Versione N° 1.0

LaurTec

Modulo NRF24L01+

Libreria e utilizzo

Autore : *Marcello Pinna*

ID: UP0015-IT

INFORMATIVA

Come prescritto dall'art. 1, comma 1, della legge 21 maggio 2004 n.128, l'autore avvisa di aver assolto, per la seguente opera dell'ingegno, a tutti gli obblighi della legge 22 Aprile del 1941 n. 633, sulla tutela del diritto d'autore.
Tutti i diritti di questa opera sono riservati. Ogni riproduzione ed ogni altra forma di diffusione al pubblico dell'opera, o parte di essa, senza un'autorizzazione scritta dell'autore, rappresenta una violazione della legge che tutela il diritto d'autore, in particolare non ne è consentito un utilizzo per trarne profitto.

La mancata osservanza della legge 22 Aprile del 1941 n. 633 è perseguibile con la reclusione o sanzione pecuniaria, come descritto al Titolo III, Capo III, Sezione II.
A norma dell'art. 70 è comunque consentito, per scopi di critica o discussione, il riassunto e la citazione, accompagnati dalla menzione del titolo dell'opera e dal nome dell'autore.

AVVERTENZE

I progetti presentati non hanno la marcatura CE, quindi non possono essere utilizzati per scopi commerciali nella Comunità Economica Europea.

Chiunque decida di far uso delle nozioni riportate nella seguente opera o decida di realizzare i circuiti proposti, è tenuto pertanto a prestare la massima attenzione in osservanza alle normative in vigore sulla sicurezza.

L'autore declina ogni responsabilità per eventuali danni causati a persone, animali o cose derivante dall'utilizzo diretto o indiretto del materiale, dei dispositivi o del software presentati nella seguente opera.
Si fa inoltre presente che quanto riportato viene fornito così com'è, a solo scopo didattico e formativo, senza garanzia alcuna della sua correttezza.

L'autore ringrazia anticipatamente per la segnalazione di ogni errore.

Tutti i marchi citati in quest'opera sono dei rispettivi proprietari.

Indice

Introduzione

Nell'ambito di diverse applicazioni elettroniche, siano esse domotiche, robotiche, hobbistiche o industriali, si ha sempre la necessità di trasmettere e ricevere dati a distanza. Attualmente, i sistemi utilizzati sono di tre tipi: elettrici, ottici, e wireless. Mentre i primi due sistemi utilizzano cavi più o meno ingombranti, il terzo utilizza esclusivamente le onde radio, che consentono comunque una maggiore "libertà di movimento". Per applicazioni hobbistiche, e non solo, applicate a domotica e robotica, alcune fabbriche del settore hanno creato diversi tipi di schede dedicate (moduli) con le quali è possibile trasferire dati via radio. Il più famoso è sicuramente il modulo Xbee, prodotto dalla Digi, un altro meno famoso ma non per questo meno performante a livello di prestazioni è il modulo NRF24L01+, prodotto dalla Nordic Semiconductor. In questo tutorial verranno descritte le caratteristiche di quest'ultimo e la libreria dedicata che ho scritto, permettendone così il corretto funzionamento. Raccomando comunque a tutti una lettura attenta lettura del datasheet per meglio capire le sue peculiarità.

Nota

Il sistema presentato nell'articolo presenta l'utilizzo di moduli RF. I sistemi risultanti non sono certificati secondo i relativi organi di competenza. In particolare non possiedono la certificazione CE, FCC. I sistemi presentati devono essere pertanto considerati solo sistemi di sviluppo e valutazione. Potrebbero essere soggetti ad interferenze elettromagnetiche ed essere fonte di disturbo elettromagnetico per altri dispositivi elettronici.

Applicazioni

Il transceiver NRF24L01 permette di realizzare sistemi a controllo remoto dei più disparati. Gli esempi che sono riportati nell'articolo sono solo di tipo funzionale per mostrare l'utilizzo della libreria. Applicazioni tipiche sono il controllo remoto di carichi, quali le luci, il portone elettrico o anche la trasmissione di informazioni da parte di sensori remoti, come per esempio la temperatura, umidità al fine di permettere un controllo remoto della caldaia in funzione dei parametri ambientali.

Descrizione del Modulo

Dal momento che il transceiver NRF24L01+ è un sistema ad alta frequenza che lavora a 2.4GHz, è bene far uso di moduli che contengono già il transceiver e la circuiteria e per l'adattamento d'impedenza tra il buffer di trasmissione, il modulo LNA (*Low Noise Amplifier*) e l'antenna. Il modulo preso in considerazione nell'articolo è dalle misure molto compatte, circa 1,5 cm x 2,9 cm, di cui esistono due versioni: una con antenna integrata direttamente nel circuito stampato, che copre 100 metri di distanza, e l'altra con antenna esterna, che copre 1000 metri di distanza (Open space – spazio aperto). Entrambe queste versioni hanno un costo veramente irrisorio, (la seconda versione ha un prezzo inferiore ai 7 euro). La banda di frequenza del modulo è compresa 2,4GHz e 2,525 GHz, ovvero compatibile con la banda nominata ISM (Industria, Scienza, Medicina). I consumi sono piuttosto modesti con corrente di picco in modalità RX/TX inferiore a 14 mA. In modalità risparmio energetico i consumi sono inferiori a 1μA. La tensione operativa è compresa tra 1,9 e 3,6 volt, adatto quindi per applicazioni Ultra Low Power.
Il canale di trasmissione/ricezione può essere liberamente impostato nel software con un passo di 1MHz, determinando così la frequenza di lavoro finale. Allo stesso modo è possibile impostare la sua velocità di trasmissione, espressa in Mps, ed la potenza, espressa in decibel da utilizzare. La piedinatura ha un passo standard da 2,54 mm e la modalità di comunicazione utilizzata è l'SPI. In Tabella 1 sono riportati le principali differenze tra il modulo in questione ed il modulo Xbee, a parità di una copertura in distanza pari a 100 metri.

Modulo	NRF24L01+	XBEE
Marchio	Nordic	DIGI
Dimensioni approssimative	1,5 x 2,9 cm	2,4 x 2,8 cm
Tensione operativa	1,9 – 3,6 volt	3,3 volt
Piedinatura	2,54 mm (passo standard)	2 mm
Frequenza operativa	2,4 GHz	2,4 GHz
Modalità di comunicazione	SPI	USART
Bidirezionale	Si	Si
Low power	Si	Si
Prezzo	Inferiore a 2 euro	Superiore a 20 euro

Tabella 1: *Confronto tra i moduli basati sul NRF24L01+ e i moduli XBee.*

A bordo di questo modulo, oltre al chip NRF24L01+ e tutto l'hardware necessario a corredo, sono presenti otto linee di collegamento identificate dalla serigrafia, come:

1. GND
2. VCC
3. CE Chip Enable
4. CSN Chip Select NOT
5. SCK Clock
6. MOSI Master Out Slave In

7. MISO Master In Slave Out
8. IRQ Interrupt Request

Le prime due servono ovviamente per alimentare il modulo ad una tensione compresa tra 1,9V e 3,6V. La linea CE, in combinazione con il bit PRIM_RX, presente nel registro CONFIG, serve per impostarlo in trasmissione o ricezione. La linea CSN serve per abilitare o meno la scrittura dei vari registri. Le linee SCK - MOSI - MISO sono esclusivamente dedicate alla comunicazione SPI. L'ultima linea, IRQ, serve per utilizzare un eventuale interrupt esterno. In Figura 1 si riporta un'immagine del modulo in questione:

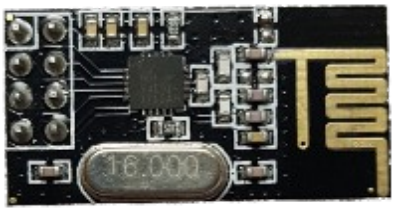

Figura 1: *Modulo utilizzato per il transceiver NRF24L01+.*

Poiché l'NRF24L01+ non ha pin 5V tolleranti, un suggerimento che vorrei dare per alimentarlo senza che subisca danni è di usare un adattatore dedicato, dal costo inferiore anch'esso ai 2 euro. Questo adattatore, alimentato con una tensione di 5V, fornisce in uscita una tensione di 3,3V, idonea perfettamente allo scopo. In Figura 2 è ripotata un'immagine di questo secondo modulo.

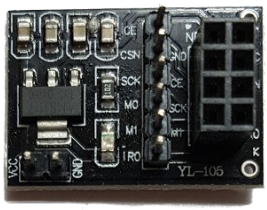

Figura 2: *Modulo con Level shifter ed LDO 5V → 3.3V.*

Registri del modulo

Affinché il modulo vada in trasmissione o ricezione, esso deve essere propriamente configurato, cioè impostato, scrivendo sui diversi registri presenti all'interno del suo chip. Questo può essere fatto utilizzando le tre linee SPI viste in precedenza. Al fine di facilitare la fase di configurazione, è stata creata una libreria dedicata che svolge questo compito, descritta successivamente. Senza entrare in ogni singolo dettaglio, si riporta qui l'elenco di questi registri, almeno quelli più importanti, per l'impostazione di un lavoro minimale del modulo, rimandando un ulteriore approfondimento da parte dell'utente alla lettura del relativo datasheet:

- CONFIG
- EN_AA
- EN_RXADDR

- SETUP_AW
- SETUP_RETR
- RF_CH
- RF_SETUP
- TX_ADDR
- RX_ADDR_P0
- RX_PW_P0
- STATUS

Introduzione alla libreria

Allo scopo di mettere l'utente nelle condizioni di capire meglio la logica di funzionamento della libreria, ed allo stesso tempo invogliarlo a migliorarla da sé, riporto qui alcune definizioni estratte dal datasheet:

Enhanced ShockBurst (ESB)

E' un protocollo di base della Nordic, che supporta la comunicazione bidirezionale di pacchetti di dati. ESB offre la gestione automatica delle transazioni dei pacchetti per implementare facilmente un collegamento affidabile. Una transazione è uno scambio di pacchetti tra due ricetrasmettitori, con un ricetrasmettitore che funge da ricevitore primario (PRX) e l'altro ricetrasmettitore che funge da trasmettitore primario (PTX). Le sue caratteristiche primarie sono le seguenti:

- Lunghezza dinamica del *payload* (da 1 a 32 byte)
- Manipolazione automatica del pacchetto
- Transazione automatica del pacchetto: *Auto Acknowledgement* con o senza *payload* e Auto Ritrasmissione pacchetto dati.
- 6 linee data pipe per una rete di collegamento a stella

Il registro interessato per la sua abilitazione è EN_AA. Al fine di garantire una transazione sicura e priva di errori, insieme ai dati che si vogliono inviare vengono aggiunti altri byte di controllo da parte della modalità *Enhanded ShockBurst*. In definitiva, la costruzione finale di un pacchetto dati risulta composto da:

- PREAMBLE (1 byte): E' formato dal byte 10101010 e serve come informazione di inizio pacchetto e come sincronizzazione con il ricevitore.
- ADDRESS (3 - 5 byte): Byte dell'indirizzo esadecimale di trasmissione e ricezione del pacchetto stesso
- PAYLOAD (1 - 32 byte): Byte dei dati che si vogliono inviare e ricevere.
- CRC (1 - 2 byte): Byte di controllo che serve/servono al modulo ricevente per la verifica di eventuali errori di trasmissione. Nel caso di CRC non coerente, la modalità ESB avvia la procedura di ritrasmissione dati da parte del modulo trasmittente, fino a ricevimento di un pacchetto ACK di convalida, inviato dal modulo ricevente.

E' importante, in ogni caso, evidenziare il fatto che del pacchetto dati solo il *payload* viene reso disponibile. Tutto il resto del pacchetto non viene reso visibile.

Data Pipe

Per data pipe si intendono 6 linee parallele in ricezione (da 0 a 5), identificate ognuna con un proprio indirizzo esadecimale univoco. In pratica, si riassumono in canali logici abbinati ad altrettanti canali fisici in radiofrequenza. I registri interessati sono tre: EN_AA, EN_RXADDR e STATUS. Il primo abilita il loro Auto ACK, il secondo imposta il loro indirizzo di scrittura e lettura, il terzo li seleziona.

ACK

Per ACK (*Acknowledgment*) si intende un pacchetto di convalida dati, creato automaticamente da *Enhanced ShockBurst*, al fine di informare il modulo trasmittente che la ricezione dei dati da parte del modulo ricevente è avvenuta in modo corretto e che può inviare quelli successivi. Esso può essere vuoto (impostazione predefinita) oppure con *payload*. (In quest'ultimo caso va abilitato il bit EN_ACK_PAY nel registro FEATURE).

Ritrasmissione dati

Nel caso sfortunato in cui la ricezione dei dati per qualche motivo fallisce, il modulo PTX ritrasmette al modulo PRX lo stesso pacchetto dati per un numero di volte impostato nel registro SETUP_RETR. Se il numero impostato è uguale a 0 (ritrasmissione disabilitata) non ci sarà nessuna ritrasmissione dati ma verrà generato immediatamente un interrupt, se abilitato, nel bit 4 (MAX_RT) del registro STATUS.

Frequenza di lavoro

Come affermato in precedenza, il modulo può trasmettere e ricevere dati in una gamma di frequenza che va da 2,4MHz a 2,525 GHz (2400 - 2525 MHz) e tale frequenza viene stabilita in base al canale di comunicazione scelto per il programma, con step di 1 MHz. Ciò significa che si hanno a disposizione esattamente 2525 - 2400 = 125 canali, più il canale di partenza, il canale 0, anch'esso da considerare e rappresentato dalla frequenza base di 2,4 GHz, quindi in totale 126 canali. La frequenza di lavoro sarà stabilita tramite l'impostazione del registro RF_CH, che conterrà nel suo interno il numero, in formato esadecimale, del canale scelto. La formula utilizzata per stabilire tale frequenza è la seguente:

2400 + RF_CH = Frequenza di lavoro espressa in MHz

A titolo di esempio, se si sceglie il canale 8, si dovrà impostare 0x08 nel registro RF_CH e la frequenza risultante sarà uguale a 2400 + 8 = 2408 MHz (2,408 GHz). I registri interessati sono: RF_CH e RF_SETUP.

