410) uses the KEELOQ encryption algorithm to generate a 32-bit Response (y) and sends it back.
4. The Main module applies the KEELOQ decryption algorithm to the 32-bit response generating a new 32-bit value (x').
5. The result of the decryption process (x') is compared with the original Challenge (x)
6. If the two values match, the Replaceable module is verified to be compatible and can be used further by the Main module.

If the Replaceable module is not compatible, any attempt to guess the right response value (y) for a given challenge (x) has a probability of $1/2^{32}$ or in other words, one in four billion. The challenge and response process can then be repeated (looping through steps 1 to 6) to increase the verification security as required, effectively extending the equivalent challenge and response length by two, three.. n-times.

The resulting level of security provided by this system would still not be recommended for high-level monetary transactions, where 128-bit key (and higher) systems are considered the norm. However, Embedded Applications in the consumer and industrial field can get an exceptional level of security while achieving an optimal compromise with system cost.

FIGURE 3: KEELOQ® - IFF



THE HCS410 TRANSCODER

For the cost of a single KEELOQ HCS410 transcoder (transponder and encoder) device on the replaceable module, only one I/O line (one wire) in the interconnection, and a few lines of code on the main board processor, the user gets a very high level Authentication system, including serialization and R/W Nonvolatile Memory features.

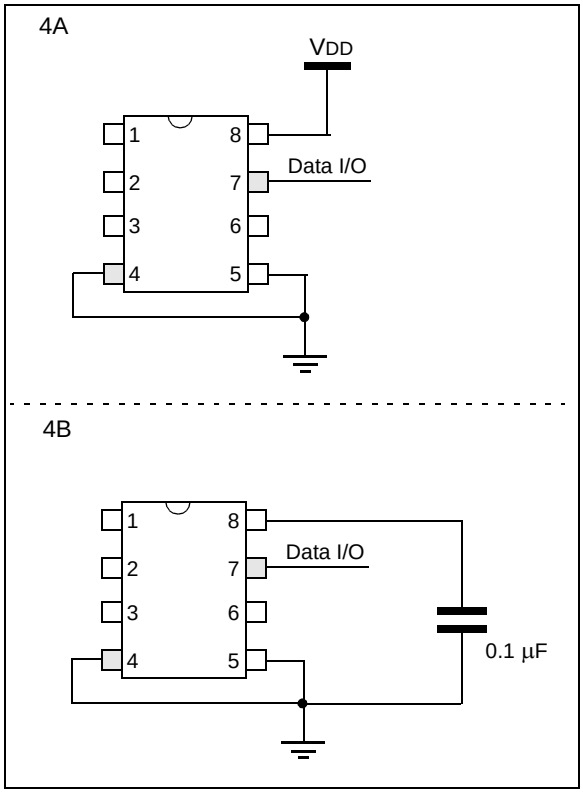
The HCS410 is available in an 8-pin SOIC (or DIP) package and requires no external components to operate to its full capabilities (see Figure 4, A). Its power supply ranges from 2V to 6.3V, with an operating current of less than 300 μA at 6.3V, posing no constraints on the replaceable module power supply design.

From the details provided by the HCS410 Data Sheet [DS40158] we learn that it is designed to operate in three different modes:

- as an Hopping Code encoder
- as a contact-less Transponder (see Microchip literature [DS41116] for WM package details and availability)
- as a wired Authentication device operating in KEELOQ-IFF mode (often referred to in the Data Sheet as the IFF-W mode)

In this application we will employ the HCS410 only in this third mode.

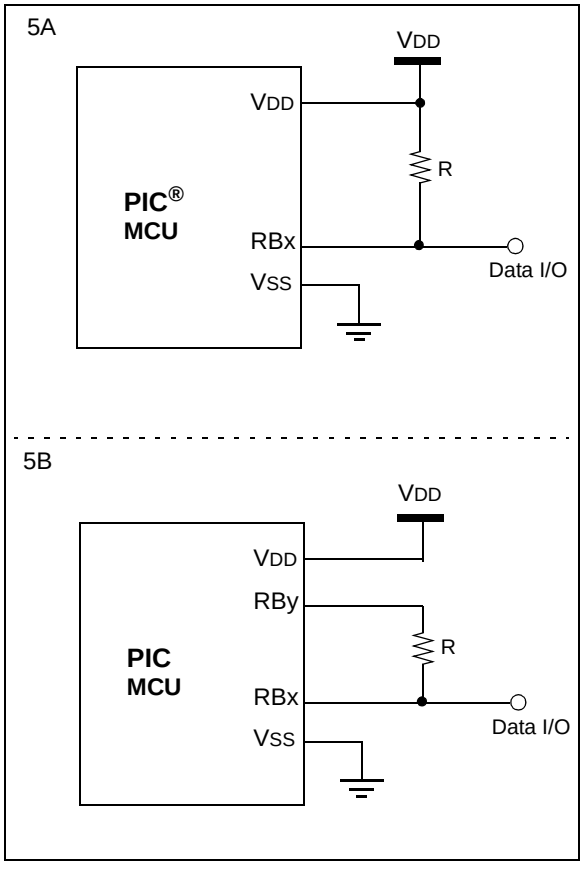
FIGURE 4: HCS410 IFF-W CONNECTIONS



In the configuration illustrated in Figure 4(B), the power supply connection of pin 8 to VDD becomes redundant, as the part derives (internally) its power supply from the same I/O (pin 7 LC0) that is used for the bi-directional communication. In this case adding a 0.1 μF capacitor (tank) on pin 8 is recommended.

The HCS410 Data I/O line is the only added wire required for the system interconnection. Correspondingly, the processor on the main module will dedicate only one extra I/O for communication with the HCS410 (see Figure 5, A). An open collector I/O will be used if available or will be emulated, by switching dynamically the I/O line from the Output low state for a logic zero (pulse), to Input (tri-state) for a logic one condition, with an external pull up resistor. When communication with the HCS410 is not requested (Standby mode), power consumption can be minimized by the main processor controlling the pull-up resistor with a second I/O (RBy) Figure 5, B. When attempting communication with the HCS410 there will be a short additional delay, necessary to charge up the tank capacitor through the pull-up resistor and internal voltage regulator, before the device powers up.

