myRFID Project

Using ATtiny2313
8-bit MCU

Hardware & Firmware Information

Revision 1.00
January 2020
© Nikos Bovos
nikos@tng.gr

Preface

This document describes the hardware, firmware and software specifications of myRFID project. Effort was made for this document to contain all the necessary information without errors in a form that can help the novice user as well as the experienced one. The language used is as simple as possible in order for everyone to understand the functions of the project. All information presented here provided “as is”. No responsibility/liability is held true if any error or damage occur by the usage of the data presented in this project. You may use all info for personal use only – as always at your own risk!

Copyright notice

All information presented here might be trademarks or registered trademarks of their respective holders. For more information about components / info mentioned in this document, please refer to the manufacturers. Whenever is possible all trademarks are referred to the footer of each page. All trademarks appear in this document are refereed to as 'registered' trademarks (others are, others might not be).

myRFID Project Copyright information

My RFID Project (myRFID) and all accompanied material / documents are copyrighted by Nikos Bovos January 2020 except where mentioned copyrights of others. You may not dis-assemble or reverse engineer the provided with this project files You may use all info for personal non-profit use, for professional use please contact me. Please always give credit where deserved, I spend many-many hours designing and building the hardware, firmware and software and implementing all these features, when distributing retain the original format, information and files. The project is designed, tested and working with the stuff shown in this document for my own needs and my own setup. You may have to alter some things to fit your needs, no responsibility/liability by any means is held if you decide to use components others than mentioned here.

Contact info, bugs report and other info

If any bugs or errors found, please report them to nikos@tng.gr (alternative email is nbovos@gmail.com) in order to be corrected as soon as possible.
For any questions feel free to send me an email, I will reply as soon as possible.
Please note, the construction and usage of data of this project is at your own risk.
Donations via PayPal to keep coding to nikos@tng.gr would be very much appreciated :-)

This document was written using Apache Open Office™ 4.1.7 on my DELL D630 Laptop, running Microsoft Windows 7 Home Premium© and exported to PDF via the PDF export function.

Usually I'm told that i over-analyze things, I'm afraid I might do it also here, so I apologize in advance for that...

Open Office first build by Sun microsystems & the community (now acquired by Apache Software Foundation)
Apache Open Office is a trademark of Apache Software Foundation
Microsoft, Windows, Windows 7 Home Premium are registered trademarks of Microsoft Corporation

What is myRFID Project ?

myRFID project is an implementation of a control system consisting of MCU, LCD, RFID reader module, optionally a personal computer (PC) and Input/Output controlled devices. It is build around an 8-bit MCU and designed for my personal (and professional) use as well. The MCU selected for this project is my beloved Microchip's ATtiny2313* (formerly by Atmel) which has been widely used in these types of projects for many years. Although this project is running for 6 and more years in some variation, it will be presented here/now for the “basic” operations.

Basic specifications of the project

1) Read EM4100 compatible tags / cards (125 kHz) with RDM6300 module
2) Show tag read to 2x16 LCD.
3) Report also the tag read to Personal Computer.
4) Perform an action according to the card/tag read
5) MCU clock frequency = 4MHz external crystal
6) Communication at 9600bps, 8-data bits, 1-stop bit and No-parity.

And as always...My lovely quote :-)

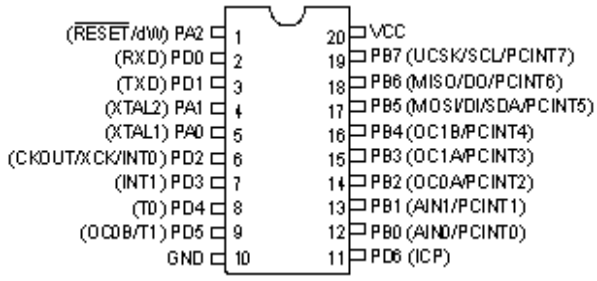
A must-say note! *Although no code accompany this document, the guidelines and information provided here are more than enough for anyone with "typical knowledge" to build this project and adjust it for his/her own needs. Because I spent too much of my – not so free - time to construct, write, build, test, update etc I'm awfully irritated when I'm asked to provide the full listing for "copy & paste” from others. I'm sorry to say that, but buying 2 boards, connect them with 2 cables and writing two lines of code to your computer does not help you in any way to learn how and more important why it works (even if that LCD screen is displaying 'Hello world !')...It's better to try and fail and then to repeat until you get it and learn a lot in the process, than to get the “working code” from others without even scratching your brain on how to do it.*

Nevertheless, in this project (like all my recent projects uploaded to AvrFreaks) some code will be provided to make you feel less “pissed” for reading all my bla-bla-blas without any practical example/code...

