Knowledge of baseline

CP, CPD, ID

The purpose of these pages is to get to know some peculiarities of Baseline ICPs that are usually overlooked:

- **CP** - Code *Protection* : Protect code from proofreading
- **CPD** - *Code Protection Data area* : Flash Data Protection
- **ID** -*Dentifiers* : identifier of the chip

In addition, let's also look at something about EEPROM memory.

CP

One of the options that can be activated through the initial configuration is that of protecting what is written in the program memory. This option is present in any PIC, although it may look slightly different depending on the amount of memory built into the chip.

Once this option is enabled, it will no longer be possible to read the contents of the program memory through the development tools. This helps protect the program from unwanted copies. For Baselines, just add in the config line:

```
    _CP_ON
```

This can be done for any of the sources seen so far; The result will be that you will not be able to access the program memory area through the programming device. To disable the option, you need to reprogram the microcontroller, i.e. delete its contents.

Code protection is used in commercial products to prevent copying. During experimentation it is recommended NOT to activate it, since it would only insert a useless element. It will be enabled when, once the program has been verified, you want to protect it from unwanted copies.

It should be clarified that:

- **the protection does not prevent the operation of the program**, but blocks the reading through the ICSP connection between the chip and the programming device, whose management software will indicate the impossibility of accessing the memory.
- The only way to get rid of the protection is to wipe the chip, which, of course, also erases the program itself.

For your information, there are methods to bypass protection, but they are complex and expensive. One of the common attacks involves decapsulating the chip to get on the surface of the

bare platelet, various methods to remove protective layers and interfere with connections, and then, using microprobe, read the contents of the memory. Other methods employ radiography and the like. There are a number of companies that offer this type of service, although, of course, these are non-legal activities in many jurisdictions.
It must be said, however, that the cost and risks of the operation are significant and generally, unless you want simple clones, hacking chips or reverse engineering can be inconvenient compared to writing the program from scratch.

As far as Baselines are concerned, it is important to know that the protection prevents the code from being read again only from location 40h, i.e. the first 64 locations of the program memory are accessible even if the protection is activated.

This means that a sensitive part of the code cannot be written in this area if you want to protect it. The solution is to move the code further:

```
RESVEC  ORG 00
      goto   0x40        ; Skip unprotected area
        ORG 0x40         ; Start Code in Protected Area
      movwf  OSCCAL
```

This "wastes" 63 locations of program memory, an area in itself not very large in the Baselines: if you need to take advantage of this too, you can insert tables or subroutines whose rereading by third parties does not constitute a harm. For example, you can place messages or lookup tables:

```
RESVEC  ORG 00
      goto   0x40        ; Skip unprotected area

; Tables on page 0
; copyright
cpymsg dt  "© www.microcontroller.it"

; segment data table
segtbl
      andlw 0x0F         ; Low nibble only
      addwf PCL,f        ; punta PC
 dt   0x3F, 6, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 7, 0xEF, 0x6F

; Check for Overrun Area first 64 locations
   #if $ > 0x3F
  error "Troppi elementi nell' area 00-3Fh"
   #endif
```

The final three lines are a safety check to avoid exceeding the capacity of the area, generating an error message in the compilation: if the Program Counter is greater than 3Fh it will be necessary to consider moving the address of the initial jump or decreasing the elements inserted.

In this example, we used the `dt` directive which allows the quick creation of `retlw tables`.

DT

The **DT** (*Define Data Table*) directive allows you to create retlw tables in a very simple way. The syntax is:

[label]	sp	Dt	sp	expr, expr,...

The directive generates a line of **retlw** statement for each expression in question. Each expression must yield an 8-bit value. ASCII characters are entered with their value. It is ideal for creating tables in programs for PIC10/12/16. For example,:

```
dt  0x3F, 6, 0x5B, 0x4F,"MSG",0
```

It will be compiled as:

```
retlw 0x3F
 retlw 6
 retlw 0x5B
 retlw 0x4F
 retlw 0x4D
 retlw 0x53
 retlw 0x47
 retlw  0
```

CPD

Chips that have Flash Data (EEPROM), such as the 16F526, also have the option of re-reading protection of this area. Similar to the previous one, the following is enabled in the config line:

```
_CPD_ON
```

Note the aforementioned lack of homogeneity in the configuration items: for 12F519, the option becomes:

```
_CPF_ON
```

Again, it should be clarified that:

- **the protection does not prevent the operation of the program**, nor the operations in EEPROM by it. Its purpose is to block the reading through the ICSP connection between the chip and the programming device, whose management software will indicate the impossibility of accessing this memory area.
- Deletion of the option requires, as before, a new programming of the chip.

It is advisable not to enable this option if it is not necessary.

ID

A little-known element of PICs is the area of identifiers.
These are typically 4 locations, present in all PCIs, in the Flash area; Their content is not accessible from the program, but they can be written and read during programming. In these, the user can enter codes for the identification of the product, for example in relation to the firmware version or a checksum or other.

For this reason, a special MPASM statement is used: `idlocs` . The use is simple: it is a line inserted in the source:

```
__idlocs  [espressione]
```

As a result, the four ID locations are written with the value of **`[expression]`**. For Baselines, only the lowest 4 bits of the ID locations should be used, leaving the others at 0. For example,:

```
 #include p16f526.inc ; Include standard header per il processore scelto

__idlocs  0x1234     ; set ID a 1234
```

`idlocs` is preceded by a double underlining (____) and must be after the processor is defined, in order for the compiler to correctly allocate the desired value in memory: this is a command that acts in a similar way to__________`config`.
Writing the ID is not mandatory and can be safely ignored.