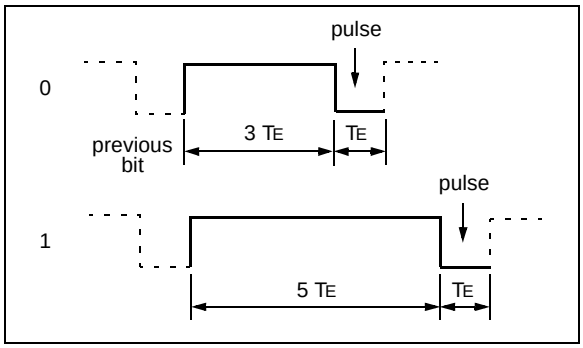
FIGURE 5: PIC® MCU IFF-W CONNECTION



HCS410 IFF OPERATION

Communication with the HCS410 over the Data I/O line uses a Pulse Position Modulation format. The protocol is asymmetrical, meaning that the encoding of ones and zeros differs depending on the direction of communication. This also provides for power supply over the same data line, while optimizing communication speed. When communicating to the HCS410, data and command bits are expressed with the modulation format represented in Figure 6 below.

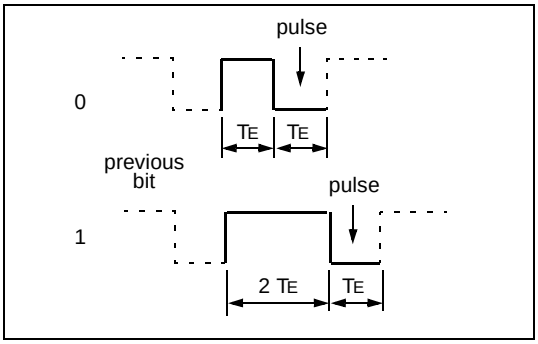
FIGURE 6: COMMANDS TO THE HCS410



Where TE represents the basic Time Element which can be configured to be 100 μs or 200 μs, by setting appropriately the TBSL bit in the device configuration memory.

When the HCS410 sends data bits back, they are expressed with the modulation format represented in Figure 7 below.

FIGURE 7: RESPONSES FROM THE HCS410



Communication activity with the HCS410 always starts with the wait for a specific 5-bit pattern of (01010) known as the Acknowledge Pulse, see Figure 8.

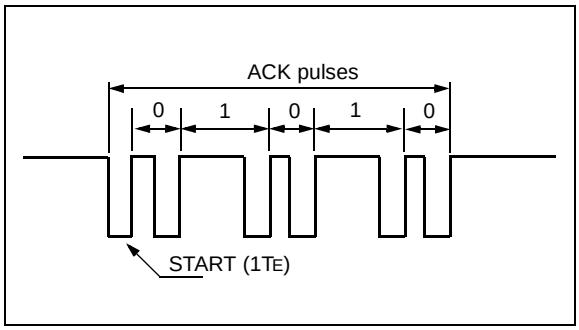
The device will send a new Acknowledge Pulse sequence immediately after a successful Power-up Reset, and will repeat the pattern periodically while waiting for a command.

The internal clock oscillator of the HCS410 is based on an RC circuit and can be calibrated by programming the OSCCAL bits in the device configuration memory.

A device programmer can make use of the Acknowledge Pulse sequence to verify such calibration or to finely tune (auto-baudrate) the communication routines.

In the following, for simplicity, we will consider that the HCS410 clock oscillator has been calibrated.

FIGURE 8: ACKNOWLEDGE PULSES



All activity of the HCS410 is performed exclusively in response to specific 5-bit commands. The available commands offer a very powerful set of functions for Authentication applications:

- IFF-Encryption using one of two possible 64-bit keys stored in the device configuration memory
- Read and Write access to four independent 16-bit locations of nonvolatile memory (EEPROM)
- Read access to a 32-bit Serial Number, split in two 16-bit words for convenience
- Write access to the device Configuration word (protected by a special 32-bit transport code)
- The ability to reconfigure the entire device memory, including all keys and calibration parameters

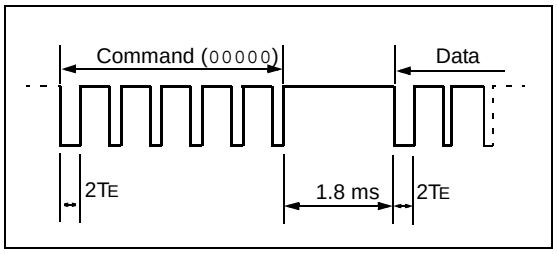
TABLE 1: IFF COMMANDS

Command	Description
00001	Read Configuration word
00010	Read low serial number
00011	Read high serial number
001XX	Read user area (00, 01, 10, 11)
01000	Configure HCS410
01001	Write Configuration word
01010	Write low serial number
01011	Write high serial number
011XX	Write user area (00, 01, 10, 11)
10001	IFF Encryption using KEY-1
10101	IFF Encryption using KEY-2

Commands must be sent to the HCS410 immediately after receiving the Acknowledge Pulse sequence. After completion of one command, the user must wait for the next Acknowledge Pulse sequence from the HCS410 before requesting a new operation.

When sending a command to the HCS410 and subsequently, when sending packets of 16 or 32 data bits, a long START bit (2TE) is required as illustrated in Figure 9.

FIGURE 9: COMMAND-DATA START BIT



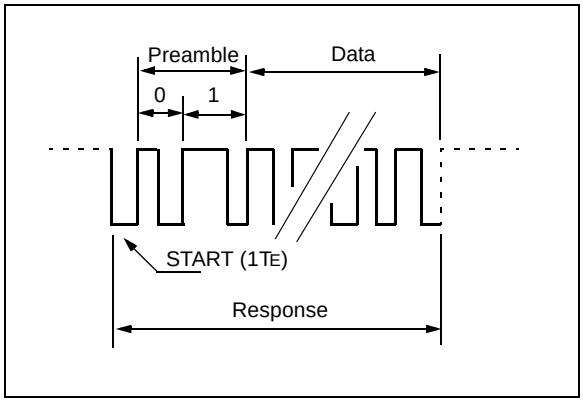
Commands that read and write to memory locations will require 16-bit packets of data to be sent after the command (and a short delay of 1.8 ms) and will respond with packets of the same size.

Encryption commands for authentication, require 32 bits of data to be sent after the command, and respond with the same data length.

Finally, configuration commands are protected by a special 32-bit code, known as the Transport Code. It acts as the secret combination of a safe, and prevents the device configuration from being lost, should a communication error transform a normal command into a configuration command.

Responses from the HCS410 on the contrary, are always preceded by a (single TE) START pulse, followed by a fixed preamble composed of a (0, 1) pattern as shown in Figure 10.

FIGURE 10: RESPONSE PREAMBLE



A more complete summary of the IFF commands and waveforms can be found in Figure 11 and Table 2.