Atmel and AVR are registered trademarks of Microchip Technology
ATtiny2313 is a 8bit AVR micro-controller

The project's micro-controller - AVR ATtiny2313

myRFID is built with this micro-controller, the one in PDIP package as presented below (picture from datasheet):



The MCU has the AVR 8-bit RISC architecture
120 instructions, most executed in 1 clock cycle
32 general purpose registers x 8 bit each
2 kBytes of In-System programmable flash
128 bytes EPROM memory
128 bytes internal SRAM memory
One 8-bit Timer/Counter
One 16-bit Timer/Counter, one Programmable serial USART and up to 20 MIPS throughput at 20MHz clock and many other features, you can check on Microchip's web site this microcontroller on the following web page: https://www.microchip.com/wwwproducts/en/ATtiny2313

The ATtiny2313 MCU has 18 Input/Output programmable pins. It was selected mainly because it has been used in numerous of past projects and the number of I/O pins needed are below the maximum available I/O pins. As far as I know it's still in production....right ? Even though it is an “old” MCU will do the job just fine...

ATtiny2313 Port-Map

PORT A	Function	Direction	PORT B	Function	Direction
PA0	4 MHz Crystal	IN (No setup)	PB0	LCD data bit 0	OUT
PA1	4 MHz Crystal	IN (No setup)	PB1	LCD data bit 1	OUT
PA2	Used as $\overline{\text{RESET}}$	IN (No setup)	PB2	LCD data bit 2	OUT
			PB3	LCD data bit 3	OUT
			PB4	LCD data bit 4	OUT
			PB5	LCD data bit 5	OUT
			PB6	LCD data bit 6	OUT
			PB7	LCD data bit 7	OUT
			PORT D	Function	Direction
			PD0	USART RxD	IN (No setup)
			PD1	USART TxD	OUT (No setup)
			PD2	Device #1	OUT
			PD3	Device #2	OUT
			PD4	Device #3 / LED	OUT
			PD5	LCD 'RS' pin	OUT
			PD6	LCD 'E' pin	OUT

Useful Notes

a) Port A is a 3-bit Port (usually PA2 = $\overline{\text{RESET}}$)
b) Port B is an 8-bit port
c) Port D is an 7-bit port

Port pins mentioned as “No setup” are automatically set by the MCU by configuring it's modules (or fuses).

Principles of operation and various components used

The basic operation of this project is to read a RFID card/tag using a RDM6300 module and signal an output high or low (this will depend on the device to be controlled). It can be used as a stand-alone device (without the need of a personal computer), or it can be connected to a PC serial port (either to a USB-to-Serial converter or to a serial port – if there is one available).

From the port map is shown that there are 3 I/O pins available (PD2, PD3, PD4), so up to 3 external devices can be controlled (and by using a 3-to-8 demultiplexer up to 8 devices can be controlled - practically up to 7 devices since one combination of the three I/O will be used as “idle mode” without any external device been connected), or any other combination we want. By using a microcontroller we can program any action and timing to fit our needs. As always, one could select a larger MCU for the job, but...not me!

Throughout the years the same project have been used in 3 different combinations. One project has one RDM6300 module connected to PC serial port via a MAX232 converter (TxD) to transmit the RFID tag to a PC-Software (which records the date / time and the person that holds the card) and signals through the mcu an electronic door-latch (mcu is connected for RxD in a PC-serial port) and this is used as a time-attendance application, one project has two RDM6300 (or similar) modules connected to two PC-serial ports (TxD) and the MCU is signaling two parking bars (with pulses of 24V) for entering and leaving a parking spot (also the PC is recording the car & driver and time of entrance and departure) and this application is used for recording entrances and departures. The third one presented here today is the very “basic” application that has a 2x16 LCD screen that display the tag number read and according to a saved tag number in the EEPROM of the MCU is driving an external device (a door-latch). In this project a LED is connected to PORTD.4 pin of the MCU that flashes at frequency of 1Hz (to show that the project is not stuck) and flashes for 3 seconds while the correct card is read (and the output signal is “high”). The “correct” tag number is stored to MCU EEPROM and checked at run-time against the tag currently read.