Indirizzi di scrittura e lettura

Per poter comunicare tra loro, oltre allo stesso canale di comunicazione, (ovvero la stessa frequenza), i due moduli TX e RX hanno bisogno anche di un indirizzo logico. Tale indirizzo può essere scelto in modo arbitrario ma deve essere uguale per entrambi i moduli. Ciò significa che entrambi i registri TX_ADDR e RX_ADDR_Px (dove Px equivale alla linea data pipe scelta) devono essere impostati con lo stesso valore. Questo indirizzo può essere composto da un minimo di 3 ad un massimo di 5 byte.

Libreria per il transceiver NRF24L01+

Questa breve descrizione si riferisce esclusivamente alla libreria nrf24l01 v1.4, da utilizzare con compilatore XC8 Microchip. La libreria fornisce funzioni specifiche per ogni registro e l'inizializzazione del modulo in modalità trasmittente oppure ricevente.

Descrizione

La libreria nrf24l01 è composta da due file chiamati nrf24l01.c e nrf24l01.h, che contengono tutti i comandi e riferimenti per l'inizializzazione del modulo in modalità trasmittente oppure ricevente. Nel file nrf24l01.c sono contenute le due funzioni principali, `void NRF24L01_Init_TX (void)` e `void NRF24L01_Init_RX (void)`, che impostano il modulo rispettivamente in modo TX e RX, mentre nel file nrf24l01.h sono presenti le impostazioni per le linee CE - CSN, per il numero dei byte da inviare e ricevere, ed infine gli indirizzi di memoria associati ai diversi registri del modulo. In questo modo risulta più facile la sua gestione; infatti, per impostare il modulo in trasmissione basterà richiamare nel main del programma la relativa funzione corrispondente, ovvero `void NRF24L01_Init_TX ()`. La stessa cosa andrà fatta per impostare il modulo in ricezione, richiamando la funzione `void NRF24L01_Init_RX ()`. L'inizializzazione dei moduli è uguale per tutti e due, tranne per la gestione della modalità di lavoro (TX o RX), ma questo verrà fatto in ultimo, impostando un particolare bit (PRIM_TX) del registro CONFIG.

Il fatto che sia stata abilitata la modalità TX o RX, impostando a 0 oppure ad 1 il bit PRIM_RX del registro CONFIG, non è in realtà sufficiente per permettere al modulo di trasmettere o ricevere dati. Affinché questo avvenga, si deve impostare anche il pin CE alto o basso, secondo lo schema della Tabella 2:

	Modo	PRIM_RX	CE
1	RX	1	1
2	TX	0	1
3	TX	0	Minimo 10µs di impulso alto

Tabella 2: *Impostazione della modalità TX e RX.*

La libreria nrf24l01.c prevede la prima e la terza impostazione. La terza è rispettata nella relativa funzione RX, mentre la prima imposta inizialmente a 0 il pin CE nella relativa funzione TX. Ciò significa che, in fase di scrittura del main, sarà compito dell'utente impostare ad 1 il pin CE, aspettare almeno 10 us (1 ms) affinché la trasmissione sia terminata e poi azzerarlo nuovamente (si vedano gli esempi allegati).

La libreria prevede solo l'utilizzo del modo *Enhanced Shockburts*, con Auto ACK abilitato e con un *payload* fisso, (numero dei byte da inviare o ricevere, numero può essere impostato nel file nrf24l01.h per un massimo di 32). La comunicazione tra i due moduli è gestito da eventi interrupt tramite la linea data pipe 0. Per ottenere questo, si dovrà scrivere su alcuni registri presenti all'interno di essi, utilizzando le linee SPI, il comando 0x20 (W_REGISTER) ed il comando WriteSPI. I nomi di questi registri (mnemonic) sono definiti nella mappa del registro contenuta nel file nrf24l01.h e ad ognuno di essi è

associato un indirizzo di memoria esadecimale. Un riassunto dei registri interessati, con le impostazioni previste per questa libreria, è mostrato in Tabella 3.

	Registro	Descrizione
1	CONFIG	Abilita PWR_UP, EN_CRC e gli interrupt sul registro STATUS
2	EN_AA	Abilita Enhanced Shockburts e Auto ACK
3	EN_RXADDR	Abilita data pipe 0
4	SETUP_AW	Imposta 5 byte come larghezza dell'indirizzo di scrittura - lettura
5	SETUP_RETR	Disabilita ritrasmissione dati (n° ritrasmissioni = 0)
6	RF_CH	Imposta il canale 2 per trasmissione - ricezione
7	RF_SETUP	Imposta velocità 2Mps, 0 db
8	TX_ADDR	Imposta 5 byte per l'indirizzo di scrittura
9	RX_ADDR_P0	Imposta 5 byte per l'indirizzo di lettura del data pipe 0
10	RX_PW_P0	Imposta il payload, da 1 a 32, in RX nel data pipe 0 (nrf24l01.h)
11	STATUS	Imposta data pipe 0 e gestisce l'evento interrupt
12	CONFIG	Imposta TX (PRIM_RX = 0); imposta RX (PRIM_RX = 1)

Tabella 3: *Registri principali utilizzati nella libreria.*

Come si può notare, l'unica differenza tra la funzione TX e quella RX risiede nella impostazione finale del bit PRIM_RX nel registro CONFIG. In effetti, questo risulta essere il registro più importante perché, oltre a gestire la trasmissione o la ricezione dati, è quello che abilita anche gli interrupt necessari ai due moduli per gestire la ricezione e trasmissione dei dati.

Prima di approfondire, è necessario chiarire alcuni concetti riguardo questi eventi interrupt. Riassumendo brevemente quanto scritto nel datasheet, impostando entrambi i moduli in modalità *Enhanced Shockburst* con interrupt abilitati, tutto ciò che segue avviene in automatico: il modulo TX trasmette il pacchetto dati al modulo RX quindi si imposta immediatamente in ricezione, in attesa di ricevere un pacchetto ACK di convalida entro un brevissimo lasso di tempo di 130uS. Viceversa, il modulo RX, una volta ricevuto e convalidato il pacchetto dati, si imposta momentaneamente in trasmissione ed invia il pacchetto ACK di convalida al modulo TX. In caso di errore, ovvero se l'ACK per qualche motivo non viene ricevuto, il modulo TX provvede a ritrasmettere tutto il pacchetto completo per un numero di ritrasmissioni impostato nel registro SETUP_RETR. Tutto questo processo è gestito da alcuni interrupt interni al modulo stesso. Infatti, si deve tenere presente che il modulo NRF24L01 ha un pin attivo di interrupt a basso livello, IRQ, composto da tre sorgenti chiamati MAX_RT, TX_DS e RX_DR, presenti nel registro STATUS. La selezione di queste sorgenti viene effettuata con i bit 4, 5 e 6 nel registro CONFIG. Per entrambi questi due registri, i bit 4 e 5 riguardano l'interrupt in trasmissione, il bit 6 l'interrupt in ricezione. I tre bit presenti nel registro CONFIG sono a tutti gli effetti delle bitmask che, se impostati a 0, abilitano l'interrupt sul sorgente selezionato nel registro STATUS, se impostati ad 1 lo disabilitano. Affinché venga generato l'evento interrupt, è necessario portare ad 1 (ovvero ripulire) tutti

questi bit presenti nel registro STATUS.