FIGURE 11: IFF COMMANDS AND WAVEFORMS

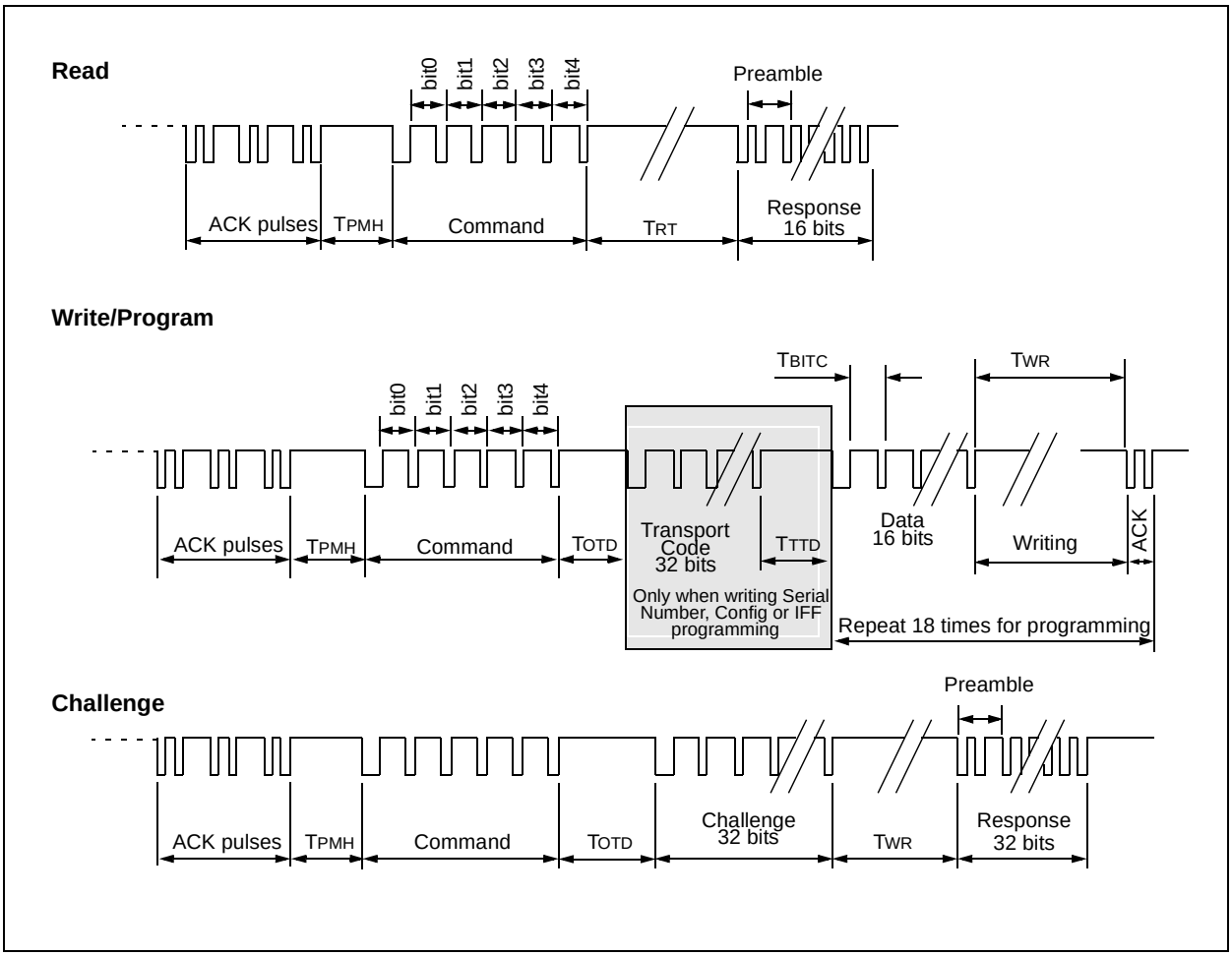


TABLE 2: IFF TIMING PARAMETERS

Parameter	Symbol	Minimum	Typical	Maximum	Units
PPM Command Bit Time Data = 1 Data = 0	TBITC	 3.5 5.5	 4 6	 — —	TE
PPM Response Bit Time Data = 1 Data = 0	TBITR	 — —	 2 3	 — —	TE
PPM Command Minimum High Time	TPMH	1.5	—	—	TE
Response Time (Minimum for Read)	TRT	6.5	—	—	ms
Opcode to Data Input Time	TOTD	1.8	—	—	ms
Transport Code to Data Input Time	TTTD	6.8	—	—	ms
IFF EEPROM Write Time (16 bits)	TWR	—	—	30	ms

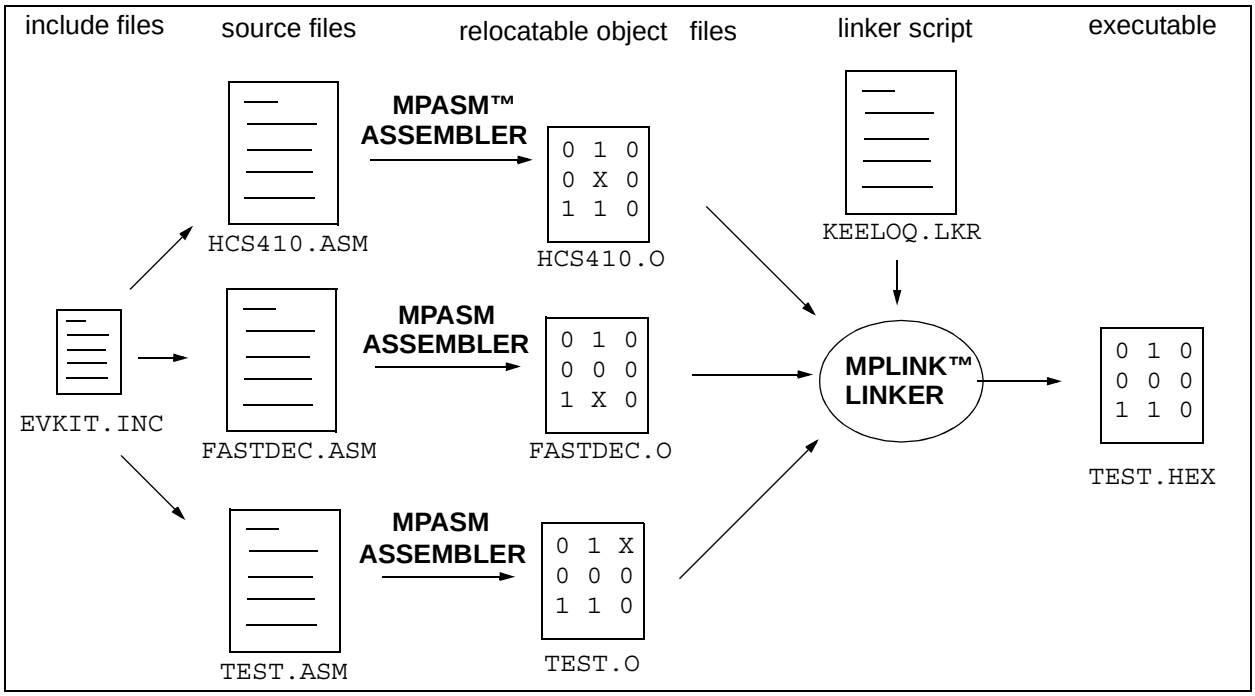
APPLICATION SOFTWARE

The following sections of this Application Note will concentrate on the software. We will develop a small set of functions for communication with an HCS410 in authentication applications. The code was written as a set of relocatable object modules. While the use of relocatable modules and a linker is natural for high level language programmers, many programmers familiar with the Microchip PIC MCU assembler MPASM™, might not be familiar with the use of MPLINK™ object linker and the techniques required to produce relocatable object modules in assembler. Figure 12 illustrates the adopted project structure.