In this version, since the RDM6300 module has UART-level output it is connected directly to the MCU RxD pin (PD0) and the TxD pin of the MCU (PD1) is connected to the PC Serial port through a simple Mosfet switch (if it is desired to record to PC the times that the door-latch is activated - practically I have used the PC function only during the time of development and now the device is used as stand-alone). By connecting the RxD of MCU to RDM6300 module and the TxD to PC, it is accomplished the task of reading and reporting each tag number using the one available USART of ATtiny2313... If transmission is not desired, the application can be used without a connection to PC and the TxD pin can be left unconncted.

LCD Display - 2 lines X 16 characters

For the visual output of the project, a 2 line by 16 characters LCD display is used. The particular LCD chosen is 1602A (HD44780 compatible display) shown in the following picture (this one has yellow matrix letters in blue background and backlight on-module):

The pin-out of the LCD display is:

Pin	Function	Pin	Function
1	Ground	**2**	Power supply (+5V DC)
3	Contrast adjust	**4**	Register Select (RS)
5	Read/Write data (R/$\overline{W}$)	**6**	Enable signal (E)
7	Data bit 0	**8**	Data bit 1
9	Data bit 2	**10**	Data bit 3
11	Data bit 4	**12**	Data bit 5
13	Data bit 6	**14**	Data bit 7
15	Backlight + (~ 4.5V DC)	**16**	Backlight - (Ground)

The signaling of the LCD is done by 8-bit data transfers between the MCU and the LCD. Pin 5 of the LCD (R/$\overline{W}$) is connected to GND since only writing to LCD is desired in this project. Timing and signaling of this LCD are compatible with Hitachi's HD44780U* commands - there are many sources and libraries available to make it work... As always I decided to use my “own driver” since I have used many times these LCDs in the past without any problems so far (and it's not difficult if you follow the guidelines of the datasheet)...

Before use, the LCD has to be initialized as instructed by Hitachi's datasheet. The timings used are a little “stretched” to cover all possible combinations of power and different LCD modules from various manufacturers. Since the initialization process is only done once in the beginning of the firmware (after RESET), there is no problem to have a little more “stretched” timings, since will not affect the operation of the system, as shown in the following piece of code that is used (and have been used for many years now):

HD44780U is a Dot Matrix Liquid Crystal Display Controller/Driver LSI IC by Hitachi

```
;*********************************************************************
;* Subroutine        : InitLCD
;* Uses              : genio
;* Purpose           : Initialize LCD
;*                   : LCD will be put for 8-bit transfers and timing is
;*                   : little "stretched" compared to HD44780U timings
;*                   : to cover power-supply cases (slow and normal)
;*********************************************************************
InitLCD:
     rcall         del200ms              ; Wait more than 40ms after Vcc=2,7V

     ldi           genio, LCD_cmd8bit    ; Load command LCD_cmd8bit to genio
     rcall         LCD_Command           ; Send command to LCD
     rcall         del5ms                ; Delay more than 4.1ms

     ldi           genio, LCD_cmd8bit    ; Load command LCD_cmd8bit to genio
     rcall         LCD_Command           ; Send command to LCD
     rcall         del5ms                ; Delay more than 100us

     ldi           genio, LCD_cmd8bit    ; Load command LCD_cmd8bit to genio
     rcall         LCD_Command           ; Send command to LCD
     rcall         del5ms                ; Delay more than 100us

     ldi           genio, LCD_DispNoBlink; Load LCD_DispNoBlink to genio
     rcall         LCD_Command           ; Send command to LCD
     rcall         del5ms                ; Delay more than 100us


     ldi           genio, LCD_Cursor     ; Load Increase Address + Cursor
     rcall         LCD_Command           ; Send command to LCD
     rcall         del5ms                ; Delay more than 100us

     ret                                 ; Return from subroutine...Init Done.
```

- genio is defined as high register (r16-r31)
- del200ms subroutine produces 200ms delay
- del5ms subroutine produces 5ms delay
- LCD_Command pulses the 'E' pin of the LCD in 50 microseconds intervals
- Equivalents of commands above (LCD_xxxx) from HD44780U datasheet are:

```
LCD_cmd8bit     = $38 = 56dec = 0b00111000
LCD_DispNoBlink = $0E = 14dec = 0b00001110
LCD_Cursor      = $06 =  6dec = 0b00000110
```