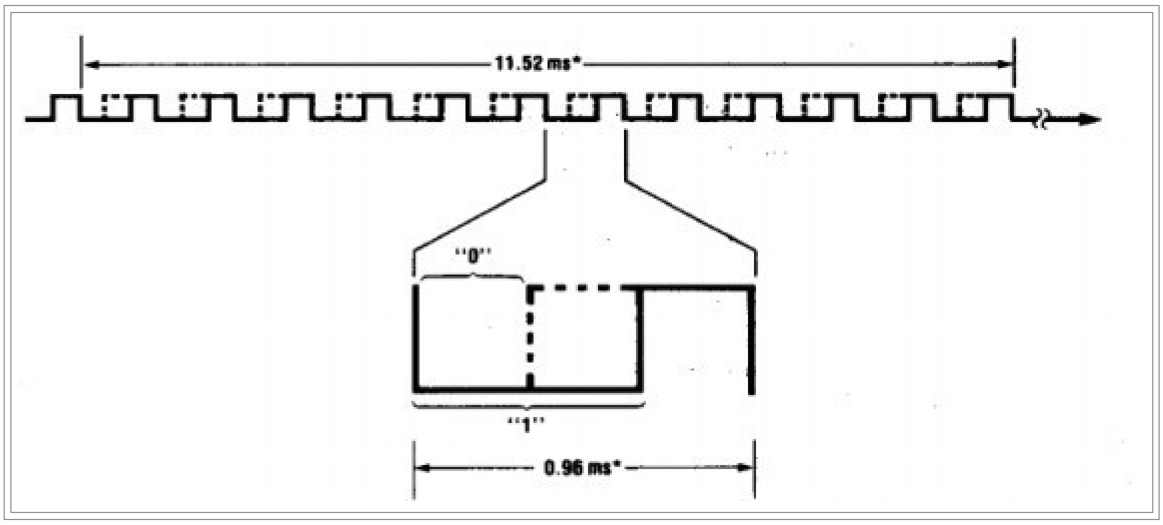
For your information, **`idlocs`** is a default element of the compiler and is referenced in the ***processorname.lkr file*** (for example, ***C:\Program Files\Microchip\MPASM Suite\LKR\16F526.lkr***)

```
CODEPAGE NAME=.idlocs START=0x440 END=0x443
```

An application

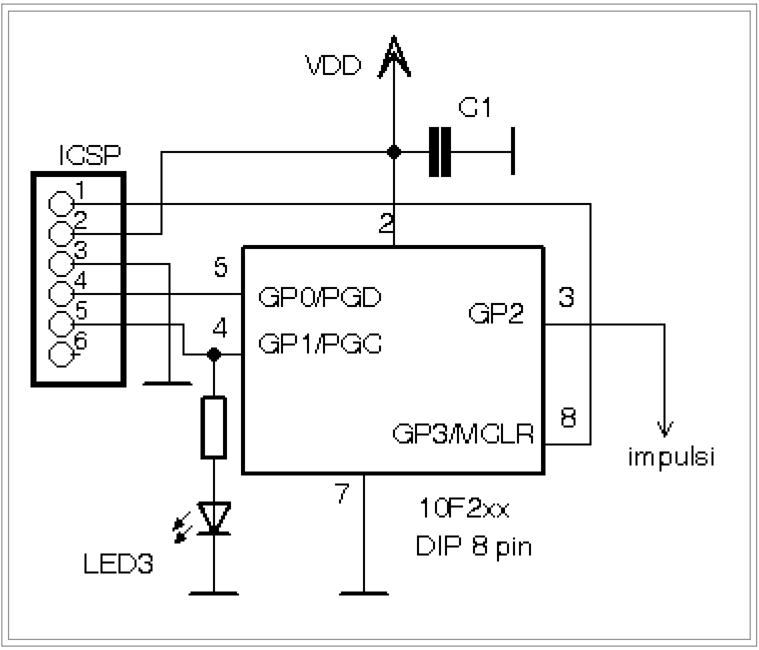
In the past, numerous integrated circuits have been developed for remote controls, such as gate openers, to implement coding systems that prevent operation with keys other than the appropriate ones.
For some time now, these functions have been performed by microcontrollers, even with high-security algorithms (such as Keeloq), but it is also possible to replace the various UM3750, MC146026, MM53200, UM86409, etc.
In particular, let's see how, with a small PIC, it is possible to obtain the same MM53200 signal.

It is an integrated circuit from National Semiconductors dating back to the 80s of the last century (!) and now practically out of production, but which, having been one of the first of its kind, has been used in a considerable quantity, both applications and parts.
The chip receives 12 bits as input, usually encoded with dip switches, and transmits a string of 12 elements, preceded by a start bit:



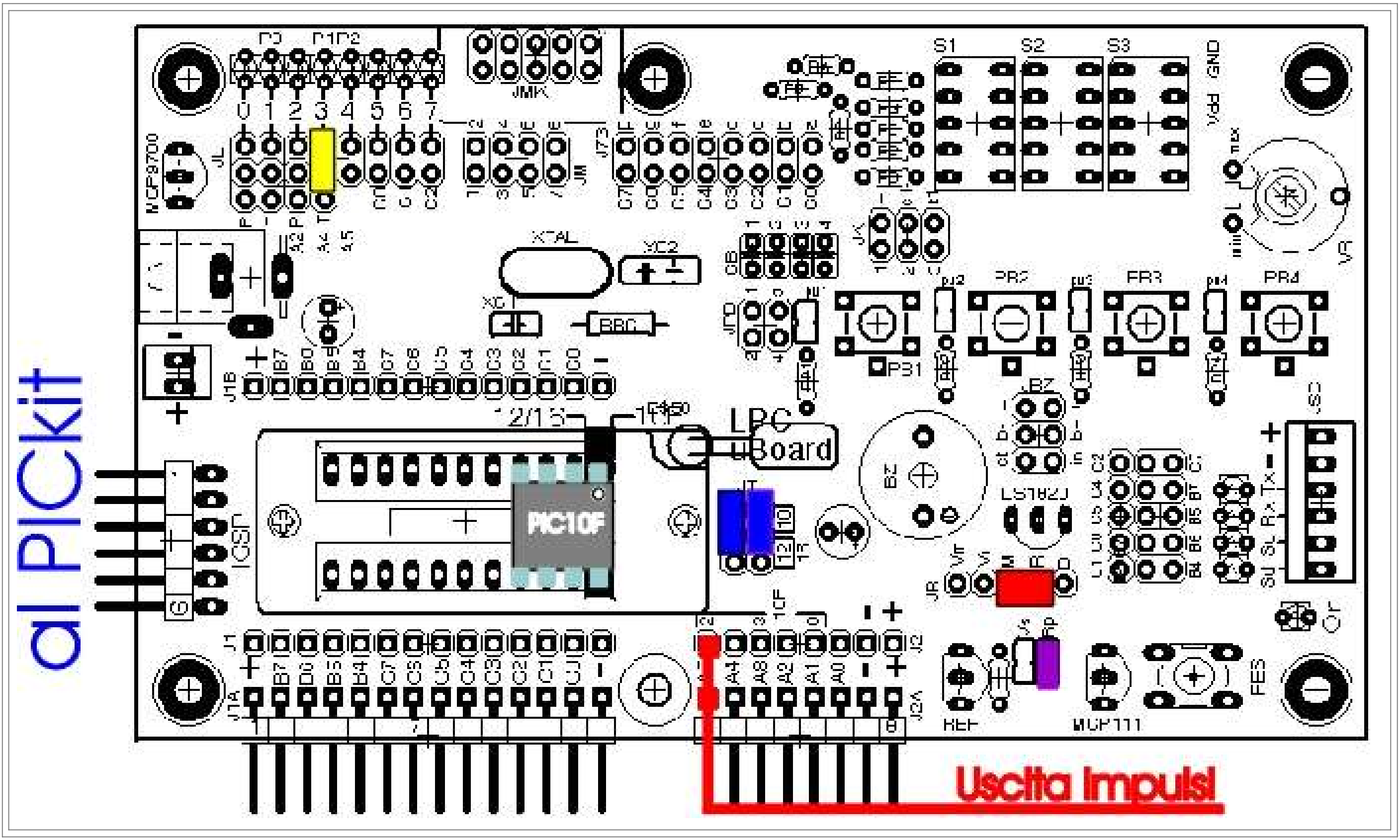
Each bit of data has a nominal duration of 0.96ms. It starts with a 0.32ms start pulse, followed by 12 "bits" of data.
A bit at 0 is output as a low level of 0.32ms, followed by a high level of 0.64ms. A bit at 1 is output as a low level of 0.64ms, followed by a high level of 0.32ms.
The overall duration of the string is 11.25ms, and this is the minimum pause between one string and the next.
The timing is not critical, as the original application includes an RC oscillator at about 100kHz.
The IC works as both an encoder and a decoder.

We can see how it is possible to easily emulate this kind of signal with a simple microcontroller.
If we exclude the use of external dip switches, whose configuration is easily readable and we "integrate" them into the program, where they cannot be read by malicious people. Of course, to change the key we have to reprogram the component, but how many do you want you have changed to the position of the dip switches of your gate opener?
So it's not a strange choice, since this significantly increases security and allows us to use chips with a very low pin count, such as the 10F2xx or 12F5xx.
Nothing prevents you from using a chip with a sufficient number of pins (18-20), where 12 will be used to read external dip switches, allowing the coding to be changed without reprogramming, but this is not the purpose of the exercise.
The electrical circuit is simple:

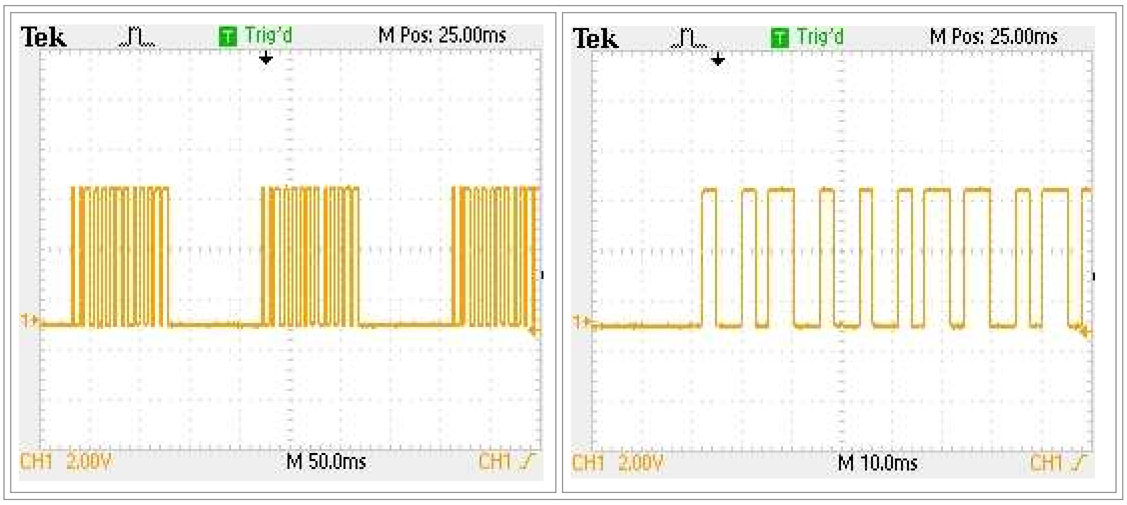


In particular, we add an LED that indicates the correct operation of the micro. Every time we energize the circuit, the encoded string will be output continuously.