Nel caso del modulo TX, l'interrupt imposta a 0 il bit 4 (MAX_RT) quando viene raggiunto il valore massimo di ritrasmissioni del pacchetto dati impostato nel registro SETUP_RETR e il bit 5 (TX_DS), quando riceve il pacchetto ACK vuoto dal modulo ricevente. Nel caso non fosse vuoto verrebbe azzerato anche il bit 6 (RX_DR).

Nel caso del modulo RX, l'interrupt imposta a 0 solo il bit 6 (RX_DR) non appena il pacchetto dati viene ricevuto. Questo in teoria perché, sperimentalmente, ho constatato che questa affermazione non è vera e lo spiegherò in seguito quando descriverò una funzione particolare di controllo relativa al modulo ricevente.

Da ciò si evince che per ripulire questi bit è necessario un bitmask, valido per entrambi, uguale a 0x70.

Si tenga inoltre presente che, in stato di Reset (default), molti di questi registri hanno già dei bit abilitati. Questi bit sono riscritti dalla libreria in fase di inizializzazione del modulo. Infatti, esiste una differenza tra il comando W_REGISTER e la scrittura vera e propria: W_REGISTER predispone il registro per la scrittura mentre il comando WriteSPI riscrive l'intero registro.

Bit da impostare su ogni registro del modulo in fase di inizializzazione

CONFIG:

bit 0	(PRIM_RX)	Impostato a 0 va in TX, impostato ad 1 va in RX.
bit 1	(PWR_UP)	Impostato ad 1 provvede ad alimentare il modulo.
bit 2	(CRCO)	Impostato a 0 invia un byte di codifica.
bit 3	(EN_CRC)	Impostato ad 1 abilita l'invio del CRC.
bit 4	(MAX_RT)	Impostato a 0 abilita l'interrupt nel sorgente del registro STATUS.
bit 5	(TX_DS)	Impostato a 0 abilita l'interrupt nel sorgente del registro STATUS.
bit 6	(RX_DR)	Impostato a 0 abilita l'interrupt nel sorgente del registro STATUS.

Comando da inviare: 0x0A

EN_AA:

bit 0	(ENAA_P0)	Impostato ad 1 abilita *Enhanced Shockburts* e l'Auto ACK del data pipe 0.

Comando da inviare: 0x01

EN_RXADDR:

bit 0	(ERX_P0)	Impostato ad 1 abilita il data pipe 0.

Comando da inviare: 0x01

SETUP_AW:

bit 0 (AW) Entrambi impostati ad 1 abilitano 5 byte (predefiniti) come
bit 1 (AW) larghezza dell'indirizzo di scrittura e lettura.

Comando da inviare: 0x03

SETUP_RETR:

bit 0 (ARC) Entrambi impostati ad 0 disabilitano la ritrasmissione dati.
bit 1 (ARC)

Comando da inviare: 0x00

RF_CH:

bit 0 (RF_CH) Impostati rispettivamente a 0 e 1 selezionano il canale 2 (predefinito)
bit 1 (RF_CH) frequenza di lavoro = 2400 + RF_CH = 2400 + 2 = 2402 Mhz.

Comando da inviare: 0x02

RF_SETUP:

bit 1 (RF_PWR)
bit 2 (RF_PWR) bit 1 e 2, impostati entrambi ad 1 selezionano 0db
bit 3 (RF_DR_HIGH) bit 3, impostato ad 1 seleziona la velocità di 2 Mbps

Comando da inviare: 0x0E

TX_ADDR:

Indirizzo di scrittura predefinito. Questo indirizzo può essere stabilito in modo arbitrario ma deve essere uguale a quello impostato nel registro RX_ADD_P0

Comando da inviare: 0xE7 0xE7 0xE7 0xE7 0xE7

RX_ADDR_P0:

Indirizzo di lettura predefinito. Questo indirizzo può essere stabilito in modo arbitrario ma deve essere uguale a quello impostato nel registro TX_ADDR.

Comando da inviare: 0xE7 0xE7 0xE7 0xE7 0xE7

RX_PW_P0:

Imposta il numero dei byte da trasmettere o ricevere (*payload*) nel data pipe 0. Fa parte di un pacchetto dati, formattato come descritto nel datasheet. Tale valore è definito nel file nrf24l01.h

STATUS:

bit 1 (RX_P_N0) I bit 1, 2 e 3 impostati a 0 selezionano il data pipe 0.
bit 2 (RX_P_N0)
bit 3 (RX_P_N0)
bit 4 (MAX_RT) I bit 4, 5 e 6 impostati ad 1 ripuliscono le sorgenti di interrupt.

bit 5 (TX_DS)
bit 6 (RX_DR)

Comando da inviare: 0x70

Scrittura di un registro del modulo

La linea guida da seguire per scrivere sui registri interessati, preferibilmente seguendo l'ordine crescente di ogni indirizzo di memoria corrispondente, è la seguente:

1. Abilitare la scrittura sul registro scelto portando la linea CSN a 0.
2. Inviare il comando 0x20 (W_REGISTER).
3. Effettuare un OR con il suo indirizzo di memoria corrispondente.
4. Scrivere sui diversi bit del registro scelto, come suggerito dal datasheet.
5. Disabilitare la scrittura sul registro scelto portando la linea CSN a 1.

Lettura di un registro del modulo

La linea guida da seguire per leggere sui registri interessati, è la seguente:

1. Abilitare la lettura del registro scelto portando la linea CSN a 0.
2. Inviare il comando 0x00 (R_REGISTER).
3. Effettuare un OR con il suo indirizzo di memoria corrispondente.
4. Leggere i diversi bit del registro scelto, come suggerito dal datasheet.
5. Disabilitare la lettura del registro scelto portando la linea CSN a 1.

Scrittura dei registri della libreria

In questo paragrafo è descritta la procedura da seguire per la scrittura di ogni registro previsto nella libreria, utilizzando le funzioni Microchip e la libreria delay LaurTec.

La prima cosa da fare, prima di scrivere, è impostare le linee CE - CSN in uscita ed impostare CSN a 1 prima di scrivere nei diversi registri.

```
void Setup_CE_CSN (void)
{
     // Imposto le linee CE e CSN in uscita
     NRF24L01_CE = 0;
     NRF24L01_TRIS_CE = 0;

     // Disabilito la scrittura sui registri del modulo
     NRF24L01_CSN = 1;
     NRF24L01_TRIS_CSN = 0;
     delay_ms (100);
}
```

Si inizia quindi a scrivere sul primo registro (CONFIG) e lo si imposta come spiegato in precedenza, ovvero abilitando solo PWR_UP e EN_CRC, lasciando inalterati gli altri parametri.

```
void Enable_PWR_UP_CRC (void)
{
     // Abilito la scrittura del registro
     NRF24L01_CSN = 0;

     //Inviando il comando 0x20 (W_REGISTER) e facendo un OR con l'
     //indirizzo del
     //suo mnemonic (registro) associato, imposto  ad 1 il bit 5 dello
     //stesso indirizzo,
     //segnalando al modulo che devo scrivere sul registro corrispondente
     //(vedere mappa del registro nel file nRF24L01.h )
     WriteSPI (W_REGISTER | CONFIG);

     //Scrivo 0x0A  ed imposto ad 1 i bit 1 e 3 di questo registro,
     //ovvero abilito PWR_UP
     //e EN_CRC.  Il bit 0, PRIM_RX, è quello che stabilisce la modalità
     //di lavoro del
     //modulo: se è uguale a 0 il modulo andrà in trasmissione (PTX), se
     //è uguale ad  1
     //andrà in ricezione (PRX). Per una migliore gestione del sistema,
     //inizialmente
     //imposto entrambi i moduli in trasmissione

     //Abilito PWR_UP e EN_CRC
     WriteSPI (0x0A);

     //Disabilito la scrittura del registro
     NRF24L01_CSN = 1;
     delay_ms (10);
}
```