After initialization the LCD is put to 8-bit mode with no-blinking cursor (for blinking cursor the above LCD_DispNoBlink must be replaced by $0F = 15dec = 0b00001111)

Note also that if using "Return Home" instruction (0b0000001x) or "Clear Display" instruction (0b00000001) in any part of your program, these instructions have execution time of 1,64ms max, so you must wait accordingly - or check the Busy Flag of the LCD (but this has to be done with the R/$\overline{W}$ pin at +5V, so must be connected to MCU to control it's level, with +5V for reading busy flag and Read data or address of LCD module and 0V for writing to display and address. Because in this project the R/$\overline{W}$ pin is connected to 0V, then it is necessary to wait for at least the maximum time mentioned in the HD44780U datasheet). All other instructions take up to max. 40 microseconds to complete.

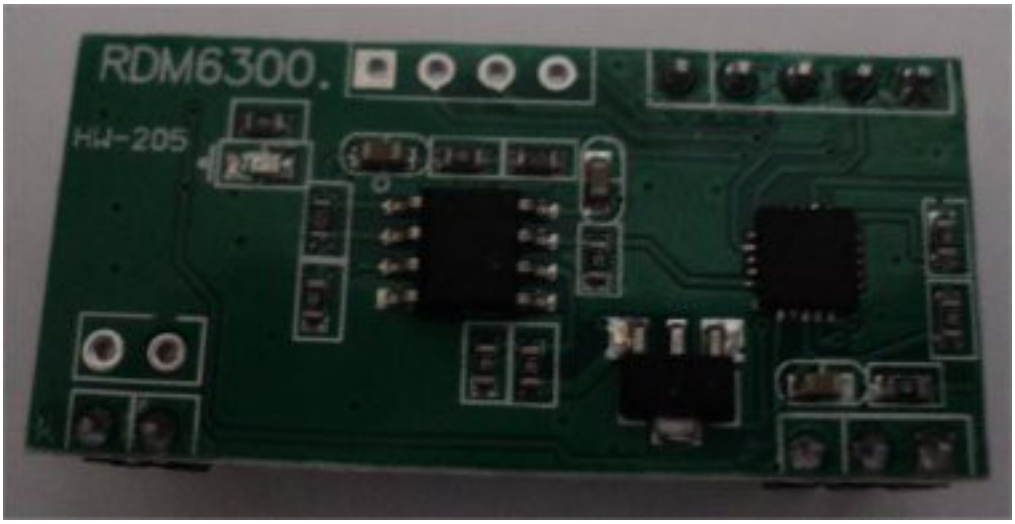
After LCD initialization, a welcome message is displayed for about 2 seconds and then the “default” message appears and the MCU is waiting a tag to be read... The following pictures show that:

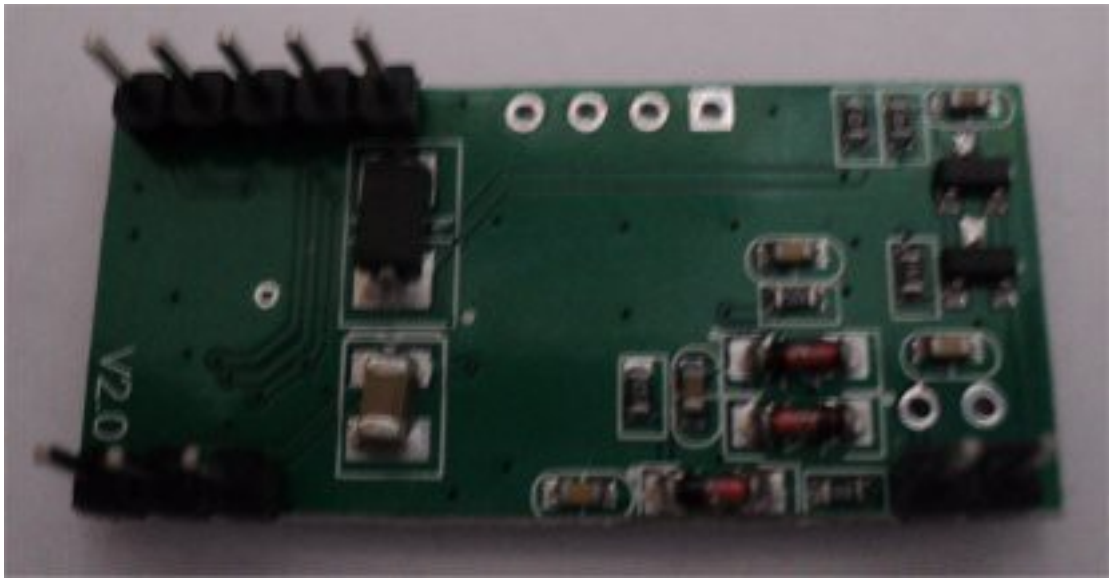
“Cosmetics” on the pictures above are by the protective thin film that hasn't yet been removed from the LCD module – because it will not be used in this project in the end...

