

Πολυτεχνείο Κρήτης

Τμήμα Ηλεκτρονικών Μηχανικών & Μηχανικών Υπολογιστών

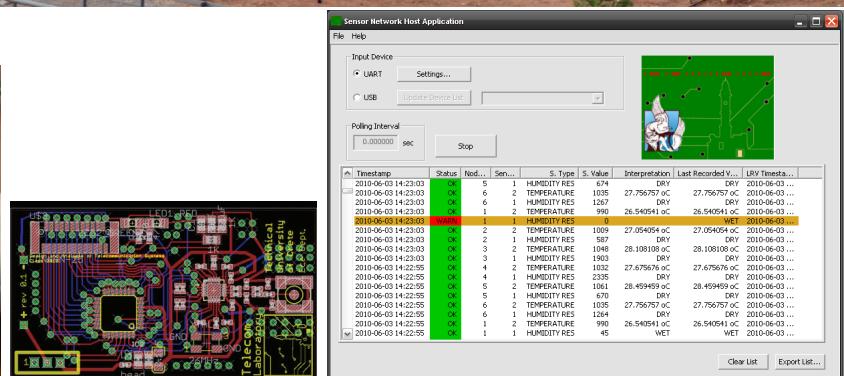
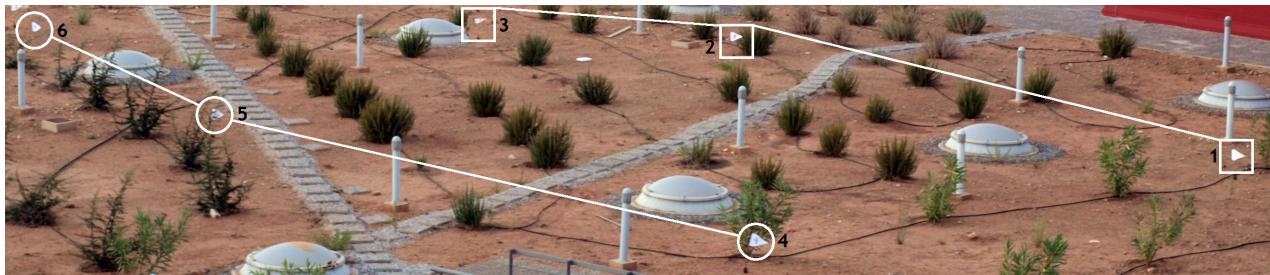
Χειμερινό Εξάμηνο 2010-2011



## ΤΗΛ 412 Ανάλυση και Σχεδίαση (Σύνθεση)

### Τηλεπικοινωνιακών Διατάξεων

7<sup>ου</sup> Εξαμήνου



## Πειραματική Προγραμματιζόμενη Ψηφιακή Ζεύξη

Τηλεπικοινωνιακό Τλικό  
Διδάσκων: Άγγελος Μπλέτσας  
aggelos@telecom.tuc.gr



# 1 Εισαγωγή

## 1.1 Περιγραφή Εργαστηριακής Άσκησης

Στα πλαίσια του εργαστηρίου πρόκειται να πειραματιστούμε με μία ψηφιακή ζεύξη, προγραμματιζόμενη από λογισμικό. Συγκεκριμένα, θα πειραματιστούμε με πομποδέκτες που σχεδιάστηκαν και υλοποιήθηκαν για το μάθημα και βασίζονται σε χαμηλής κατανάλωσης ψηφιακά, ενσωματωμένα ραδιόφωνα, στα 2.4 GHz. Τα ραδιόφωνα αυτά, ελέγχονται από λογισμικό σε γλώσσα C, μέσω του μικροελεγκτή 8051, με τον οποίο έχουμε ήδη πειραματιστεί, στο προηγούμενο εργαστήριο.

Στο εργαστήριο αυτό, θα μάθουμε να προγραμματίζουμε την ψηφιακή ζεύξη. Ας σημειωθεί πως παρόμοιες ψηφιακές ζεύξεις αποτελούν την βάση πληθώρας εφαρμογών, όπως τα εμπορικά δίκτυα αισθητήρων. Πιο συγκεκριμένα, θα μελετήσουμε και θα επεξεργαστούμε τους κώδικες AB\_tx.c και AB\_rx.c. Ο κώδικας AB\_tx υλοποιεί έναν πομπό με συγκεκριμένες παραμέτρους λειτουργίας, ενώ ο κώδικας AB\_rx υλοποιεί έναν αντίστοιχο δέκτη.

Επιπλέον μας δίνονται: τα header files compiler\_defs.h, C8051F320\_defs.h, common.h και CC2500.h που περιέχουν ορισμούς σταθερών παραμέτρων λειτουργίας των πομποδεκτών μας.