Dando per scontato che si sia capito il meccanismo di lavoro della linea CSN per abilitare o meno la scrittura, si deve abilitare l'auto ACK del solo data pipe 0 nel registro EN_AA, impostando allo stesso tempo la modalità *Enhanced Shockburts*.

```
void Enable_AUTO_ACK (void)
{
     NRF24L01_CSN = 0;
     WriteSPI (W_REGISTER | EN_AA);

     //Imposto ad 1 il bit 0 di ENAA_P0  ed abilito ESB e l'Auto ACK del
     //data pipe 0
     WriteSPI (0x01);
     NRF24L01_CSN = 1;
}
```

Nel registro EN_RXADDR si abilita il data pipe 0, senza ancora selezionarlo. Questo deve essere fatto all'ultimo nel registro STATUS.

```
void Enable_DATAPIPE (void)
{
     NRF24L01_CSN = 0;
     WriteSPI (W_REGISTER | EN_RXADDR);

     //Imposto ad 1 il bit 0 di ERX_P0 ed abilito il data pipe 0
     WriteSPI (0x01);
     NRF24L01_CSN = 1;
}
```

Nel registro SETUP_AW si deve imposta la larghezza dell'indirizzo a 5 byte.

```
void Setup_ADDRESS_WIDTH (void)
{
     NRF24L01_CSN = 0;
     WriteSPI (W_REGISTER | SETUP_AW);

     //Imposto ad 1 i bit 0 e 1 di AW ed imposto la larghezza
     //dell'indirizzo a 5 byte
     WriteSPI (0x03);
     NRF24L01_CSN = 1;
}
```

Nel registro SETUP_RETR si disabilita la ritrasmissione del pacchetto dati (ciò comporta che alla prima trasmissione, convalidata o meno da ESB, viene generato un evento di interrupt nel bit 4 (MAX_RT) del registro STATUS.).

```
void Disable_RETRANSMISSION (void)
{
     NRF24L01_CSN = 0;
     WriteSPI (W_REGISTER | SETUP_RETR);

     //Imposto ad 0 i bit 0 e 1 di ARC e disabilito la ritrasmissione dei
     //dati
     WriteSPI (0x00);
     NRF24L01_CSN = 1;
}
```

Nel registro RF_CH si imposta il canale 2.

```
void Setup_CHANNEL (void)
{
     NRF24L01_CSN = 0;
     WriteSPI (W_REGISTER | RF_CH);

     //Imposto rispettivamente a 0 e 1  i bit 0 e 1 di RF_CH ed imposto
     //il canale 2.
     //La frequenza del canale RF sarà uguale a 2402 MHz, secondo la
     //formula F = (2400 + RF_CH) MHz
     WriteSPI (0x02);
     NRF24L01_CSN = 1;
}
```

Nel registro RF_SETUP si imposta la potenza e la velocità di trasmissione.

```
void Setup_MPS_DB (void)
{
     NRF24L01_CSN = 0;
     WriteSPI (W_REGISTER | RF_SETUP);

     //Imposto ad 1 i bit 1, 2 di RF_PWR (0db) e ad 1 il bit 3 di
     //RF_DR_HIGH (2Mbps)
     // Imposto così i decibel (potenza) e la velocità di trasmissione (0
     //db, 2Mbps)
     WriteSPI (0x0E);
     NRF24L01_CSN = 1;
}
```

Nel registro TX_ADDR si imposta l'indirizzo di scrittura per il data pipe 0.

```
void Setup_WRITING_ADDRESS (void)
{
     NRF24L01_CSN = 0;
     WriteSPI (W_REGISTER | TX_ADDR);

     // Imposto l'indirizzo di scrittura del data pipe 0
     WriteSPI (0xE7);
     WriteSPI (0xE7);
     WriteSPI( 0xE7);
     WriteSPI (0xE7);
     WriteSPI( 0xE7);
     NRF24L01_CSN = 1;
     NRF24L01_CE = 1;
     delay_ms ( 1);
     NRF24L01_CE = 0;
}
```

Nel registro RX_ADDR_P0 si imposta l'indirizzo di lettura per il data pipe 0.

```
void Setup_READING_ADDRESS (void)
{
     NRF24L01_CSN = 0;
     WriteSPI(W_REGISTER | RX_ADDR_P0);

     //Imposto l'indirizzo di lettura del data pipe 0
     WriteSPI (0xE7);
     WriteSPI (0xE7);
     WriteSPI( 0xE7);
     WriteSPI (0xE7);
     WriteSPI( 0xE7);
     NRF24L01_CSN = 1;
     NRF24L01_CE = 1;
     delay_ms (1);
     NRF24L01_CE = 0;
}
```

Nel registro RX_PW_P0 si imposta il *payload* (numero dei byte da trasmettere o ricevere, definito nel file nrf24l01.h).

```
void Setup_RX_PAYLOAD (void)
{
     NRF24L01_CSN = 0;
     WriteSPI (W_REGISTER | RX_PW_P0);

     //Imposto il numero dei byte in RX payload nel data pipe 0
     WriteSPI (PAYLOAD_BYTES);
     NRF24L01_CSN = 1;
     delay_ms(100);
}
```

Per la pulizia dei buffer TX e RX, necessaria in fase di trasmissione o ricezione, si deve inviare in esecuzione la seguente funzione:

```
void Clear_BUFFER_TX_RX (void)
{
```

```
    //Pulisco il buffer TX
    NRF24L01_CSN = 0;
    WriteSPI (FLUSH_TX);
    NRF24L01_CSN = 1;

    //Pulisco il buffer  RX
    NRF24L01_CSN = 0;
    WriteSPI (FLUSH_RX);
    NRF24L01_CSN = 1;
}
```

Prima di trasmettere o ricevere un pacchetto dati si deve selezionare il data pipe 0 per entrambi i moduli e ripulire, ovvero portare ad 1, i bit interessati all'evento interrupt.

```
void Clear_STATUS_reg (void)
{
    NRF24L01_CSN = 0;
    WriteSPI (W_REGISTER | STATUS);

    // Ripulisco il registro STATUS
    // Per questa operazione utilizzo la bitmask 0x70
    // Lascio impostati a 0  i bit 1, 2 e 3 e seleziono il data pipe 0
    // Imposto ad 1 i bit 4, 5 e 6 e ripulisco gli interrupt interessati
    WriteSPI (0x70);
    NRF24L01_CSN = 1;
}
```

Per impostare il modulo in trasmissione si deve usare questa funzione:

```
void Setup_TX_Mode (void)
{
    NRF24L01_CSN = 0;
    WriteSPI (W_REGISTER | CONFIG);

    //Imposto ad 0 il bit PRIM_RX e la linea CE a 0 (modalità TX)
    WriteSPI (0x0E);
    NRF24L01_CSN = 1;
    NRF24L01_CE = 0;
    delay_ms(100);
}
```

Per impostare il modulo in ricezione si deve usare quest'altra funzione:

```
void Setup_RX_Mode (void)
{
    NRF24L01_CSN = 0;
    WriteSPI (W_REGISTER | CONFIG);

    // Imposto ad 1 il bit PRIM_RX e la linea CE ad 1  (modalità RX)
    WriteSPI (0x0F);
    NRF24L01_CSN = 1;
    NRF24L01_CE = 1;
    delay_ms(100);
}
```

Il listato completo della del file nrf24l01.c può essere scaricato dal sito LaurTec.