Emulation on the LPCuB requires a few jumpers:



The output signal will be visible with an oscilloscope:



In the additional projects of the Baseline exercises there is also an application of the receiver, so that a key-lock pair can be made, applicable to any remote control method.

The program

Very simple, it consists only of a loop where the pulses are emitted according to what is coded. The times are made with the usual delay loops already seen, using the **Timer0** preloaded with a given value and waiting for the overflow:

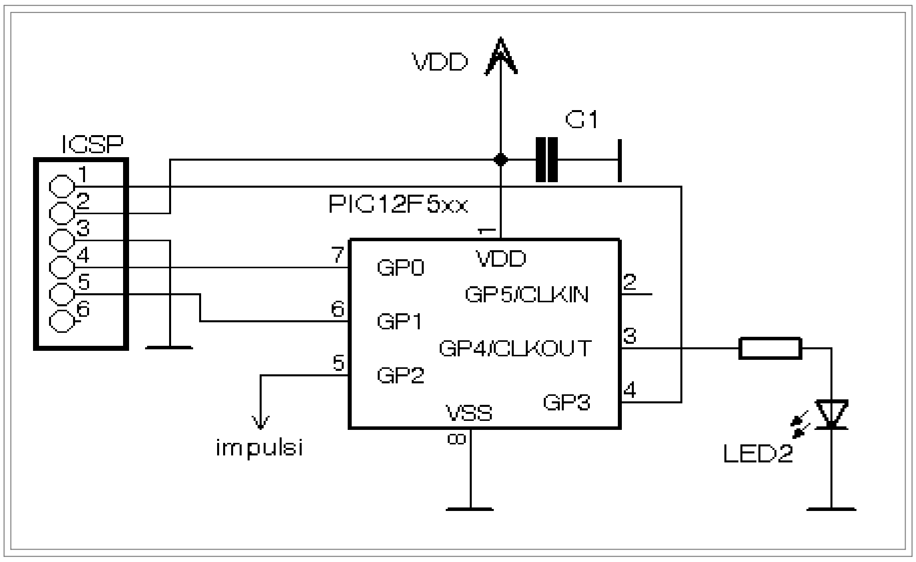
```
; Timings derived from the 4MHz internal clock
; 640us
us640 movlw . 94
      movwf TMR0
      movf TMR0,w
      skpz
       goto $-2
      retlw 0
```

A simple subroutine structure is used that use both levels of the Baseline stack.
The chain of successive `calls` encode a 12-bit word equal to 010001101111. Bits at 1 indicate an open dip switch and are executed by the `bit1` routine; conversely, bits at 0 indicate a closed dip switch and are executed by `the bit0` routine. By changing the sequence, other values of the 4096 possible can be encoded.

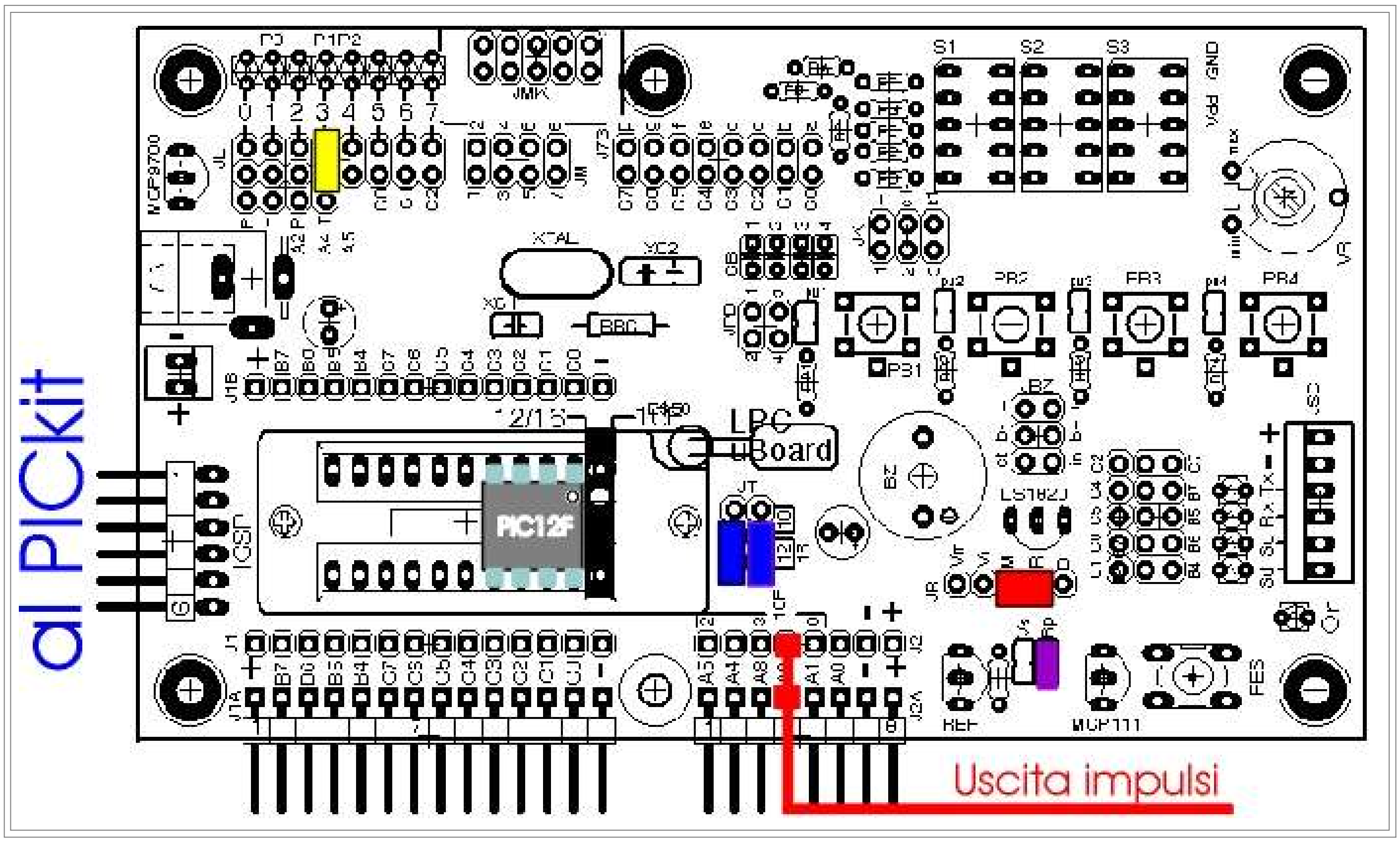
As mentioned, commercial realizations make use of an RC oscillator, therefore of low precision; It has been found that the pulse duration times vary between 0.6 and 1.4ms, but this is within the tolerances of the components and does not affect the decoding; although not with absolute precision, which has not been researched, the program uses the nominal values indicated by the data sheet, i.e. 960us, divided into 3 equal parts, therefore 320 and 640us, taking advantage of the internal clock at 4MHz.