RFID module RDM6300

This one can be found in many variations and modes of operation, even in different versions. The “standard” RDM630 transmits data using Weigang 26 format in packets of 8 bytes x 4 packets (D0 – D7 , D10-D17, D20-D27, D30-D37) but it is out of the scope of this document to describe it.

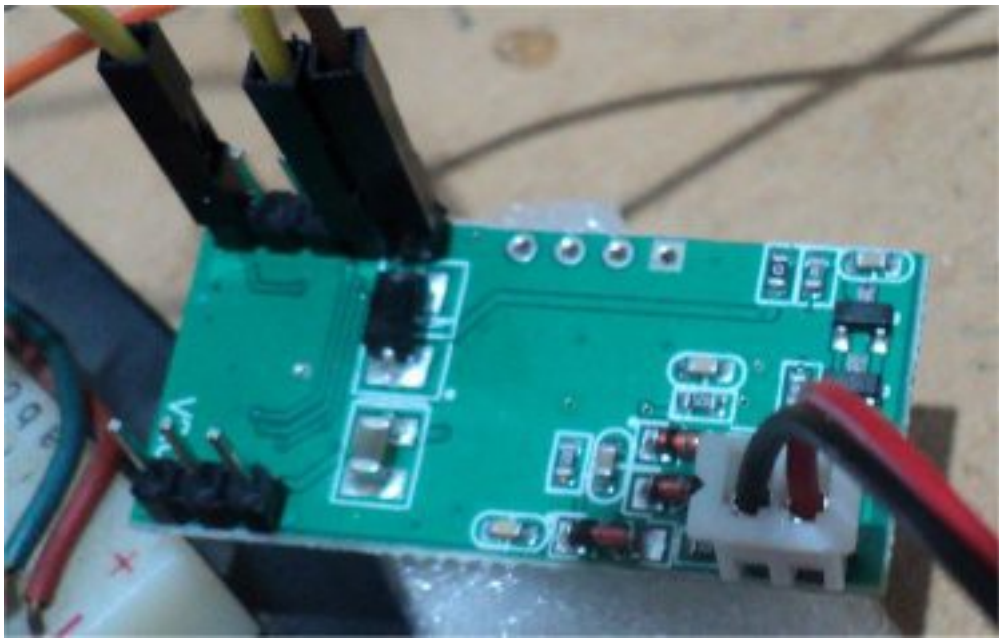
The RDM6300 module is a RDM630 with ASCII output of data stream and this is the one that is used in this project. Both transmit on the TxD pin in serial TTL form at 9600bps, 8 data bits, 1 stop bit and No-parity. The full formation of the stream transmitted is: a “start” character ($02), followed by 10 ASCII characters, two ASCII Checksum characters (total XOR of data characters) and a “stop” character ($03). The one used in this project is presented below:





According to “general” RDM630 datasheet, from left to right: Up-Left Pins 1 to 5 with 1=TxD, 2=RxD, 3=N.C, 4=Gnd, 5=+5V DC, Down-Left Pins 1 to 3 with 1=Led, 2=+5V DC, 3=GND and down-right Pins 2 to 1 with 2=Antenna 2 and 1=Antenna 1.

Actually the 3-pin header was not used since the module connected as shown in the following picture (pins 4,5=Gnd , +5V and pin 1 is the TxD pin, 2 and 3 unconnected)



The 3-pin header down-left was unconnected (Vcc and Gnd are common to 2 connectors), pin 1 is a 'Led' connection but the Led was constantly ON – datasheet state nothing about, some guys that have used this module say that it is on during operation and turns off when a RFID tag is read, but I haven't seen it turn off (maybe on different model)...Antenna connection makes no difference if red wire is in Ant1 or Ant2.