## 1.2 Εξοπλισμός

- PC με εγκατεστημένο το πρόγραμμα Silicon Laboratories Integrated Development Environment (IDE). Πρόκειται για το περιβάλλον ανάπτυξης του προγράμματος ελέγχου του μικροελεγκτή. Στο Silicon Laboratories IDE περιλαμβάνονται:
  - source-code editor.
  - source level debugger.
  - in-system flash programmer.
- C8051F320 πλακέτα ανάπτυξης.
- USB Debug Adapter.
- 2 χαμηλού κόστους και συχνότητας πομποδέκτες Chipcon/TI CC2500EMK.
- Silabs ToolStick Base Adapter.
- Silabs ToolStick C8051F320 Daughter Card.
- USB extension cable.
- 1 Battery Holder.
- 2 AA 3Volt batteries.
- Ψηφιακός παλμογράφος.

Με τα παραπάνω, είναι δυνατή η υλοποίηση των προγραμματιζόμενων, ελεγχόμενων από λογισμικό, ενσωματωμένων πομποδεκτών iCubes v0.1, ένας εκ των οποίων παρουσιάζεται στο παρακάτω

σχήμα. Τα iCubes v0.1 ουσιαστικά αποτελούνται από μία πλακέτα μικροελεγκτή Silabs Tool-Stick C8051F320 Daughter Card την οποία συνδέσαμε σε μια πλακέτα ραδιοφώνου Chipcon/TI CC2500EMK. Η σύνδεση έγινε με έξυπνο τρόπο, μέσω ειδικών καλωδίων και η επικοινωνία των δύο παραπάνω διατάξεων υλοποιείται με την βοήθεια του πρωτοκόλλου Serial Peripheral Interface (SPI). Το πρωτόκολλο αυτό θα εξηγηθεί εν συντομίᾳ παρακάτω, και είναι ήδη υλοποιημένο σε εσωτερική διάταξη, τόσο στον μικροελεγκτή, όσο και στο ραδιόφωνο.



Figure 1: Ο πομποδέκτης iCube, v0.1, ο οποίος υλοποιήθηκε για τις ανάγκες του μαθήματος. Παρατηρήστε τον προγραμματιστή μέσω USB, την θήκη των δύο AA μπαταριών και την κεραία, συνδεδεμένη στο ραδιόφωνο.

## 2 Επικοινωνία μεταξύ Μικροελεγκτή και Ραδιοφώνου

Η επικοινωνία του μικροελεγκτή και του ραδιοφώνου γίνεται μέσω SPI interface το οποίο έχει αρχικοποιηθεί σε 4-wire mode. Master device είναι ο μικρολεκτής και slave device είναι το ραδιόφωνο (Σχήμα 2).

Η συγκεκριμένη διεπαφή SPI αποτελείται από τις παρακάτω γραμμές μεταφοράς:

- MISO - (Master Input - Slave Output) που χρησιμοποιείται για τη μεταφορά δεδομένων από τον slave προς τον master. Αντιστοιχεί στο pin SO στην πλευρά του ραδιοφώνου.
- MOSI - (Master Output - Slave Input) που χρησιμοποιείται για τη μεταφορά δεδομένων από τον master προς τον slave. Αντιστοιχεί στο pin SI στην πλευρά του ραδιοφώνου.
- SCK - (System Clock) στην οποία ο master καθορίζει το κοινό ρολόι που θα χρησιμοποιήσουν οι συσκευές για την επικοινωνία. Αντιστοιχεί στο pin SCLK στην πλευρά του ραδιοφώνου.
- NSS - (Slave select) σήμα με διαφορετική λειτουργία ανάλογα με το mode λειτουργίας του SPI. Αντιστοιχεί στο pin CSn στην πλευρά του ραδιοφώνου.