Funzione di controllo per ricezione dati

Come si può notare, nel file nrf24l01.c è presente una funzione di lettura degli interrupt generati, relativa al registro STATUS del modulo RX, chiamata `unsigned char NRF24L01_DataReady (void)`. Questa funzione controlla che, nella fase transitoria tra una ricezione e l'altra del pacchetto dati, il bit 6 (RX_DR) sia posto ad 1, ovvero ripulito, in modo da poter generare continuamente l'evento interrupt. Ma qui mi trovo in disaccordo su quando scritto nel datasheet, ovvero che l'interrupt sul modulo RX riguarda solo il bit RX_DR e credo di poterlo, in un certo senso, dimostrare. Infatti, volendo verificare lo stato del solo bit 6, ignorando i bit 4 e 5, avrei dovuto scrivere più correttamente: `if ((data & 0x40) == 0x40))`. Se "data" fosse stato uguale a 0x50, 0x60 o 0x70, in ogni caso con il bit 6 posto ad 1, non sarebbe cambiato nulla come risultato finale ai fini della verifica. Invece, scrivendo erroneamente l'uguaglianza if (data == 0x40) ed allo stesso tempo constatando che il sistema di comunicazione tra i due moduli funziona egregiamente, mi sono reso conto che anche i bit 4 e 5 vengono intercettati dall'interrupt ed azzerati. Se così non fosse, con valori di "data" diversi da 0x40, si verificherebbe un errore return 1. L'azzeramento del bit 4 può essere giustificato dal fatto che la ritrasmissione dati è disabilitata e quindi viene generato immediatamente l'interrupt, ma trovo inspiegabile al momento giustificare l'azzeramento del bit 5, poiché nessun ACK viene ricevuto dal modulo RX. Tuttavia, considerando il fatto che la transazione avviene in modo regolare, ho deciso di non apportare modifiche a questa funzione.

Scrittura della libreria nrf24l01.h

All'interno di questo file sono definiti, oltre alla mappa del registro, anche le linee CE - CSN utilizzate, ed il *payload*. Queste linee sono arbitrarie, il che significa che possono essere scelti eventualmente altri pin liberi del microcontrollore per la loro gestione, a patto che non siano quelle delle linee SPI. Il `#define PAYLOAD_BYTES` definisce il numero dei byte da inviare o ricevere tra i moduli e deve essere compreso tra 1 e 32. Questo significa che l'utente deve prevedere in anticipo di quanti byte necessita per il suo Firmware. Per fare un esempio, nel caso si voglia trasmettere e ricevere un valore dal convertitore ADC a 10 bit, il cui valore massimo è uguale a 1023, necessito di due byte (uno alto ed uno basso). Entrambi i file nrf24l01.h (TX e RX) devono essere modificati scrivendo il numero 2 in corrispondenza del `#define PAYLOAD_BYTES`.

Listato completo del file nrf24l01.h (TX – RX)

```
#ifndef   NRF24L01_H
#define  NRF24L01_H

//*********************************************************************
//  DEFINE
//*********************************************************************
#define NRF24L01_CE          LATEbits.LATE0
#define NRF24L01_TRIS_CE     TRISEbits.TRISE0

#define NRF24L01_CSN         LATEbits.LATE2
#define NRF24L01_TRIS_CSN    TRISEbits.TRISE2

// Numero di byte da inviare o leggere
#define PAYLOAD_BYTES        1
```

```
//*****************************************************************************
//  MAPPA DEL REGISTRO
//*****************************************************************************
//        Mnemonic                value
#define   R_REGISTER              0x00
#define   W_REGISTER              0x20
#define   R_RX_PAYLOAD            0x61
#define   W_TX_PAYLOAD            0xA0
#define   FLUSH_TX                0xE1
#define   FLUSH_RX                0xE2
#define   REUSE_TX_PL             0xE3

#define   CONFIG                  0x00
#define   EN_AA                   0x01
#define   EN_RXADDR               0x02
#define   SETUP_AW                0x03
#define   SETUP_RETR              0x04
#define   RF_CH                   0x05
#define   RF_SETUP                0x06
#define   STATUS                  0x07
#define   OBSERVE_TX              0x08
#define   CD                      0x09
#define   RX_ADDR_P0              0x0A
#define   RX_ADDR_P1              0x0B
#define   RX_ADDR_P2              0x0C
#define   RX_ADDR_P3              0x0D
#define   RX_ADDR_P4              0x0E
#define   RX_ADDR_P5              0x0F
#define   TX_ADDR                 0x10
#define   RX_PW_P0                0x11
#define   RX_PW_P1                0x12
#define   RX_PW_P2                0x13
#define   RX_PW_P3                0x14
#define   RX_PW_P4                0x15
#define   RX_PW_P5                0x16
#define   FIFO_STATUS             0x17

//*****************************************************************************
//  PROTOTIPI DI FUNZIONE
//*****************************************************************************
void Setup_CE_CSN (void);
void Enable_PWR_UP_CRC (void);
void Enable_AUTO_ACK (void);
void Enable_DATAPIPE (void);
void Setup_ADDRESS_WIDTH (void);
void Disable_RETRANSMISSION (void);
void Setup_CHANNEL (void);
void Setup_MPS_DB (void);
void Setup_WRITING_ADDRESS (void);
void Setup_READING_ADDRESS (void);
void Setup_RX_PAYLOAD (void);
void Clear_BUFFER_TX_RX (void);
void Clear_STATUS_reg (void);
void Setup_TX_Mode (void);
void Setup_RX_Mode (void);
unsigned char NRF24L01_DataReady (void);
void NRF24L01_Init_TX (void);
void NRF24L01_Init_RX (void);
```

Prova di comunicazione tra due moduli

Hardware necessario

Sono necessari due microcontrollori, alimentati separatamente e corredati di tutti i componenti elettronici che garantiscano il loro funzionamento. Essi devono essere programmati con i relativi file HEX generati dalla compilazione. I test sono stati fatti facendo uso di due schede Freedom II, in quando già complete del tutto. Inoltre servono due moduli da collegare ciascuno ad una scheda, secondo il seguente schema:

- RE0 = CE — si può usare qualunque altro pin
- RE2 = CSN — si può usare qualunque altro pin
- RB0 = SDI (MISO)
- RB1 = CLK
- RC7 = SDO (MOSI)
- RA5 = SS — non collegato

Nel caso si utilizzino microcontrollori diversi questi collegamenti andranno modificati secondo le indicazioni descritte nel relativo datasheet.