After reading a tag, 12 characters transmitted (10 being the data and 2 the checksum characters). To give an example, a tag was read and the resulting data stream was = ($02)(0106E88978)(1E)($03) (transmitted without the parenthesis – I placed them to distinguish the comprising elements of the data stream)

- Full Stream = □0106E889781E▪
- $02 = Start character (shown as □ above)
- 0106E88978 = 10 data characters
- 1E = 2 checksum characters
- $03 = Stop character (shown as ▪ above)

To validate the above data, a total XOR for the entire data stream shown above in pairs of 2 characters each, that is:

($01) XOR ($06) XOR ($E8) XOR ($89) XOR ($78) = $1E

which is correct and so the transmission is considered to be valid (a final XOR between the resulting XOR byte and transmitted checksum must always result to $00 since a XOR of a byte with itself always clears the byte).

Note that validation has to be made in digit-pairs of the received data. It can't be made by XORing each character (or the equivalent hex value of the ASCII character). In the example, taking equivalent values: $30 (for '0') XOR $31 (for '1') etc etc will produce the value $74 which is not correct (the final XOR with checksum will give $6A which is different than the expected $00, so invalid).

The subroutine to validate the checksum is shown below.. It's not rocket science but it does the job just fine in the simplest way possible...

```
;*********************************************************************
;* Subroutine        : ValidateCS
;* Purpose           : Validate Checksum against received checksum data
;*                     If checksum is valid set 'T' bit in SREG
;*                     Altered registers are not preserved here,
;*                     since it's not necessary.
;*
;* Uses              : genio    (result from Chars_2_To_1Byte + Fetch from RAM)
;*                     genioSec (counter for counting the bytes loaded/XORed)
;*                     Checksum (to hold the result of XOR)
;*                     medium   (hold low for Chars_2_To_1Byte conversion)
;*                     large    (hold high for Chars_2_To_1Byte conversion)
;*                     Z-pair   (hold address to fetch from SRAM)
;*
;* Created           : 02 January 2020
;*********************************************************************
ValidateCS:
     clt                            ; Clear 'T' bit in SREG
     ldi       ZH, high(dRecv + 1)  ; Load ZH with address of dRecv+1 in SRAM
     ldi       ZL, low(dRecv + 1)   ; Load ZL with address of dRecv+1 in SRAM
     ld        genio, Z+            ; Get byte to genio, increase Z-pointer
     mov       large, genio         ; Copy genio to large
     ld        genio, Z+            ; Get next byte to genio, increase Z
     mov       medium, genio        ; Copy genio to medium
     rcall     Chars_2_To_1Byte     ; Convert 2 ASCII->1 byte, result on genio
     mov       CheckSum, genio      ; Copy to Checksum - starting byte for XOR
     ldi       genioSec, 2          ; 2 bytes fetched so far...
Valid_Loop:
     ld        genio, Z+            ; Get next byte to genio, increase Z
     mov       large, genio         ; Copy genio to large
     ld        genio, Z+            ; Get next byte to genio
     mov       medium, genio        ; Copy genio to medium
     subi      genioSec, -2         ; Add 2 to genioSec counter (subtract -2)
     rcall     Chars_2_To_1Byte     ; Convert to 1 byte, result on genio
     eor       CheckSum, genio      ; XOR CheckSum with genio,result on CheckSum
     cpi       genioSec, 12         ; Compare genioSec to 12 (max bytes to fetch)
     breq      Valid_Check          ; If genioSec=12 finished XORs, check result
     rjmp      Valid_Loop           ; else jump back to repeat for next byte
Valid_Check:
     cpi       Checksum, 0          ; Compare CheckSum with 0 (or tst Checksum)
     brne      Valid_Out            ; If <> 0 - IT'S INVALID don't alter 'T'
     set                            ; Else i come here, CheckSum = 0 set 'T'
Valid_Out:
     ret                            ; Return from subroutine, came with rcall
```

The routine Chars_2_To_1Byte convert 2 ASCII characters to one byte (for instance 2 ASCII '1F' (equivalents large=$31 , medium=$46) will be converted to $1F (by the “simple way”, subtract 48, check for 0-9 , update for A-F etc). Gets large, medium (high registers) and produce the result to genio (high register also). If after calling this routine the 'T' bit in Status Register = 1 then the stream is valid, else it is discarded and not displayed or checked for the “correct” RFID value stored in EEPROM. The “dRecv” is the USART receive buffer address declared in SRAM which holds the full stream received from RDM6300 module (along with the start and stop characters), that's why fetching of bytes start from dRecv+1 address (after Start byte) and genioSec register (high register also) count the number of bytes, that's why it is checked for equality to maximum value (=12).