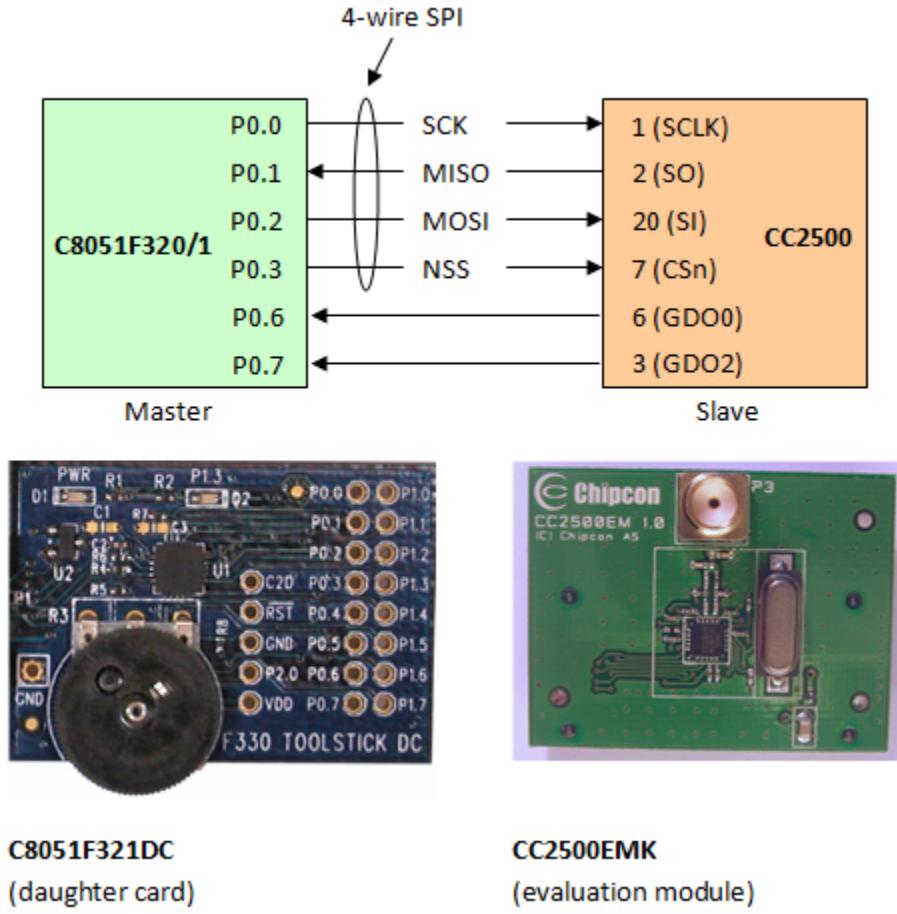


Figure 2: Οι διατάξεις που χρησιμοποιήθηκαν στην υλοποίηση των iCubes. Η διάταξη του μικροελεγκτή επικοινωνεί με το ψηφιακό ραδιόφωνο μέσω 4-wire SPI, όπου ο μικροελεγκτής λειτουργεί ως master και το ραδιόφωνο ως slave.

Παρατηρήστε επίσης, την ύπαρξη δύο ακόμη γραμμών μεταφοράς δεδομένων (General Data Output - GDO) από το ραδιόφωνο προς τον μικροελεγκτή. Οι δύο αυτές γραμμές μεταφέρουν σήματα από το ραδιόφωνο προς τον ελεγκτή, αλλά δεν αποτελούν μέρος της διεπαφής SPI. Επίσης υπάρχουν γραμμές τροφοδοσίας και γείωσης, οι οποίες δεν περιγράφονται στο Σχήμα 2.

## 2.1 Δομή Δεδομένων

Στο πρωτόκολλο επικοινωνίας μεταξύ C8051F320 και CC2500 υπάρχουν 3 τύποι δεδομένων:

- Header Byte: [R/W bit] + [burst access bit] + [6-bit address]  
Περιέχει πληροφορία για ανάγνωση/εγγραφή, πολλαπλή πρόσβαση και τη διεύθυνση του καταχωρητή (του ΡΑΔΙΟΦΩΝΟΥ), τον οποίο θέλουμε να διαβάσουμε/γράψουμε. Αποστέλλεται από τον μικροελεγκτή προς το ραδιόφωνο.
- Status Byte: [Chip\_RDYn bit] + [3-bit STATE] + [4-bit FIFO\_BYTES\_AVAILABLE]  
Πληροφορίες κατάστασης του ραδιοφώνου (οι πιθανές καταστάσεις του ραδιοφώνου περιγρά-

φονται παρακάτω) και των ουρών TX/RX που χρησιμοποιεί ο C8051F320. Αποστέλλεται από το ραδιόφωνο προς τον μικροελεγκτή, ενημερώνοντας για την κατάσταση στην οποία βρίσκεται το ραδιόφωνο. Σημειώνεται πως η ουρές TX\_FIFO, RX\_FIFO έχουν μήκος 64 bytes, και συνεπώς τα 4 bits δεν επαρκούν για να περιγράψουν πλήρως τις ουρές αυτές. Συνεπώς, ο μέγιστος ακέραιος  $2^4 - 1 = 15$  στο status byte, θα πρέπει να ερμηνευτεί ως μια ένδειξη τουλάχιστον 15 bytes (και όχι ακριβώς) στις ουρές αυτές.

- Data Byte: [8-bit data]

Δεδομένα από/προς ουρά (λήψης/αποστολής) του ραδιοφώνου. Αποστέλλονται είτε από τον μικροελεγκτή προς το ραδιόφωνο (π.χ. για εκπομπή, προς TX\_FIFO), είτε από το ραδιόφωνο προς τον μικροελεγκτή (π.χ. για λήψη, από RX\_FIFO).