Qualche chiarimento sugli esempi di programmi

Gli esempi allegati a questo tutorial sono stati compilati con MPLABX v2.05 e XC8 v1.21, utilizzando le librerie Microchip, mentre il microcontrollore scelto per la comunicazione è il PI18F4550. Esistono però due versioni degli stessi esempi compilati con la libreria LaurTec LTlib v4.0.4. Queste librerie possono essere usate in modo indipendente dalle librerie Microchip. Il grande vantaggio sta nella versatilità ovvero, scegliendo un'altra MCU prevista nella libreria LaurTec, basta cambiare, in fase di creazione del programma, solo il nome del microcontrollore ed esso verrà compilato ed eseguito correttamente dal nuovo Hardware. Ovvio che, essendo questa una libreria proprietaria, è stato necessario modificare la sintassi dei comandi di scrittura e lettura SPI nel file nrf24l01.c originale. Tutto il resto resta invariato. Ogni esempio microchip è suddiviso in due cartelle chiamate TX e RX, all'interno delle quali sono presenti i relativi sorgenti, il file delay.c, modificato nel valore della frequenza del quarzo per lavorare ad 8 MHz con oscillatore interno, il file delay.h, i file nrf24l01.c e nrf24l01.h ed il file di configurazione dei bit PIC18F4550_config. Anche i due esempi LaurTec sono suddivisi in due cartelle chiamate TX e RX, all'interno delle quali sono presenti i relativi sorgenti, il file LTlib_delay.c, modificato nel valore della frequenza del quarzo per lavorare ad 8 MHz con oscillatore interno, il file LTlib_delay.h, ed i file nrf24l01.c e nrf24l01.h.

Nel caso si usi il PIC18F46K22 non è necessario modificare il valore della frequenza del quarzo nei file delay.c e LTlib_delay.c.

Esempio di programma

Come esempio, ho scelto quello della trasmissione e della ricezione di un valore adc compreso tra 0 e 1023. Questo esempio è stato compilato sia usando il compilatore XC8 della Microchip che la libreria Laurtec LTlib 4.0.4. Mentre per il primo (Lettura_adc_con_display_lcd_3) non credo ci sia bisogno di dare chiarimenti, per il secondo (Lettura_adc_con_display_lcd_3_LTlib) sarà necessario un ulteriore approfondimento da parte dell'utente, cosa che esula dallo scopo di questo tutorial. In ogni caso, qualunque libreria si usi, si dovrà:

- Impostare il microcontrollore affinchè utilizzi il suo clock interno a 8 Mhz, in quanto i moduli lavorano in modo ottimale con questa frequenza.
- Impostare 2 byte in corrispondenza del #define PAYLOAD_BYTES.

L'hardware necessario si risolve con due schede Freedom II dotate di display LCD, (questo perché sia il dato trasmesso che quello ricevuto devono essere entrambi visualizzati), un potenziometro da 2,2K collegato al pin RA1 (AN1) della scheda freedom trasmittente e, chiaramente, i due moduli collegati come descritto in precedenza. Prenderò in considerazione solo la parte relativa alla trasmissione, al fine di confrontare i due firmware, eliminando alcuni commenti presenti nei programmi originali.

Lettura_adc_con_display_lcd_3 (TX)

```
//************************************************************
// INCLUDE
//************************************************************
#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include "PIC18F4550_config.h"
#include <spi.h>
#include "nrf24l01.h"
#include "LCD_44780.h"
#include "LCD_44780.c"
#include "delay.h"

//************************************************************
// DEFINE
//************************************************************
#define LINE_SPI_SCK   LATBbits.LATB1
#define LINE_SPI_SDO   LATCbits.LATC7
#define LINE_SPI_SDI   LATBbits.LATB0
#define LINE_SPI_SS    LATAbits.LATA5    // non usata

#define LCD_DEFAULT
#define NUMERO_CAMPIONI 8

//************************************************************
// PROGRAMMA PRINCIPALE
//************************************************************
int main(void)
{
     // Variabile utilizzata per il numero delle letture
     unsigned char i;
```

```
// Variabile per salvare la sommatoria dei dati letti
unsigned int sommatoria = 0;

// Variabile per salvare il valore della conversione
unsigned int lettura = 0;

// Array per i dati da trasmettere
unsigned char bufferTX[32];

// Imposto PORTA
// RA5 impostato in uscita per linea SS
LATA = 0x00;
TRISA = 0b11011111;

// Imposto PORTB
// RB0 impostato in ingresso per linea SDI (MISO)
//RB1 impostato in uscita per linea SCK (CLOCK)
LATB = 0x00;
TRISB = 0b11111101;

// Imposto PORTC
// RC7 impostato in uscita per linea (MOSI)
LATC = 0x00;
TRISC = 0b01111111;

// Imposto PORTD
// Tutta in uscita per pilotaggio display lcd
LATD = 0x00;
TRISD = 0x00;

// Imposto PORTE
// RE0 impostato in uscita per linea CE
// RE2 impostato in uscita per linea CSN
LATE = 0x00;
TRISE = 0x00;

// Imposto LCD
LCD_initialize (20);

// Abilito A0-A1 come ingressi analogici
// VREF sono impostate a massa e VCC
ADCON1 = 0b00001101;

// Seleziono AN1 come ingresso
ADCON0 = 0b00000100;

// Imposto i tempi di conversione e giustificazione a destra
// TAD : FOSC/16
// TACQ: 16 TAD
// Giustificazione destra
ADCON2 = 0b10110101;

// Abilito l'ADC
ADCON0 |= 0b00000001;

//attesa prima di iniziare la prima conversione
//basterebbero 2us
delay_ms (1);

// Imposto oscillatore interno 8 MHz
OSCCON = 0b01110010;
```

```
    // Inizializzo le linee per l'SPI
    LINE_SPI_SCK = 0x00;
    LINE_SPI_SDO = 0x00;
    LINE_SPI_SDI = 0x01;

    // Chiudo l'SPI per evitare eventuali errori)
    CloseSPI() ;

    //Inizializzazione del modulo SPI
    //Baudrate Fosc/16 @20MHz -> 1250000 KHz
    OpenSPI(SPI_FOSC_16,MODE_00,SMPEND);

    // Inizializzo il modulo NFR24L01
    NRF24L01_Init_TX();

while(1)
    {
          sommatoria = 0;

          // Effettuo 8 letture per fare una media
          for (i =0; i<NUMERO_CAMPIONI; i++)
          {
                // Avvio la conversione Analogico/Digitale
                ADCON0bits.GO = 1;

                // Attendo la fine della conversione
                while(ADCON0bits.GO)
                      ;
                // Prelevo il valore della conversione
                lettura = (((unsigned int) ADRESH) << 8) | ADRESL;

                // Sommatoria delle letture fatte
                sommatoria = sommatoria + lettura;
          }

          // Calcolo della media
          sommatoria = sommatoria / NUMERO_CAMPIONI;

          LCD_clear ();
          LCD_write_message ("ADC = ");

          // Visualizzo sul display il dato che trasmetto
          LCD_goto_xy (8,1);
          LCD_write_integer (sommatoria, 4, ZERO_CLEANING_ON);

          delay_ms(50);

          // CASTING da INT a CHAR per i valori low e high contenuti nei
          // registri ADRESL e ADRESH
          // byte meno significativo
          bufferTX[0] = (unsigned char) ADRESL;
          // byte più significativo
          bufferTX[1] = (unsigned char) ADRESH;

          // Pulisco i buffer TX e RX
          Clear_BUFFER_TX_RX ();

          // Ripulisco il registro STATUS
          Clear_STATUS_reg ();

          // Scrivo il buffer TX
          NRF24L01_CSN = 0;
```

```
            WriteSPI(W_TX_PAYLOAD);
            WriteSPI (bufferTX[0]);
            WriteSPI (bufferTX[1]);
            NRF24L01_CSN = 1;
            NRF24L01_CE = 1;
            delay_ms(1);
            NRF24L01_CE = 0;
            delay_ms(100);
      }
}
```