The break time is not critical and can be extended at will; It depends on the receiving system and what it needs to do. Here we use about 11.25ms, but by increasing the time parameter it is possible to get over 160ms. The LED is lit at the beginning of the string and off before pause.

You might be wondering if it wasn't easier to simply insert the LED in parallel to the power supply, but it should be borne in mind that, by controlling it from the PIC, we have a confirmation that the program is working and that the pulse string is emitted.
MM53200 must be powered at voltages between 7 and 11V and this leads to the use of 9V PP or 12V MN23 batteries. The PIC can be powered from 2v, so even a 3V CR2302 type tablet cell can be fine; More than anything else, the voltage will depend on the transmission medium that follows the encoder (radio, infrared, ultrasound, etc.).
A source is provided for the 10F200/2/4/6 and a version for 12F508/09/10/19, but, of course, it can be easily carried over to any PIC; This is the related wiring diagram:



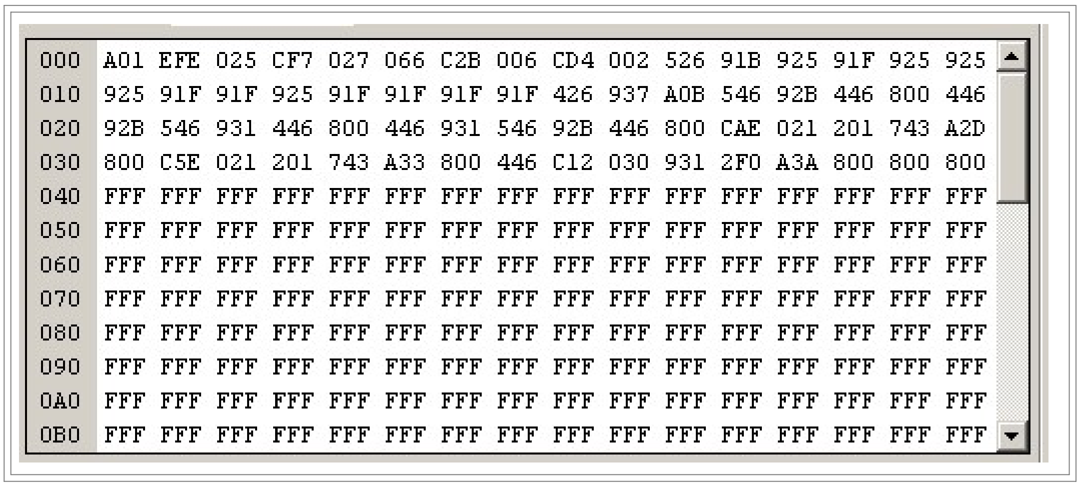
and the provision on the LPCuB:



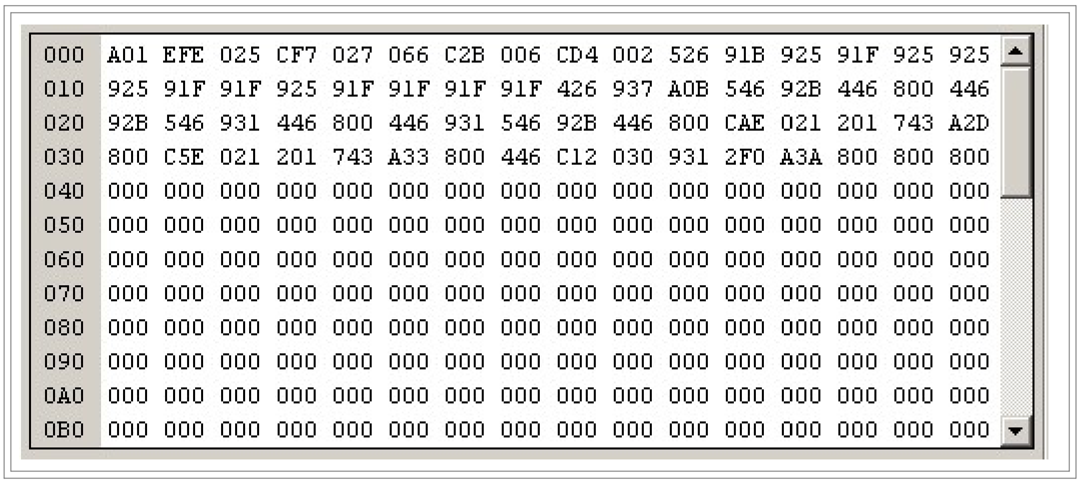
It should be noted that, as already pointed out, the differences concern only the specific characteristics of the chips and their initializations, while the core of the program is identical.