## 2.2 Οι Καταχωρητές του Ραδιοφώνου

Το ραδιόφωνο CC2500 έχει 3 τύπους καταχωρητών (registers), οι οποίοι είναι προσβάσιμοι μέσω SPI:

1. Configuration Registers (με εύρος διευθύνσεων 0x00-0x2E).

Περιέχουν τις ρυθμίσεις για τη λειτουργία του ραδιοφώνου. Μπορούν να γραφτούν και να διαβαστούν.

2. Status Registers (με εύρος διευθύνσεων 0x30-0x3D, burst bit = 1).

Περιέχουν τις πληροφορίες κατάστασης του ραδιοφώνου. Μπορούν μόνο να διαβαστούν, ένας τη φορά (δεν υπάρχει δυνατότητα για ανάγνωση ριπής, αν και το burst bit παίρνει την τιμή '1' - η τιμή burst bit=0 είναι δεσμευμένη για τα command strobes, τα οποία περιγράφονται παρακάτω).

3. Command Strobe Registers(με εύρος διευθύνσεων 0x30-0x3D, burst bit = 0).

Περιέχουν εντολές που ενεργοποιούν/απενεργοποιούν διάφορες λειτουργίες του ραδιοφώνου. Για να χρησιμοποιηθούν, το burst bit πρέπει να είναι 0 και χρειάζεται να αποσταλεί μόνο ένα header byte από τον μικροελεγκτή C8051F320.

Τέλος, οι registers που αντιστοιχούν στα δεδομένα εκπομπής και λήψης είναι οι δύο ουρές εκπομπής και λήψης (TX\_FIFO/RX\_FIFO). Αντιστοιχούν στη διεύθυνση 0x3F. Για εγγραφή στην TX\_FIFO, πρέπει το R/W bit του header byte να είναι 0, ενώ για ανάγνωση από την RX\_FIFO, το R/W bit πρέπει να είναι 1.

Υπάρχει ακόμα ο καταχωρητής PATABL (0x3E), ο οποίος είναι ειδικός configuration register μήκους 8 bytes και χρησιμοποιείται για τη ρύθμιση της ισχύος εκπομπής. Τα bytes του γράφονται είτε ένα τη φορά, είτε πολλαπλά, χωρίς την χρήση διαδοχικών αποστολών header butes (burst mode). Επιτρέπεται τόσο η εγγραφή, όσο και η ανάγνωσή του.

## 2.3 Καταστάσεις Ραδιοφώνου

Οι καταστάσεις του ραδιοφώνου που καταχωρούνται σε 3 bits στο status byte του ραδιοφώνου, δηλώνουν τα διαφορετικά είδη λειτουργίας (modes) του ραδιοφώνου. Η κατάσταση λειτουργίας του ραδιοφώνου πρέπει να είναι γνωστή στον μικροελεγκτή, ώστε να μπορεί να αποφασίσει ποια

διαδικασία θα εκτελέσει (αποστολή/λήψη πακέτου, ενεργοποίηση „ύπνου“/χαμηλής κατανάλωσης ρεύματος κλπ). Οι καταστάσεις, με την κωδικοποίηση σε 3 bits, ακολουθούν:

- 000-IDLE: Μόνο ο κρύσταλλος και τα ψηφιακά μέρη του ραδιοφώνου είναι ενεργά. Όλα τα υπόλοιπα είναι κλειστά για εξοικονόμηση ενέργειας!
- 001-RX: Δέκτης.
- 010-TX: Πομπός.
- 011-FSTXON: Ετοιμότητα για εκπομπή (έχει σταθεροποιηθεί η παραγωγή κεντρικής συχνότητας από το PLL).
- 100-CALIBRATE: Κατάσταση ρύθμισης (calibration) του PLL.
- 101-SETTLING: PLL υπό ρύθμιση.
- 110-RXFIFO\_OVERFLOW: Υπερχείλιση της ουράς λήψης.
- 111-TXFIFO\_UNDERFLOW: Υποχείλιση στην ουράς εκπομπής.

Θα επανέλθουμε πιο αναλυτικά στο θέμα των καταστάσεων του ραδιοφώνου, κατά την εξέταση του κώδικα.

## 2.4 Συνοπτική Περιγραφή

Σε γενικές γραμμές, ο μικροελεγκτής διαβάζει ή εγγράφει συγκεκριμένους καταχωρητές του ραδιοφώνου. Για να γίνει αυτό, αποστέλλει header byte στο ραδιόφωνο, μέσω της διεπαφής SPI, με την διεύθυνση του καταχωρητή στο ραδιόφωνο που θέλει να διαβάσει ή να εγγράψει. Στην συνέχεια, το ραδιόφωνο απαντά με ένα data byte ή με πλήθος από data bytes, ανάλογα με το αν έχει ζητηθεί πολλαπλή ανάγνωση ριπής (burst mode).