Led operation and flashing

In this implementation, a led is connected in series with a 330 Ohm resistor to PORTD4 of the mcu and to common ground. During “normal” operation (before a tag is read), this led flashes at a frequency of 1Hz. The timing is accomplished by setting Timer-1 (16 bit timer) to overflow every 1 second. In every timer overflow interrupt, PD4 output is toggled (by writing '1' to PIN register). When a tag is successfully read, Timer-1 is configured for 200ms thus flashing for the time the output is driven high (approximately 3 seconds) before returning to the normal state of 1sec flash (and the output goes low).

The following code is used in the initialization routine of the project to setup Timer-1 for 1Hz timeout:

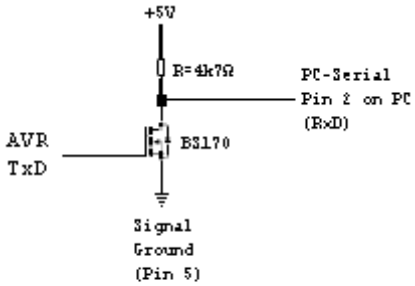
```
ldi         genio, (1<<CS12) | (1<<CS10)  ; Shift 1 to CS12, CS10 bit positions
out         TCCR1B, genio                 ; Set Timer-1 prescaler value = 1024
ldi         genio, (1<<TOIE1)             ; Shift 1 to TOIE1 bit position (bit 7)
out         TIMSK, genio                  ; Enable Timer-1 Overflow Interrupt
ldi         genio, high(61629)            ; Load High byte of value 61629 to genio
ldi         genioSec, low(61629)          ; Load Low byte of value 61629 to genioSec
out         TCNT1H, genio                 ; Set High Byte of TIMER-1 Counter FIRST
out         TCNT1L, genioSec              ; Set Low Byte of TIMER-1 Counter after.
```

The project is using Timer-1 prescaler of 1024 (and thus Timer-1 frequency of approx. 3,9 kHz) and so for 1 second 3906 timer 'ticks' (increments) must occur, thus the startup value of Timer-1 will be 65535-3906 = 61629.

For setting for 200ms overflow interrupt, the starting value of timer will be 64735 for 65535-64735 = 800 Timer-1 steps to overflow every 200ms.

Connect MCU to PC

This is an optional connection and is made during the development of this project for get the ID of every tag read. To cover as little space as possible, the following circuit was used:



Connect MCU TxD to PC-Serial RxD
Nikos Bovos December 2019

Not very much to analyze, signaling is TTL compatible (0V to 5V) and worked correctly with a standard PC and a laptop computer that was tested for all RFID tags and cards read. If the device will be used to record the tags also, it can be used along with a simple PC-software that communicates at 9600 bps with 8 data bits, 1 stop bit and No parity.

By using a crystal oscillator for the MCU guarantees that the above circuit will work all the time (though not tested on extreme environments). Alternatively you can select the internal RC oscillator at 4MHz of the ATtiny2313 by programming the fuses accordingly, but it necessary to calibrate the internal oscillator for the power supply and operating temperature of the device (guidelines for calibration are provided by the manufacturer and various other sources, try www.avrfreaks.net for many articles about it), but since no other I/O pins were needed and the cost of a crystal is very low, it is a pity to bother calibrating the internal RC oscillator than to place a crystal (and two 22pF capacitors) for accurate clock of the MCU.

For this project NO PCB was made, since it is not to be used. A simple breadboard with all the stuff was wired to test the code. This project is working as described in previous pages for some years now without problems (so far...). Today I decided after resting on holidays to write some notes about the “general usage” of the RDM6300 module and the usage of a LCD display (I think that all other information not presented here, are the same for any project involving a microcontroller)...

The firmware is written in assembly, total space is only 644 words, about 31% of the available MCU Flash Memory, which is more than fine, leaving 1404 words available for any simple or complex task... The RDM6300 is a small cheap board that can add RFID support to a wide variety of your projects, with or without the usage of a MCU.

Thanks for reading and baring with all my bla-bla-blas...
Happy new year to all, take care, have fun, see ya!