Lettura_adc_con_display_lcd_3_LTlib (TX)

```
//************************************
// INCLUDE
//************************************
#include <xc.h>
#include <LTlib.h>
#include "LTlib_delay.h"
#include "LTlib_delay.c"
#include <module_IO.h>
#include <module_IO.c>
#include <module_ADC.h>
#include <module_ADC.c>
#include <module_SPI.h>
#include <module_SPI.c>
#include <LCD_44780.h>
#include <LCD_44780.c>
#include "nrf24l01.h"

//************************************
// PROGRAMMA PRINCIPALE TX
//************************************
int main(void)
{
      // Array per memorizzare i dati da trasmettere
      unsigned char bufferTX[32];

      // Imposto tutte le porte come ingressi - I/O
      IO_set_all_ports_as_inputs();

      // Imposto tutta la PORTE come uscita per i segnali CE e CSN - I/O
      IO_set_port_direction(IO_PORTE, IO_ALL_PORT_OUTPUT);

      // Inizializzo il display LCD
      LCD_initialize(20);

      // Imposto tutte le porte analogiche come digitali - I/O
      ADCON1 = 0x0F;

      // Imposto oscillatore interno 8 MHz
      OSCCON = 0b01110010;

      // Apro l'SPI
      SPI1_baudrate(SPI_CLK_OSC_16);
      SPI1_mode (SPI_MODE_0);
      SPI1_open (SPI_MASTER_DEVICE);

      // Inizializzo il modulo NFR24L01
```

```
    NRF24L01_Init_TX();

    while (1)
    {
          LCD_home();
          LCD_write_message("ADC: ");
          LCD_write_integer (ADC_read_channel(ADC_CH1), 4,
                             LCD_ZERO_CLEANING_ON);

          delay_ms(50);

          // CASTING da INT a CHAR per i valori low e high contenuti nei
          // registri ADRESL e ADRESH
          bufferTX[0] = (unsigned char) (ADC_BUFFER_LOW);
          bufferTX[1] = (unsigned char) (ADC_BUFFER_HIGH);

          // Pulisco i buffer TX e RX
          Clear_BUFFER_TX_RX();

          // Ripulisco il registro STATUS
          Clear_STATUS_reg();

          // Scrivo il buffer TX
          NRF24L01_CSN = 0;
          SPI1_write_byte (W_TX_PAYLOAD);
          SPI1_write_byte  (bufferTX[0]);
          SPI1_write_byte  (bufferTX[1]);
          NRF24L01_CSN = 1;
          NRF24L01_CE = 1;
          delay_ms(1);
          NRF24L01_CE = 0;
          delay_ms(100);
    }
}
```

La descrizione sintetica di entrambi i programmi si riassumono in poche righe: dopo avere inizializzato le porte del microcontrollore, il display LCD, l'ADC, l'oscillatore interno, le linee SPI ed il modulo, i programmi entrano in esecuzione. Nel ciclo `while` avviene la lettura del valore ADC, mostrato come valore intero sul display, che viene diviso in due byte ed infine inviato in trasmissione. Come si può notare, anche se già fatto in fase di inizializzazione del modulo, prima della trasmissione dei valori ADC vengono ripuliti ulteriormente i buffer TX-RX ed il registro Status. Questo perché ci troviamo appunto in ciclo `while`, dove un programma deve essere eseguito all'infinito ed è quindi necessario eliminare i vecchi dati. Se si confronta il secondo firmware, quello compilato con la libreria LTlib 4.0.4, con il primo si nota che:

- Tutte le porte sono state inizializzate come ingressi con una singola funzione.
- Si è impostata un'intera porta in uscita, in questo caso PORTE, con una singola funzione.
- Non c'è stato bisogno di impostare tutta la PORTD in uscita per pilotare il display LCD, in quanto la funzione `LCD_initialize(20)` lo fa automaticamente
- Nonostante l'impostazione iniziale di tutte le porte analogiche come digitali, la funzione `ADC_read_channel(ADC_CH1)` ha impostato automaticamente RA1 come pin analogico, aprendo il canale AN1 e leggendone i valori contenuti nei registri.
- Il riconoscimento automatico delle linee SPI coinvolte e la loro impostazione in

ingresso o in uscita avviene grazie ai moduli di libreria dedicati, module_SPI.c e module_SPI.h.

Sta quindi all'utente finale decidere, per i suoi programmi, se utilizzare il compilatore Microchip oppure questa nuova libreria LaurTec.

Per coloro che usano mikroC, l'utente *mpeino* del Forum ha scritto un programma di esempio perfettamente funzionante che trasmette e riceve un byte, solo facendo il porting della libreria che ho scritto adattandola alle sue esigenze. Questo programma è allegato a questo tutorial, insieme ai programmi microchip e LaurTec, come base di partenza per tutti.

Conclusione

Essendo la libreria NRF24L01+ molto versatile, si adatta perfettamente al porting per altri microcontrollori di altre marche e per altri ambienti di lavoro. Ad esempio, la utilizzo modificata anche con gli Atmel, facendo comunicare un PI18F4550 con un atmega 2560. Questo verrà descritto in seguito, in un altro lunghissimo Tutorial dedicato e che sto scrivendo. In ogni caso, per qualunque critica, suggerimento, chiarimento segnalazione di errori, resto a vostra disposizione.

Allegati

Insieme a questo tutorial sono allegati, in formato zip, i seguenti documenti:

- Libreria nrf24l01 v1.4
- Esempi di programmi per compilatore e librerie XC8 Microchip.
- Esempi di programmi per compilatore XC8 Microchip e librerie LTlib 4.0.4.
- Esempio di programma per compilatore mikroC – di Massimo Peino.

Ringraziamenti

Ringrazio l'utente **Massimo Peino** per avermi coinvolto nello studio di questo piccolo modulo dalle caratteristiche impressionanti e del quale, in un certo senso, mi sono "innamorato". Spero vivamente che la nostra passione, il nostro comune impegno e lavoro possa servire un domani a qualcuno per le proprie applicazioni.

Indice Alfabetico

Bibliografia

[1] www.LaurTec.it: sito dove poter scaricare la libreria ed esempi mostrati nell'articolo.

History

Data	Versione	Autore	Revisione	Descrizione Cambiamento
05/04/18	1.0	Marcello Pinna	Mauro Laurenti	Versione Originale.