Code protection

In the provided sources, code protection is not enabled (**`_CP_OFF`**). This allows us to re-read the contents of the program memory.


```
000  A01 EFE 025 CF7 027 066 C2B 006 CD4 002 526 91B 925 91F 925 925
010  925 91F 91F 925 91F 91F 91F 91F 426 937 A0B 546 92B 446 800 446
020  92B 546 931 446 800 446 931 546 92B 446 800 CAE 021 201 743 A2D
030  800 C5E 021 201 743 A33 800 446 C12 030 931 2F0 A3A 800 800 800
040  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF
050  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF
060  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF
070  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF
080  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF
090  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF
0A0  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF
0B0  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF
```

The window is that of the **PickitProgrammer**. We observe that the program is very short and occupies only from 00h to 30Bh.
We recompile the source after enabling protection, replacing it in the configuration lines **`_CP_ON`**.
If we try to re-read the chip, we get this indication:

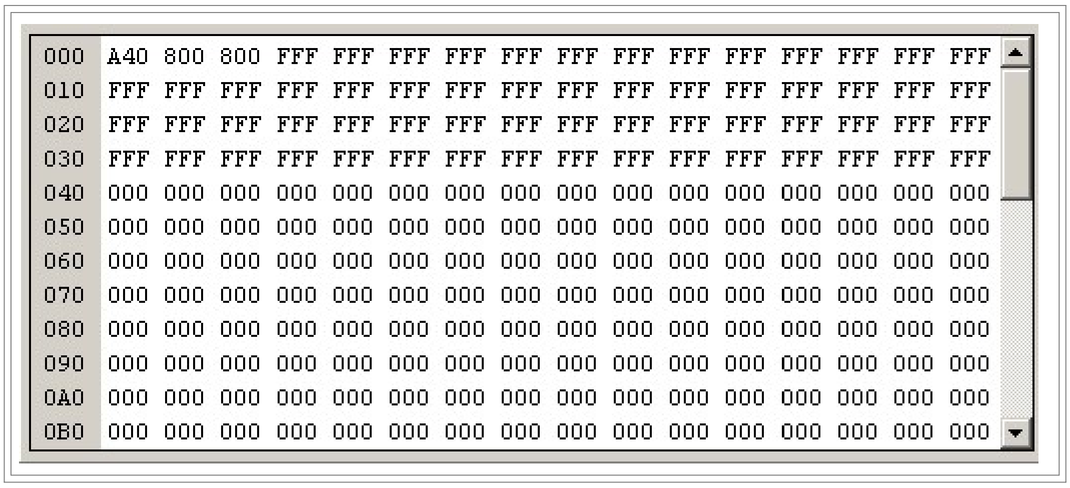

```
000  A01 EFE 025 CF7 027 066 C2B 006 CD4 002 526 91B 925 91F 925 925
010  925 91F 91F 925 91F 91F 91F 91F 426 937 A0B 546 92B 446 800 446
020  92B 546 931 446 800 446 931 546 92B 446 800 CAE 021 201 743 A2D
030  800 C5E 021 201 743 A33 800 446 C12 030 931 2F0 A3A 800 800 800
040  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
050  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
060  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
070  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
080  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
090  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
0A0  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
0B0  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
```

It appears to us as filled with 000 any location from 40h onwards, but the program is still accessible, since it is in the first 64 locations. Only what is written in subsequent locations cannot be read.
However, the encoding "key" is right in the first bytes. The solution is simple: uncomment the line that contains the second ORG and get:

```
   ORG 0x40        ; remain free the first 64 locations
```

```
                    ; for reread protection
```

Let's recompile the program and now the accessible part of the chip will show this:



```
000  A40 800 800 FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF
010  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF
020  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF
030  FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF FFF
040  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
050  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
060  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
070  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
080  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
090  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
0A0  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
0B0  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
```

now we can only read the first byte with the A40h instruction, i.e. goto 40h.
Locations up to 3Fh are not written and are therefore empty (FFFh, Flash status erased and not rewritten)
The program, which is now 40 hours or more, can no longer be read, appearing as 000 and security is safe.
The program memory space between the first instruction and 40h remains unused, but in other applications it may contain data or sections of code whose unwanted reading does not create problems for the security of the application.
For example, we can put a copyright on it.