For convenience, as a hardware target we used the KEELOQ Evaluation Kit II (DM303006) as the Main module, and we will use the prototype area to connect an HCS410 in the configuration represented in Figure 4 (B).

The demonstration code can be downloaded to the PIC16F877 present on the demo board through the serial port using the Evaluation Kit windows software (Download Firmware function). Alternatively the user will be able to connect the In-Circuit Debugger MPLAB® ICD to the J4 connector on the same demo board. This second arrangement will also permit the user to insert breakpoints and view the contents of RAM and EEPROM while exploring the code and the device functionality.

FIGURE 12: PROJECT STRUCTURE



As shown in Figure 12, there are several source files that contribute to the creation of the demonstration application. In order of importance, these files are:

- the `EVKIT.H` include file, containing the definitions of all the I/Os of the PIC16F877 that controls the KEELOQ EVALUATION KIT II main board
- the `HCS410.ASM` module, containing all the communication specific functions that will be analyzed in detail in the following sections
- the `FASTDEC.ASM` module, containing the standard KEELOQ decrypt module
- the `TEST.ASM` module, containing the Main Loop of the demonstration application
- the `KEELOQ.LKR` linker script, that defines the memory allocation rules to be used by the linker when assembling the object modules together.

WRITING RELOCATABLE CODE

The reader should refer to the "*MPASM™ Assembler, MPLINK™ Object Linker, MPLIB™ Object Librarian User's Guide*" (DS33014) for more details and programming examples with special reference to Chapter 4: "Using MPASM™ to Create Relocatable Objects."

THE HCS410 MODULE

The HCS410 module exports four global functions for easy access to the main functionality offered by the HCS410 transcoder. They are:

- 32-bit data Encryption with one of two 64-bit keys
- Read access to non volatile memory (4 x 16 bits)
- Write access to non volatile memory (4 x 16 bits)
- 32-bit Serial Number read

The module uses four RAM locations (in an overlay segment) as local variables and temporaries, and it references to one external 32-bit buffer (HOP0..HOP3) for data I/O.

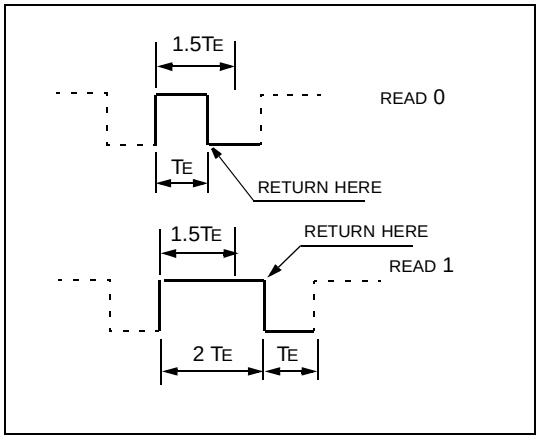
In order to keep the code simple and let the reader concentrate on the core concepts of the communication mechanism, all of the HCS410 communication routines have been written for the most generic PIC MCU using only the Watchdog Timer (as a time-out mechanism) in all waiting loops. This means that if, at any time, the communication fails, or the HCS410 does not respond (or is missing from the replaceable module), the Main module will RESET, preventing the application from executing further.

The following sections will describe the design and use of the routines composing the HCS410.ASM module from the bottom up.

The Delay routine provides multiple of 4 μs delays and adapts to three different clock frequencies; 4 MHz, 8 MHz and 20 MHz depending on the definition of the CLOCK variable (by default CLOCK is defined to be equal to 20 MHz that is the oscillator frequency used by the KEELOQ Evaluation Kit II demo board.

The WaitFall function is one of the main building blocks of the communication routine. Its role is fundamental in the receiving of the special PPM modulation format used by the HCS410 device. WaitFall loops, polling the Data I/O line until a falling edge is detected (pulse) and returns with a '0' or a '1', depending on whether the pulse was detected before or after a specific lapsed time (1.5 TE) .

FIGURE 13: WAITFALL OPERATION



The ReceiveData function uses the WaitFall function to discriminate actual data bits sent by the HCS410. ReceiveData also synchronizes on the START pulse and discards the Preamble (0,1) pattern. The Preamble pattern is extremely valuable in wireless applications to distinguish the transcoder response from the reader circuit noise, but in a wired application, such as the one we are developing, there is no use for it.

Receive Data assembles CNTBIT bits in bytes and proceeds filling the HOP buffer.

The inner loop, labeled ReceiveBit, is also used by the GetACK function to capture the Acknowledge pulses and compare them against the specific (01010) ACK pattern.

The function SendData, performs the opposite task, sending CNTBIT bits from the HOP buffer over the Data I/O line, according to the defined PPM format.

The SendCommand function uses the inner loop of the SendData function to send a 5-bit command.

Finally, let's review the main four functions that the HCS410.ASM declares as global for other modules to use.

Encrypt1 and Encrypt2 respectively request the HCS410 to perform KEELOQ encryption on a 32-bit challenge value taken from the HOP buffer, using the first or second 64-bit key available. The function will return with the 32-bit response in the same HOP buffer.

ReadUserData requests the HCS410 to perform a 16-bit data read command from one of the four User memory locations according to the value of the W register upon call (0,1,2,3). In addition to that, if W contains the value -1 (0xFF) the 16-bit data returned will be the 16 LSb's of the device Serial Number. If W contains the value -2 (0xFE), the 16-bit data returned will be the 16 MSb's of the device Serial Number.

WriteUserData similarly requests the HCS410 to write a 16-bit data to one of the four User memory locations according to the value contained in the W register upon call (0,1,2,3). A write to the Serial Number locations, similarly to the operation of the ReadUserData command, is not supported in the current version (such command would require sending additional 32 bits of Transport Code before the data).