Εξαίρεση στον παραπάνω κανόνα, αποτελεί η αίτηση από τον μικροελεγκτή για ανάγνωση ενός καταχωρητή κατάστασης (status register) ή ενός καταχωρητή command strobe: σ' αυτές τις περιπτώσεις, πολλαπλή ανάγνωση δεν επιτρέπεται/ορίζεται. Στις δυο αυτές περιπτώσεις καταχωρητών, που αφορούν την ίδια περιοχή μνήμης του ραδιοφώνου, το burst bit χρησιμοποιείται μόνο για να διαχωρίσει μεταξύ command strobes και status registers, δηλ. αν και χρησιμοποιείται, δεν υπονοεί λειτουργία ριπής (burst mode).

Σε φυσικό επίπεδο, η επικοινωνία αρχικοποιείται από τον ‚αφέντη‘ (master) μικροελεγκτή, ο οποίος θέτει την γραμμή Channel Select (CSn) (ή αλλιώς NSS από την μεριά του ραδιοφώνου) σε λογική τιμή ‚0‘. Οποιαδήποτε αλλαγή αυτής της λογικής κατάστασης σε ‚1‘, διακόπτει την επικοινωνία. Στην συνέχεια, ο ‚δούλος‘ (slave) ραδιόφωνο απαντά στην γραμμή MISO, με κατάλληλο σήμα ‚ετοιμότητας“, ενημερώνοντας τον μικροελεγκτή ότι είναι σε θέση να επικοινωνήσει. Στην συνέχεια, η επικοινωνία προχωρά σύμφωνα με τα παραπάνω.

Θα επανέλθουμε πιο αναλυτικά στο θέμα της συνομιλίας μεταξύ ραδιοφώνου και μικροελεγκτή, κατά την εξέταση του κώδικα.

### 3 Αναλυτική Περιγραφή Δοθέντος Λογισμικού

#### 3.1 Κώδικας AB\_tx.c

Ορισμός των header files που περιέχουν απαραίτητες δηλώσεις καταχωρητών και μνήμης (π.χ. Special Function Registers - SFR, xdata, code).

```
#include "compiler_defs.h"
#include "C8051F320_defs.h"           // SFR declarations
#include "common.h"
#include "cc2500.h"
```

Μια δομή δεδομένων (struct) που περιέχει όλους τους καταχωρητές ρυθμίσεων για το CC2500 radio module. Ο τύπος BYTE υποδηλώνει ότι πρόκειται για 8-bit καταχωρητές. Οι καταχωρητές αυτοί πρέπει να αρχικοποιηθούν, από τον μηχανικό της ζεύξης, ανάλογα με την εφαρμογή. Ευτυχώς, υπάρχει γρήγορος τρόπος για την ταυτόχρονη εύρεση όλων των τιμών, ο οποίος γίνεται με συμπληρωματικό λογισμικό (RF Studio), και ο οποίος θα εξηγηθεί παρακάτω.

```
typedef struct S_RF_SETTINGS{
    BYTE FSCTRL1;      // Frequency synthesizer control.
    BYTE FSCTRL0;      // Frequency synthesizer control.
    BYTE FREQ2;         // Frequency control word, high byte.
    BYTE FREQ1;         // Frequency control word, middle byte.
    BYTE FREQ0;         // Frequency control word, low byte.
    BYTE MDMCFG4;       // Modem configuration.
    BYTE MDMCFG3;       // Modem configuration.
    BYTE MDMCFG2;       // Modem configuration.
    BYTE MDMCFG1;       // Modem configuration.
    BYTE MDMCFG0;       // Modem configuration.
    BYTE CHANR;          // Channel number.
    BYTE DEVIATN;       // Modem deviation setting (when FSK mod).
    BYTE FREND1;        // Front end RX configuration.
    BYTE FREND0;        // Front end RX configuration.
    BYTE MCSMO;          // Main Radio Control State Machine config.
    BYTE FOCCFG;         // Frequency Offset Compensation Config.
    BYTE BSCFG;          // Bit synchronization Configuration.
    BYTE AGCCTRL2;       // AGC control.
    BYTE AGCCTRL1;       // AGC control.
    BYTE AGCCTRL0;       // AGC control.
    BYTE FSCAL3;          // Frequency synthesizer calibration.
    BYTE FSCAL2;          // Frequency synthesizer calibration.
    BYTE FSCAL1;          // Frequency synthesizer calibration.
    BYTE FSCAL0;          // Frequency synthesizer calibration.
    BYTE FTEST;           // Frequency synthesizer calibration control
    BYTE TEST2;           // Various test settings.
    BYTE TEST1;           // Various test settings.
```

```

    BYTE TEST0;      // Various test settings.
    BYTE FIFOTHR;   // RXFIFO and TXFIFO thresholds.
    BYTE IOCFG2;    // GDO2 output pin configuration
    BYTE IOCFG0;    // GDO0 output pin configuration
    BYTE PKTCTRL1;  // Packet automation control.
    BYTE PKTCTRL0;  // Packet automation control.
    BYTE ADDR;      // Device address.
    BYTE PKTLEN;    // Packet length.
} RF_SETTINGS;