16A_10F.asm

```
;**********************************************************************
; 16A_10F.asm
;----------------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 16A
;                     Program protection against re-reading.
;                     Send encoded signal in MM53200 emulation
;                     Dip switches in the program.

;    PIC            : 10F200/2/4/6
;    Support        : MPASM
;    Version        : 1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :Afg
;
;
;  Pin use :
;  ________________
;     10F200/2/4/6 @ 8 pin DIP       10F200/2/4/6 @ 6-pin SOT-23
;
;               |‾‾\/‾‾|                   *‾‾‾‾‾‾|
;          NC -|1    8|- GP3        GP0 -|1    6|- GP3
;         Vdd -|2    7|- Vss        Vss -|2    5|- Vdd
;         GP2 -|3    6|- NC         GP1 -|3    4|- GP2
;         GP1 -|4    5|- GP0            |______|
;               |______|
;
;                           DIP SOT
;    NC                      1:
;     Vdd                     2: 5: +
;       GP2/T0CKI/FOSC4/[COUT] 3:4: Out LED at Vss
;     GP1/[CIN-]/ICSPCLK      4: 3:
;     GP0/[CIN+]/ICSPDAT      5: 1:
;    NC                      6:
;     Vss                     7: 2: -
;     GP3/MCLR/VPP            8: 6:
;
;    [] only 204/6
;
;**********************************************************************
;            DEFINITION OF PORT USE
; GPIO map
; | 3 | 2 | 1 | 0 |
; |-----|-----|-----|-----|
; | in | LED |      |     |
;
;#define    GPIO,GP0   ;
#define    LED3 GPIO,GP1    ; LED between pins
and Vss #define            pulse GPIO,GP2
           ; Pulse Output
;#define    GPIO,GP3   ;
;
; ######################################################################
; Processor Definition
```

```
 #ifdef___10F200
        LIST      p=10F200
        #include <p10F200.inc>
 #endif
 #ifdef___10F202
        LIST      p=10F202
        #include <p10F202.inc>
 #endif
 #ifdef___10F204
        LIST      p=10F204
        #include <p10F204.inc>
#define proccomp
 #endif
 #ifdef___10F206
        LIST      p=10F206
        #include <p10F206.inc>
#define proccomp
 #endif
        Radix     DEC

; ######################################################################
;                          CONFIGURATION
;
; No WDT, no CP, pin4=GP3;
  __config _CP_OFF & _MCLRE_OFF & _WDT_OFF

; ######################################################################
;                          RAM
; general purpose RAM
         CBLOCK 0x10
    D1                ; ENDC Delay Counter

time    EQU .18       ; Break Time

; ######################################################################
;                        LOCAL MACRO
;
; LED status with LEDON MACRO
button pressed
        Bsf   LED3
       ENDM
; LED status with button released
LEDOFF MACRO
        Bcf   LED3
       ENDM
; high-level pulse output
PULSEH MACRO
        Bsf   pulse
       ENDM
; low-level pulse output PULSEL
MACRO
        Bcf   pulse
       ENDM

; ######################################################################
;                           RESET ENTRY
```

```
;
; Reset Vector
        ORG     0x00

        Goto     Main

        ;ORG      0x40       ; Vacate the first 64 locations
                                ; for reread protection

Main                        ; Initializations to RES
; Internal Oscillator Calibration
         ANDLW   0xFE
         movwf   OSCCAL

 #ifdef proccomp    ; only 10F204/6
; Disable Comparator
         movlw   B'11110111'
         MovWF   CMCON0
 #endif

         CLRF    GPIO        ; Latch outputs to 0

         movlw   b'00101011' ; GP4 and GP2
         out trio   GPIO

 ; Timer0 1:4 internal clock
          ;      b'11010111'
          ;        1-------     GPWU Disabled
          ;        -1------     GPPU disabled
          ;        --0-----     Internal Clock
          ;        ---1----     Falling
          ;        ----0---     prescaler to Timer0
          ;        -----100     1:4
         movlw   b'11010100'
         OPTION

         LEDON              ; LED Onler

; Virtual Dip Switches
; 010001101111 encoding
dipsw    Call    Start
         Call    bit1      ; Dip       OFF
         Call    bit0      ; Dip       ON
         Call    bit1      ; Dip       OFF
         Call    bit1      ; Dip       OFF
         Call    bit1      ; Dip       OFF
         Call    bit0      ; Dip       ON
         Call    bit0      ; Dip       ON
         Call    bit1      ; Dip       OFF
         Call    bit0      ; Dip       ON
         Call    bit0      ; Dip       ON
         Call    bit0      ; Dip       ON
         Call    bit0      ; Dip       ON

         LEDOFF            ; turn off LEDs
         Call    pause
         Goto    dipsw     ; Other Loop
```

```
; Start Start
 Pulse   PULSEH
         Call    us320
         PULSEL
         retlw   0

 ; DIPswitch ON
bit0     PULSEL
         Call    US320
         PULSEH
         call    US640
         PULSEL
         retlw   0

 ; DIPswitch
OFF bit1 PULSEL
         call    US640
         PULSEH
         call    US320
         PULSEL
         retlw   0

; Timings derived from the 4MHz internal clock
; 320us
US320   movlw   .174
        movwf   TMR0
        movf    TMR0,w
        skpz
         Goto   $-2
        retlw   0

; Timings derived from the 4MHz internal clock
; 640us
US640   movlw   .94
        movwf   TMR0
        movf    TMR0,w
        skpz
         Goto   $-2
        retlw   0

; Pause Duration Between Pause Pulse
Trains  PULSEL
        movlw   time
        movwf   D1
Psl     Call    US640
        decfsz D1,F
         Goto   Psl
        retlw   0


;*********************************************************************
;                            THE END
        END
```