THE FASTDEC MODULE

The FASTDEC.ASM module contains a fast implementation of the KEELOQ Decryption algorithm. It declares a single GLOBAL label Decrypt. This function requires the FSR register to point to an 64-bit array (8 bytes) containing the Crypt Key to be used and operates on an EXTERN 32-bit buffer (HOP0..HOP3) which is holding the data. The decryption function performs 32-bit data decryption in about 16,000 Tcycles (corresponding to about 3 ms @ 20 MHz) and returns the 32-bit output in the same buffer (HOP0..HOP3).

As in every good encryption system, security must not rely on the secrecy of the algorithm used, but solely on the secrecy of the crypt keys used. In a KEELOQ system, there is no exception to this rule: proper key management is vital. In fact, while the KEELOQ algorithm is not secret (although it is patented and released only under license), the secrecy of the crypt keys chosen is of utmost importance.

In many KEELOQ Hopping code applications, it is recommended to use special Key Generation techniques, (simple, normal, secure, etc.), to make the key used by every single unit different and unique. In KEELOQ IFF technology this is not necessary, because of its very nature, the challenge/response system already provides a very high level of security (provided a good random generation algorithm is used for calculating the challenge).

It must be remembered, however, that if a simple single crypt key (Manufacturer's Code) system is chosen, then the disclosure of this 64-bit value to third parties and subcontractors becomes extremely critical.

THE TEST MODULE

The `TEST.ASM` module contains some basic code to initialize the PIC MCU for the Evaluation Kit II board hardware configuration and to demonstrate the functionality of the HCS410 module.

In detail, the main loop will cyclically perform the following operations:

1. Request the Serial Number.
2. Write to all three 16-bit locations in User nonvolatile memory.
3. Read back the written values.
4. Generate a pseudo random challenge (using the KEELOQ decrypt function with a different key provides a good random number generator.
5. Send the challenge and request an Encryption with one of the two keys contained in the HCS410 device.
6. Send the response to the `Fastdec` function and, finally, compare it to the original challenge.

If there is a match, the loop continues. Otherwise, an error is reported and the board RESETS.

MODULE AUTHENTICATION APPLICATIONS

It must be noted that there are two main categories of applications where the Module Authentication system, as illustrated in this Application Note, can be conveniently employed. The difference between the two can be better explained with example applications. The first category, "closed systems", can be represented by a cellular phone and its battery pack. Let's say the HCS410 (authentication device) is placed in the battery pack and one contact is added to the battery to allow for the IFF bi-directional communication. This would force the replacement of the battery with the manufacturer's original batteries only. If a user wants to bypass the authentication mechanism, the phone (a closed system) must be disassembled. This not only voids any warranty, but is also perceived by the average user as a risky operation and is unlikely to be performed. On the contrary, in an "open system" such as an industrial rack-mounted stack module (or the one presented at the beginning of this Application Note), there is no restriction to work-around the authentication module, since there is easy access to the interconnection system. The key to effective use of module authentication in "open systems" is to make the authenticating device an "essential" part of the system. In the replaceable module example, this would mean storing some essential configuration information such as gain or offset values of the analog I/Os present. in the User nonvolatile memory area of the HCS410. Without those values, the system cannot perform any useful operations.

MEMORY USAGE

Program memory used: 324 words.

RAM memory used: 20 bytes.

CONCLUSIONS

The code presented shows how effective the use of an HCS410 transcoder can be in a module authentication application. In summary:

- KEELOQ IFF technology offers 64-bit security in a low cost device.
- The `HCS410.ASM` relocatable object module can be linked in any existing application with minimal load on the PIC MCU resources.
- A single pin added to the interconnection system (single wire) provides an effective bi-directional communication path.
- Four 16-bit locations of R/W memory provide space for calibration data and/or usage statistics.
- A 32-bit serial number (read-only) allows module tagging and effective production and maintenance tracking.

There is still ample room for extension of this application code. The HCS410 includes a full Anti-Collision system that would allow the connection of multiple (HCS410) devices on a single wire, in a bus-like configuration. Authentication of multiple modules using a single PIC MCU I/O would be possible on such a bus.

Further time-outs could be explicitly controlled by the use of timers (instead of the sole Watchdog Timer), to provide module "hot swapping" capabilities, such as multiple modules insertion/extraction detection.

REFERENCES

DS91002 TB003 Introduction to KEELOQ Technology

DS91000 TB001 Secure Learning RKE systems using KEELOQ Technology

DS40158 HCS410 Device Data Sheet

DS00742 AN742 Modular Mid Range Decoder

DS00744 AN744 Modular Mid Range Decoder in C

DS91041 KEELOQ Decryption Routines in C

DS00824 KEELOQ Encoders Oscillator Calibration

DS00665 Using KEELOQ to Generate Hopping Code Passwords

ADDITIONAL INFORMATION

Microchip's Secure Data Products are covered by some or all of the following: Code hopping encoder patents issued in European countries and U.S.A. Secure learning patents issued in European countries, U.S.A. and R.S.A.