```

Δήλωση global σταθερών, σχετικών με την λειτουργία τόσο του ραδιοφώνου, όσο και του μικροελεγκτή. Π.χ. η ενεργοποίηση του crossbar decoder, ο οποίος αντιστοιχίζει εσωτερικές διατάξεις του μικροελεγκτή σε εξωτερικά pins, θα γίνει μέσω της σταθεράς BM\_XBAR. Η η απενεργοποίηση των weak pull-ups σε συγκεκριμένα pins εισόδου του μικροελεγκτή γίνεται μέσω της σταθεράς BM\_WEAKPUD. Υπενθυμίζεται πως τα weak pull-ups χρειάζονται όταν το port pin συνδέεται σε πολλαπλές συσκευές, μέσω κάποιου bus, και υπάρχει προφανώς η ανάγκη, η λογική τιμή της εξόδου μιας συσκευής να μεταφέρεται ηλεκτρικά στις υπόλοιπες.

```

#define MODE_NOT_SET          0
#define TX                     1
#define RX                     2
#define CRC_OK                0x80
#define RSSI                  0
#define LQI                    1
#define BYTES_IN_RXFIFO        0x7F
#define BM_XBAR                0x40      // Crossbar Enable
#define BM_PCAOME_1            0x01      // 001: CEX0 routed to Port pin.
#define BM_WEAKPUD              0x80      // Port I/O Weak Pull-up Disable:
                                         // 0: Weak Pull-ups enabled
                                         // (except for Ports whose I/O
                                         // are configured as analog input
                                         // or push-pull output).
                                         // 1: Weak Pull-ups disabled.

```

Για το Port 0:

Δήλωση μεταβλητών όσον αφορά το serial port interface (SPI) και ανάθεση μεταβλητών σε ορισμένα pins με την χρήση της συνάρτησης SBIT. Σύμφωνα με το Σχήμα 2, το P0.0 μεταφέρει το ρολόι (SCK) από τον μικροελεγκτή στο ραδιόφωνο, το P0.1 μεταφέρει το σήμα MISO από το ραδιόφωνο προς τον μικροελεγκτή, το P0.2 μεταφέρει το σήμα MOSI από τον μικροελεγκτή προς το ραδιόφωνο και το P0.3 μεταφέρει το σήμα ενεργοποίησης της επικοινωνίας CSn από τον μικροελεγκτή προς το ραδιόφωνο. Παρατηρήστε πως η έξοδος στα σχόλια του κώδικα, αναφέρεται με βάση το ραδιόφωνο, δηλ. το σήμα εξόδου στο ραδιόφωνο είναι σήμα εισόδου στον μικροελεγκτή και αντίστροφα. Προφανώς, ο μηχανικός ήδη γνωρίζει πως οι θύρες (ports) 0 και 1 βρίσκονται στον μικροελεγκτή.

Για το Port 2:  
Δήλωση των δύο LEDs και ανάθεσή τους στο P2.2 και P2.3 με την χρήση της SBIT. Για ποιόν λόγο ονομάσαμε τα συγκεκριμένα pins P2.2, P2.3 ως LED, LED2 αντίστοιχα;

```
// Port 0
#define SCLK_          0x01
// P0.0 SPI Serial clock
//Input (mcu output)
//
#define SO_            0x02
// P0.1 SPI MISO signal, xx00 SO/GD01
//Output(from cc2500 to mcu)
#define GD01_          0x02
// P0.1
//
#define SI_            0x04
// P0.2 SPI MOSI signal, xx00 SI
//Input (mcu output)
//
#define CSn_           0x08
// P0.3 SPI slave select signal
//cc2500 Input (mcu output)
//
#define GD00_          0x40
// P0.6 xx00 GD00
//cc2500 Output (mcu input)
//
#define GD02_          0x80
// P0.7 xx00 GD02
//cc2500 Output (mcu input)

SBIT(P0_1, SFR_P0, 1);
SBIT(GD00_PIN, SFR_P0, 6);

//Port 2
#define LED1_          0x04
// P2.2      LED1, (Green)
// mcu output

#define LED2_          0x08
// P2.3      LED2, (Green)
//mcu output

SBIT(LED, SFR_P2, 2); // LED='1' means ON
SBIT(LED2, SFR_P2, 3); // LED='1' means ON
```

```
SBIT(SW2, SFR_P2, 0); // SW2='0'  
// means switch pressed
```