16A_508.asm

```
;*******************************************************************
; 16A_508.asm
;-------------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 16A
;                     Program protection against re-reading.
;                     Send encoded signal in MM53200 emulation
;                     Dip switches in the program.
;
;    PIC            : 12F508/509/510/519
;    Support        : MPASM
;    Version        : V.519-1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :Afg
;
;-------------------------------------------------------------------
;
; Pin use :
;   _______________
;       12F5xx @ 8 pin
;
;                 |¯¯\/¯¯|
;            Vdd -|1    8|- Vss
;            GP5 -|2    7|- GP0
;            GP4 -|3    6|- GP1
;       GP3/MCLR -|4    5|- GP2
;                 |______|
;
;    Vdd                   1: ++
;    GP5/OSC1/CLKIN        2: ext RC osc
;    GP4/OSC2              3: Out LED
;    GP3/! MCLR/VPP        4: In   button
;    GP2/T0CKI             5: Pulse Out
;    GP1/ICSPCLK           6:
;    GP0/ICSPDAT           7:
;    Vss                   8: --
;
;*******************************************************************
;===================================================================
;            DEFINITION OF PORT USE
;
; GPIO map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|-----|
;|     |LED4 | BTN | Out |     |     |
;
;#define    GPIO,GP0    ;
;#define    GPIO,GP1    ;
#define   pulse GPIO,GP2 ; Pulse Output
;#define    GPIO,GP3    ;
#define   LED4 GPIO,GP4 ; LED control
;#define        GPIO,GP5 ; External RC Oscillator
;
```

```
; ######################################################################

time    EQU .18         ; Break Time

;********************************************************************
; ######################################################################

; Choice of #ifdef
  processor_12F509
         LIST      p=12F509
         #include <p12F509.inc>
#define procdig                     ; Processor without analog
  #endif
  #ifdef____12F508
        LIST      p=12F508
        #include <p12F508.inc>
#define procdig                     ; Processor without analog
  #endif
  #ifdef____12F519
         LIST      p=12F519
         #include <p12F519.inc>
#define procdig                     ; Processor without analog
  #endif
  #ifdef____12F510
        LIST      p=12F510
         #include <p12F510.inc>
#define procanalog                  ; Processor with analog
  #endif
          radix dec

; ######################################################################
;                        CONFIGURATION
;
#ifdef procdig          ; Code 12F508/509/519
; Internal Oscillator, No WDT, No CP, GP3
 __config _ExtRC_OSC & _WDT_OFF & _CP_OFF & _MCLRE_OFF
 #endif
  #ifdef procanalog       ; 12F510
 ; Internal Oscillator, 4MHz, No WDT, No CP, GP3
  __config _ExtRC_OSC & _IOSCFS_OFF & _CP_OFF & _WDT_OFF & _MCLRE_OFF
 #endif

; ######################################################################
;                        RAM
;
; general purpose RAM
        CBLOCK 0x10
    D1                 ; ENDC Delay Counter

; ######################################################################
;                        LOCAL MACRO
;
; LED status with LEDON MACRO
button pressed
        Bsf   LED4
       ENDM
```

```
; LED status with button released
LEDOFF MACRO
        Bcf   LED4
       ENDM
; high-level pulse output
PULSEH MACRO
        Bsf   pulse
       ENDM
; low-level pulse output PULSEL
MACRO
        Bcf   pulse
       ENDM


; ######################################################################
;=====================================================================
;                          RESET ENTRY
;
; Reset Vector
        ORG     0x00

        Goto   Main

      ; ORG     0x40       ; Vacate the first 64 locations
                          ; for reread protection

Main                      ; Reset Initializations
 #ifdef procanalog         ; only 12F510
; Disable CLRF Analog Inputs
                ADCON0
; Disable comparators to free the BCF digital function
                CM1CON0, C1ON
 #endif

        CLRF    GPIO        ; Latch outputs to 0

        movlw   b'00101011' ; GP4 and GP2
        out trio    GPIO

; Timer0 1:4 internal clock
         ;      b'11010111'
         ;        1-------     GPWU Disabled
         ;        -1------     GPPU disabled
         ;        --0-----     Internal Clock
         ;        ---1----     Falling
         ;        ----0---     prescaler to Timer0
         ;        -----100     1:4
        movlw   b'11010100'
        OPTION

        LEDON             ; LED Onler

; Virtual Dip Switches
; 010001101111 encoding
dipsw   Call    Start
        Call    bit1      ; DIPswitch OFF
        Call    bit0      ; DIPswitch ON
        Call    bit1        ; DIPswitch OFF
        Call    bit1        ; DIPswitch OFF
        Call    bit1        ; DIPswitch OFF
        Call    bit0        ; DIPswitch ON
        Call    bit0        ; DIPswitch ON
        Call    bit1        ; DIPswitch OFF
        Call    bit0        ; DIPswitch ON
        Call    bit0        ; DIPswitch ON
        Call    bit0        ; DIPswitch ON
        call    bit0        ; DIPswitch ON
        LEDOFF              ; turn off
        Call    pause         LEDs
        Goto    dipsw       ; Other Loop

; Start Pulse
Start   PULSEH
        call    US320
        PULSEL
        retlw   0

; DIPswitch ON
bit0    PULSEL
        Call    US320
        PULSEH
        call    US640
        PULSEL
        retlw   0

; DIPswitch
OFF bit1
        PULSEL
        call    US640
        PULSEH
        call    US320
        PULSEL
        retlw   0

; Timings derived from the 4MHz internal clock
; 320us
US320   movlw   .174
        movwf   TMR0
        movf    TMR0,w
        skpz
         Goto   $-2
        retlw   0

; Timings derived from the 4MHz internal clock
; 640us
US640   movlw   .94
        movwf   TMR0
        movf    TMR0,w
        skpz
         Goto   $-2
        retlw   0

; Pause Duration Between Pause Pulse
Trains  PULSEL
        movlw   time
        movwf   D1
Psl     Call    US640
        decfsz D1,F
         Goto   Psl
        retlw   0

;*********************************************************************
;                            THE END
        END
```