APPENDIX A: EVKIT.INC SOURCE CODE

```
;*
;** EvKit II include file
;**
;*  09/27/01 LDJ updated
;*

#define TRUE  1
#define FALSE 0

;// serial port
#define RTS     PORTC,0 ;// o
#define CTS     PORTC,5 ;// i
#define TX_232  PORTC,6 ;// o
#define RX_232  PORTC,7 ;// i

;// LCD display
#define DB4     PORTD,0 ;//i/o
#define DB5     PORTD,1 ;//i/o
#define DB6     PORTD,2 ;//i/o
#define DB7     PORTD,3 ;//i/o
#define RW      PORTD,4 ;// o select RD or Write
#define RS      PORTD,5 ;// o select register
#define LCD_E   PORTD,6 ;// o chip select

;// programming Socket
#define LED     PORTD,7 ;// o switch on the yellow LED
#define VDD     PORTD,7 ;// o set socket VDD pin to 5V

;// encoder programming
#define S0      PORTB,0 ;// o clock
#define S1      PORTB,1 ;// o
#define S2      PORTB,2 ;// o
#define S3      PORTB,3 ;// o
#define DATA    PORTA,5 ;// i/o data

;// Step up circuit
#define VPPON   PORTA,4 ;// o apply voltage
#define STEPOUT PORTC,2 ;// o PWM output pin
#define VPPRST  PORTB,5 ;// o clears Vpp/ reset part
#define STEPIN  PORTA,0 ;// i analog input

;// HCS512 programming
#define DAT512  PORTA,2 ;// i/o
#define CLK512  PORTA,1 ;// i/o
#define MCL512  PORTB,4 ;// i MCLR line
```

```
;// HCS515/HCS500 programming
#define DAT515  PORTE,2 ;// i/o
#define CLK515  PORTA,3 ;// i/o

;// transponder interface
#define PWM_COIL PORTC,1 ;// o 125kHz carrier
#define COIL_IN  PORTE,1 ;// i reading transponder
#define COIL_ON  PORTB,0 ;// o activate resonant circuit

;// radio receiver
#define RFIN    PORTE,0  ;// i radio input

;// i2c interface
#define SCL     PORTC,3  ;// o Serial Clock
#define SDA     PORTC,4  ;// i/o Serial Data
;#define TRIS_SDA TRISC4

;// keypad interface
#define BOOT    PORTB,4 ;// i BOOT button
#define ROW0    PORTB,0  ;// o rows
#define ROW1    PORTB,1  ;// o
#define ROW2    PORTB,2  ;// o
#define ROW3    PORTB,3  ;// o
#define COL0    PORTB,4  ;// i columns
#define COL1    PORTB,5  ;// i
#define COL2    PORTB,6  ;// i
#define COL3    PORTB,7  ;// i


;// init masks
;// clock/data lines to decoders are default to float at init
#define DEFPA   b'11101111'; // x x DATA VPPON CLK515 DAT512 CLK512 STEPIN
#define DEFPB   b'11010000'; // COL3 COL2 VPPRST COL0 S3 S2 S1 S0/COIL_ON
#define DEFPC   b'10110110'; // RX_232 TX_232 CTS SDA SCL STEPOUT PWM_COIL RTS
;// STEPOUT defaults to input until used (external pull down)
#define DEFPD   b'00000000'; // LED E RS RW DB7..DB4
#define DEFPE   b'11101111'; // x x x !PSP x DAT515 COIL_IN RFIN

#include <p16f877.inc>

; debugging macro, outputs 1 literal digit on LCD connector
DEBUG   MACRO   LIT
        movlw   LIT & 0x0f
        iorwf   PORTD,F
        nop
        nop
        movlw   0xf0
        andwf   PORTD,F
        ENDM
```

APPENDIX B: TEST.ASM

```
;
; test program for "relocatable" ENCRYPT routine
;   using an HCS410 in IFF-W mode
;
    radix hex

    include <evkit.inc>

    include "hcs410.inc"   ; encryption lib module exports
    include "fastdec.inc"   ; decryption lib module exports

;-----------------------------------------------------------------------
;
BANK1        UDATA           ; make sure gets allocated in BANK0
KEY0    RES     1       ; crypto key stored lsb first
KEY1    RES     1
KEY2    RES     1
KEY3    RES     1
KEY4    RES     1
KEY5    RES     1
KEY6    RES     1
KEY7    RES     1       ; crypto key msb

BANK0       UDATA
HOP0    RES     1       ; hopping code 32 bit register
HOP1    RES     1
HOP2    RES     1
HOP3    RES     1

; make this variables accessible to enc/decrypt modules
    GLOBAL KEY0, KEY1, KEY2, KEY3, KEY4, KEY5, KEY6, KEY7
    GLOBAL HOP0, HOP1, HOP2, HOP3

SN      RES     4       ; temp storage for Serial Number
CHG     RES     4       ; challenge



STARTUP    CODE
    goto    START

    CODE               ; reset vector

START
    banksel TRISA
    movlw   DEFPA
    movwf   TRISA
    movlw   DEFPB
    movwf   TRISB
    movlw   DEFPC
    movwf   TRISC
    movlw   DEFPD
    movwf   TRISD
    movlw   DEFPE
    movwf   TRISE

    banksel ADCON1
    movlw   0x0e        ; AN0 only analog input
    movwf   ADCON1

    banksel OPTION_REG
    movlw   b'10001011' ; prescaler to WDT 1:8
    movwf   OPTION_REG
```

```
; init CRYPTO KEY
    banksel KEY0
    movlw0xEF        ; lsb
    movwfKEY0
    movlw0xCD
    movwfKEY1
    movlw0xAB
    movwfKEY2
    movlw0x89
    movwfKEY3
    movlw0x67
    movwfKEY4
    movlw0x45
    movwfKEY5
    movlw0x23
    movwfKEY6
    movlw0x01
    movwfKEY7       ; msb

; load init value for challenge
    banksel CHG
    movlw   0x67
    movwf   CHG+0
    movlw   0x45
    movwf   CHG+1
    movlw   0x23
    movwf   CHG+2
    movlw   0x01
    movwf   CHG+3


MainLoop
; 1. read Serial Number locations

    movlw   -1          ; SN 16 Lsb
    call    ReadUser
    movf    HOP0,W      ; save value in local variable
    movwf   SN+0
    movf    HOP1,W
    movwf   SN+1

    movlw   -2          ; SN 16 Msb
    call    ReadUser
    movf    HOP0,W      ; save value in local variable
    movwf   SN+2
    movf    HOP1,W
    movwf   SN+3

; 2. write to all EEPROM USER locations (4321)
    movlw   21
    movwf   HOP0
    movlw   43
    movwf   HOP1

; write to EEPROM location 0
    movlw   0
    call    WriteUser
; write to EEPROM location 1
    movlw   1
    call    WriteUser
; write to an EEPROM location 2
    movlw   2
    call    WriteUser
```

```
; write to an EEPROM location  3
    movlw   3
    call    WriteUser

; 3. read back values from USER EEPROM locations
; read EEPROM location 0
     movlw   0
     call    ReadUser
; read EEPROM location 1
     movlw   1
     call    ReadUser
; read EEPROM location 2
     movlw   2
     call    ReadUser
; read EEPROM location 3
     movlw   3
     call    ReadUser

; 4. generate a new challenge (and turn off LED)
; challenges should be randomized,
; Hint: use again Keeloq with a different Crypt Key
;       to act as a random number generator
; for simplicity a sequential challenge generation is used here
    bcf     LED
    incf    CHG+0,F
    btfsc   STATUS,Z
    incf    CHG+1,F

; move challenge into data buffer
    movf    CHG+0,W
    movwf   HOP0
    movf    CHG+1,W
    movwf   HOP1
    movf    CHG+2,W
    movwf   HOP2
    movf    CHG+3,W
    movwf   HOP3

; 5. call HCS410 encrypt library function
    call    Encrypt1

; 6. verify the response using Keeloq Decrypt
    movlw   KEY0       ;   enter with W pointing to CRYPTO KEY
    call    Decrypt

    movf    HOP0,W      ; compare decrypted response with challenge
    xorwf   CHG+0,W
    btfss   STATUS,Z
    goto    AError

    movf    HOP1,W      ; compare decrypted response with challenge
    xorwf   CHG+1,W
    btfss   STATUS,Z
    goto    AError

    movf    HOP2,W      ; compare decrypted response with challenge
    xorwf   CHG+2,W
    btfss   STATUS,Z
    goto    AError

    movf    HOP3,W      ; compare decrypted response with challenge
    xorwf   CHG+3,W
    btfss   STATUS,Z
    goto    AError
```

```
; if successful turn LED on
    bsf     LED

; 7. loop
    goto    MainLoop

AError
    goto    0           ; reset in case of error

    END
```