Η πρώτη βασική λειτουργία αρχικοποίησης ενός ψηφιακού μικροελεγκτή είναι ο καθορισμός του ρολογιού του συστήματος (system clock). Το ρολόι του συστήματος ορίζεται με βάση έναν ταλαντωτή, του οποίου η βασική συχνότητα πολλαπλασιάζεται ή/και διαιρείται, έτσι ώστε να παραχθεί η τελική τιμή της συχνότητας του ρολογιού. Ο βασική συχνότητα του ταλαντωτή παράγεται είτε από εσωτερικό κύκλωμα (συνήθως με χρήση RC κυκλωμάτων) ή με χρήση εξωτερικού κρυστάλλου. Στην δική μας περίπτωση θα χρησιμοποιηθεί εσωτερικός ταλαντωτής (ο οποίος εν γένει είναι λιγότερο ακριβής και θερμοκρασιακά πιο ασταθής σε σχέση με ένα εξωτερικό κρυσταλλικό ταλαντωτή). Σημειώνεται επίσης, πως το ραδιόφωνο έχει δικό του εξωτερικό, κρυσταλλικό ταλαντωτή.

Πριν την αρχικοποίηση του ρολογιού του συστήματος (δηλ. του system clock του μικροελεγκτή), απενεργοποιείται ο Watchdog timer, θέτοντας το bit6 (bit WTDE) του καταχωρητή PCA0MD ίσο με το '0':

```
PCA0MD &= ~0x40;
```

Η αρχικοποίηση του system clock απαιτεί τον χειρισμό των καταχωρητών CLKMUL, CLKSEL και OSCICN του μικροελεγκτή. Συγκεκριμένα, η εντολή

```
CLKMUL = INT_OSC = 0x00;
```

επιλέγει ως είσοδο του 4-πλού πολλαπλασιαστή συχνότητας, τον βασικό εσωτερικό ταλαντωτή του μικροελεγκτή. Υπενθυμίζεται πως ο εσωτερικός ταλαντωτής του μικροελεγκτή παράγει σήμα συχνότητας 12MHz. Συνεπώς, έξοδος του 4-πλού πολλαπλασιαστή συχνότητας, μπορεί να είναι ένα σήμα συχνότητας 48MHz, το οποίο απαιτείται στην επικοινωνία, μέσω USB.

Ωστόσο, ο χειρισμός του καταχωρητή CLKMUL απαιτεί μια σειρά συγκεκριμένων βημάτων, σύμφωνα με το εγχειρίδιο του μικροελεγκτή:

1. Reset του πολλαπλασιαστή γράφοντας την τιμή του INT\_OSC ( $0 \times 00$ ) στον καταχωρητή CLKMUL.
2. Επιλογή της πηγής του πολλαπλασιαστή από τα bits CLKMUL.1-0 (bit MULSEL).
3. Ενεργοποίηση του πολλαπλασιαστή, θέτοντας σε λογικό '1' το bit CLKMUL.7 (bit MULEN).
4. Καθυστέρηση για περίπου  $> 5\text{ms}$
5. Αρχικοποίηση του πολλαπλασιαστή, θέτοντας σε λογικό '1' το bit CLKMUL.6 (bit MULINIT).
6. Μετά την αρχικοποίηση ελέγχουμε αν το bit CLKMUL.5, που υποδεικνύει την ετοιμότητα του 4-πλού πολλαπλασιαστή, είναι ίσο με '1'. Όσο δεν είναι έτοιμος, δηλαδή:

!(CLKMUL & BM\_MULRDY),

περιμένουμε.

Τα βήματα 1 και 2 έχουν ήδη περιγραφεί, ενώ τα βήματα 3-6 περιγράφονται από τις παρακάτω εντολές (μία γραμμή για κάθε βήμα):

```
CLKMUL |= 0x80;
for (i = 0; i < 20; i++);
CLKMUL |= 0xC0;
while (!(CLKMUL & BM_MULRDY));
```

Στο σημείο αυτό, έχουμε απλά δημιουργήσει την έξοδο του 4-πλού πολλαπλασιαστή, η οποία είναι μια κυμματομορφή βασικής συχνότητας 48MHz. Δεν έχουμε ακόμη ορίσει την συχνότητα στην οποία θα λειτουργήσει ο μικροελεγκτής. Χρειάζονται δύο ακόμη βήματα:

```
CLKSEL |= FOUR_X_CLK_MULT |= 0x02;
OSCICN = INT_OSC_DIV_1 |= 0x83;
```

Στην πρώτη γραμμή, καθορίζονται τα bits CLKSEL.1-0, βάσει των οποίων προκύπτει η πηγή του ρολογιού του συστήματος. Οι πιθανές είσοδοι φαίνονται παρακάτω:

CLKSL.1	CLKSL.0	Selected Clock
00		Εσωτερικός ταλαντωτής
01		Εξωτερικός ταλαντωτής
10		SYSCLK = 4 × ClockMultiplier/2
11		RESERVED

Table 1: Εύρος τιμών CLKSL.1-0.