APPENDIX C: HCS410.INC

```
;;*****************************************
**************************
;*  Filename:   HCS410.INC
;*****************************************
**************************
;*  Author:     Lucio Di Jasio
;*  Company:    Microchip Technology
;*  Revision:   Rev 1.00
;*  Date:       28/SEP/01
;*
;*  include file that exports:
;*    relocatable library function for HCS410
use
;*
;*****************************************
**************************

    EXTERN Encrypt1, Encrypt2, ReadUser,
WriteUser
```

APPENDIX D: HCS410.ASM

```
;
;                       Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated
; (the "Company") for its PICmicro® Microcontroller is intended and
; supplied to you, the Company’s customer, for use solely and
; exclusively on Microchip PICmicro Microcontroller products. 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.
;
;;*********************************************************************
;*  Filename:   HCS410.ASM
;*********************************************************************
;*  Author:     Lucio Di Jasio
;*  Company:    Microchip Technology
;*  Revision:   Rev 1.00
;*  Date:       28/SEP/01
;*
;*  relocatable module that implements:
;*    Encrypt function challenging an HCS410 in IFFW
;*    encrypts the 32 bits hopping in IN HOP0 (LSB) TO HOP3(MSB)
;*    W contains pointer to HOP0
;* NOTE:
;*    local variables are assumed to be in BANK0
;*    HOP must be in BANK0 or BANK1
;*    assembled using MPASM v02.61
;*
;*********************************************************************
    include "evkit.inc"     ; Keeloq Evaluation Kit II board I/O defs

#define TWENTYMHZ   5
#define EIGHTMHZ    2
#define FOURMHZ     1

#define CLOCK TWENTYMHZ

    EXTERN  HOP0, HOP1, HOP2, HOP3
;--------------------------------------------------------------------
; HCS410 commands
;
#define HCS410_RCFG  0x01       ;// read config word
#define HCS410_RSNH  0x03       ;// read SN high
#define HCS410_RSNL  0x02       ;// read SN low
#define HCS410_RU0   0x04       ;// read USER area #0
#define HCS410_RU1   0x05       ;//   "       "    #1
#define HCS410_RU2   0x06       ;//   "       "    #2
#define HCS410_RU3   0x07       ;//   "       "    #3
#define HCS410_WU0   0x0c       ;// write USER area #0
#define HCS410_WU1   0x0d       ;//   "       "     #1
#define HCS410_WU2   0x0e       ;//   "       "     #2
#define HCS410_WU3   0x0f       ;//   "       "     #3
#define HCS410_ENC1  0x11       ;// challenge resp. 32Bit Key1 Encrypt
#define HCS410_ENC2  0x15       ;// challenge resp. 32Bit Key2 Encrypt
```

```
;-----------------------------------------------------------------------
; Macros to control the DATA line
; implementation specific to the Keeloq Evaluation Kit II board
;

; default condition (DATA line pull-up)
PULLUP  MACRO
        banksel     TRISA
        bsf         DATA        ; make it input
        banksel     PORTA
       ENDM

; PPM pulse condition (DATA line to GND)
PULSE   MACRO
        banksel     TRISA
        bcf         DATA        ; make it output
        banksel     PORTA
        bcf         DATA        ; drive LOW
       ENDM

        UDATA_OVR               ; defines local variables (in reusable mem)
WORK    RES     1
TMP     RES     1
CNTBIT  RES     1
DLY     RES     1

.hcs410 CODE

;-----------------------------------------------------------------------
Encrypt1
        GLOBAL Encrypt1

        call    GetACK          ; sycnhronize

; send authentication request
        movlw   HCS410_ENC1     ; encrypt request
        goto    Encrypt

;-----------------------------------------------------------------------
Encrypt2
        GLOBAL Encrypt2

        call    GetACK          ; sycnhronize

; send authentication request
        movlw   HCS410_ENC2     ; encrypt request
Encrypt
        call    SendCommand

; send challenge (data to be encrypted)
        movlw   .32
        call    SendData

; wait for response
        movlw   .32
        call    ReceiveData

        return

;-----------------------------------------------------------------------
; Read User Data
; INPUT
;   W           0..3 select User Location
;               -1 = Serial Number L
```

```
;               -2 = Serial Number H
; OUTPUT:
;   HOP0, HOP1  data read form user memory
;
ReadUser
        GLOBAL ReadUser

        addlw   HCS410_RU0      ; make it a 410 command
        movwf   HOP3            ; save in temp

        call    GetACK          ; synchronize

        movf    HOP3,W          ; send comand
        call    SendCommand
        movlw   .16
        call    ReceiveData     ; read back 1 x16 bit word
        return

;-------------------------------------------------------------------------
; Write User Data
; INPUT
;   W           0..3 select User Location
;   HOP0, HOP1  data to write in user memory
;
WriteUser
        GLOBAL WriteUser

        addlw   HCS410_WU0      ; make it a 410 command
        movwf   HOP3            ; save in temp

        call    GetACK          ; synchronize

        movf    HOP3,W          ; send comand
        call    SendCommand
        movlw   .16
        call    SendData        ; read back 1 x16 bit word
        return

;-------------------------------------------------------------------------
; Delay W x 4us
; INPUT
;   W   delay requested
;
DelayWx4us
            movwf   DLY         ; DLY x 4us
DelayL
            nop                 ; 1 Tcy
 if CLOCK > FOURMHZ
            goto    $+1         ; 4 Tcy
            goto    $+1
 endif
 if CLOCK > EIGHTMHZ
            goto    $+1         ; 12 Tcy
            goto    $+1
            goto    $+1         ;
            goto    $+1
            goto    $+1         ;
            goto    $+1
 endif
            decfsz  DLY,F         ; +3 Tcy
            goto    DelayL
            retlw   0

;-------------------------------------------------------------------------
; Wait for a Falling Edge
```

```
;
; INPUT
;   W timeout value
; timeout = DLY x .16Tcy
;
#define KTO     .16
WaitFall
            movwf   DLY
WaitFallL
            nop                 ; 1 Tcy
            goto    $+1
            goto    $+1
            goto    $+1
            goto    $+1
            goto    $+1         ;
            btfss   DATA
            retlw   0           ; return with edge

            decfsz  DLY,F
            goto    WaitFallL

; if long wait some more until it falls
WaitFallTOL
            btfsc   DATA
            goto    WaitFallTOL

            retlw   1           ; return with TO


;-----------------------------------------------------------------------
; GetACK
;   loops until it receives a correct ACK pattern
;
GetACK
        PULLUP                  ; power up the device
        movlw   .250*CLOCK/KTO
        call    DelayWx4us      ; pause 1ms

; wait for ACK signal
        movlw   TMP
        movwf   FSR             ; TMP will receive the data
        movlw   3               ; read 3 bit (preceded by 01)
        movwf   CNTBIT
        call    ReceiveBit
        movf    TMP,W
        andlw   b'11100000'     ; compare (upper) three bit
        xorlw   b'01000000'     ; match with 010 ACK pattern
        btfss   STATUS,Z
        goto    GetACK          ; if not keep waiting
        return

;-----------------------------------------------------------------------
;  ReceiveData
;
; INPUT :
;   W   number of bit to receive
; OUTPUT:
;   HOP0..HOP3 received data
;
ReceiveData
            movwf   CNTBIT
            movlw   HOP0            ; init pointer
            movwf   FSR

ReceiveBit
```

```
; wait for start falling edge (WDT timeout)
            btfsc   DATA
            goto    ReceiveBit

; discard first two bit 01 (synch pattern)
WR1
            btfss   DATA
            goto    WR1
WF1
            btfsc   DATA
            goto    WF1
WR2
            btfss   DATA
            goto    WR2
WF2
            btfsc   DATA
            goto    WF2

ReceiveNBL
            btfss   DATA            ; waiting for the new rising edge
            goto    ReceiveNBL

; measure the delay before next gap
            movlw   .300*CLOCK/KTO  ; timeout at 300us (1.5xTE)
            call    WaitFall
            movwf   WORK
            rrf     WORK,F          ; move bit to carry
            rrf     INDF,F          ; rotate bit in lsb first
WR3
            btfss   DATA            ; wait until you get to the next gap
            goto    WR3


; loop for every bit
            decf    CNTBIT,F
            movlw   0x7             ; every 8 bit increment pointer
            andwf   CNTBIT,W
            btfsc   STATUS,Z
            incf    FSR,F
            movf    CNTBIT,F
            btfss   STATUS,Z
            goto    ReceiveNBL
            return


;-----------------------------------------------------------------------
;  SendData
;
; INPUT
;   W   number of bit to send
;   HOP0..HOP3 data to send
;
SendData
            movwf   CNTBIT
            movlw   HOP0            ; init pointer
            movwf   FSR

SendCMD
            movlw   .250              ; max delay 256x4us = 1ms
            call    DelayWx4us
            movlw   .250              ; max delay 256x4us = 1ms
            call    DelayWx4us
            movlw   .250              ; max delay 256x4us = 1ms
            call    DelayWx4us
            movlw   .250              ; max delay 256x4us = 1ms
```

```
            call    DelayWx4us

            PULSE
            movlw   .100
            call    DelayWx4us      ; 2 x TE = 100x4us =  400us
;
; loop for CNTBIT
SendNBL
            PULLUP
            rrf     INDF,F          ; lsb first
            movlw   .150            ; bit 0-> 3 x TE = 150x4us =  600us
            btfsc   STATUS,C
            movlw   .250            ; bit 1-> 5 x TE = 250x4us = 1000us
            call    DelayWx4us

            PULSE
            movlw   .50             ; 1x TE = 50x4us = 200us
            call    DelayWx4us
; count bit out
            decf    CNTBIT,F
            movlw   0x7             ; every 8 bit increment pointer
            andwf   CNTBIT,W
            SKPNZ
            incf    FSR,F
            movf    CNTBIT,F
            btfss   STATUS,Z
            goto    SendNBL

            PULLUP
            return

;--------------------------------------------------------------------
; Send Command
;
; INPUT
;   W  HCS410_ command
;
SendCommand
        movwf   TMP             ; save command in temp
        movlw   TMP             ; make FSR point to it
        movwf   FSR
        movlw   5               ; send 5 bit
        movwf   CNTBIT
        goto    SendCMD        ; enter send data subroutine


    END
```

APPENDIX E: KEELOQ.LKR

```
// File: keeloq.lkr
// Sample linker command file for mid-range with:
//   1 ROM page(2k), 2 RAM banks
// Lucio Di Jasio 09/26/01
//
LIBPATH  .

CODEPAGE   NAME=vectors  START=0x0      END=0x4      PROTECTED
CODEPAGE   NAME=page     START=0x5      END=0x7FF
CODEPAGE   NAME=.idlocs  START=0x2000   END=0x2003   PROTECTED
CODEPAGE   NAME=.config  START=0x2007   END=0x2007   PROTECTED

DATABANK   NAME=sfr0     START=0x0      END=0x1F     PROTECTED
DATABANK   NAME=sfr1     START=0x80     END=0x9F     PROTECTED

DATABANK   NAME=gpr0     START=0x20     END=0x6F
DATABANK   NAME=gpr1     START=0xA0     END=0xBF     PROTECTED

SHAREBANK  NAME=gprnobnk START=0x70     END=0x7F
SHAREBANK  NAME=gprnobnk START=0xF0     END=0xFF

SECTION    NAME=STARTUP  ROM=vectors    // Reset and interrupt vectors
SECTION    NAME=PROG     ROM=page       // ROM code space
SECTION   NAME=IDLOCS ROM=.idlocs    // ID locations
SECTION    NAME=CONFIG   ROM=.config    // Configuration bits location
SECTION    NAME=BANK0    RAM=gpr0       // allocates ram in bank0
SECTION    NAME=BANK1    RAM=gpr1       // allocates ram in bank1
```

APPENDIX F: FASTDEC.INC

```
;;*****************************************************************
;*  Filename:   FASTDEC.INC
;*****************************************************************
;*  Author:     Lucio Di Jasio
;*  Company:    Microchip Technology
;*  Revision:   Rev 1.00
;*  Date:       26/SEP/01
;*
;*  include file that exports:
;*    relocatable library function for Keeloq Decrypt
;*    decrypts the 32 bits in HOP0 (LSB) TO HOP3(MSB)
;*    uses W as pointer to KEY0
;*
;*
;*****************************************************************

    EXTERN Decrypt
```

NOTES:

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.

Trademarks

ISBN: 978-1-61341-273-2

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2009

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Hangzhou
Tel: 86-571-2819-3180
Fax: 86-571-2819-3189

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-6578-300
Fax: 886-3-6578-370

Taiwan - Kaohsiung
Tel: 886-7-213-7830
Fax: 886-7-330-9305

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

05/02/11