Σύμφωνα με τα παραπάνω, η πηγή του ρολογιού είναι το σήμα του 4-πλού πολλαπλασιαστή (48MHz), διαιρεμένου διά δύο (24MHz).

Στην συνέχεια, ενεργοποιούμε τον εσωτερικό ταλαντωτή θέτοντας το bit IOSCEN.7 ίσο με '1' και τα bits OSCICN.1-0 (IFCN1, IFCN0) του καταχωρητή OSCICN ίσα με '1'. Τα δύο πρώτα bits IFCN0 και IFCN1 του καταχωρητή OSCICN ορίζουν την τελική συχνότητα του ρολογιού του συστήματος, η οποία είναι η συχνότητα που έχει οριστεί από τον καταχωρητή CLKSL, διαιρεμένη με συγκεκριμένο λόγο. Ο λόγος αυτός, στην γενική περίπτωση δίνεται από τον παρακάτω πίνακα:

IFCN0	IFCN1	System Clock Frequency
0	0	Fclksl divided by 8
0	1	Fclksl divided by 4
1	0	Fclksl divided by 2
1	1	Fclksl divided by 1

Table 2: Εύρος τιμών για τα bits IFCN0 και IFCN1, του καταχωρητή OSCICN.

Για τις τιμές, OSCICN.IFCN0=1 και OSCICN.IFCN1=1, ο λόγος διαίρεσης είναι μονάδα, και συνεπώς το ορισμένο ρολόι του συστήματος είναι η συχνότητα που ορίστηκε στον καταχωρητή CLKSL, δηλαδή τα 24MHz.

```

#define INT_OSC          0x00
#define FOUR_X_CLK_MULT 0x02
#define INT_OSC_DIV_1    0x83
#define BM_MULRDY      0x20 // Clock Multiplier Ready.
                           // This read-only bit
                           // indicates the status of the
                           // Clock Multiplier.
                           // 0: Clock Multiplier not ready.
                           // 1: Clock Multiplier ready (locked).
#define CLOCK_INIT()
do {
    UINT8 i;
    PCAOMD &= ~0x40;
    CLKMUL = INT_OSC;
    CLKMUL |= 0x80;
    for (i = 0; i < 20; i++);
    CLKMUL |= 0xC0;
    while (!(CLKMUL & BM_MULRDY));
    CLKSEL |= FOUR_X_CLK_MULT;
    OSCICN = INT_OSC_DIV_1;
} while (0)

```

Στην συνέχεια, ορίζουμε τις σταθερές (burst/single access, master/slave mode) και τις συναρτήσεις (bus ενεργό/ανενεργό) που καθορίζουν την λειτουργία του SPI bus.

```

#define WRITE_BURST 0x40
#define READ_SINGLE 0x80
#define READ_BURST 0xC0
#define BM_SPIEN     0x01// SPI0 Enable:
                           // Active high
#define BM_MSTEN     0x40// Master Mode Enable:
                           // Active High
#define BM_NSSMD1    0x08// Slave Select Mode
#define BM_SPIOE     0x02// SPI I/O Enable
#define SPI_ENABLE()  (SPIOCN |= BM_SPIEN)
#define SPI_DISABLE() (SPIOCN &= ~BM_SPIEN)

```

Με συγκεκριμένες τιμές στους καταχωρητές ρύθμισης και ελέγχου του SPI bus στον μικροελεγκτή, 4wire single master mode (SPI0CFG = BM\_MSTEN) σύμφωνα με το Σχήμα 2, και ορίζουμε το ρολόι του SPI bus ίσο με “freq”. Το “freq” μπορεί να πάρει τιμές 1,2,3,4,5,7,9 ή 11 MHz. Το NSS είναι μία έξοδος του σήματος ελέγχου από τον MCU. Θέτουμε τον καταχωρητή ελέγχου SPI0CN ίσο με BM\_NSSMD1 (Slave Select Mode). Τέλος, ενεργοποιούμε το SPI.

```

#define SPI_INIT(freq)
    do {
        SPI0CFG = BM_MSTEN; //SPI0 config.register
        SPI0CN = BM_NSSMD1; //SPI0 control register
        SPIOCKR = freq; //SPI0 clock register
        SPI_ENABLE(); //enable SPI0
    } while (0)

// where freq is one of:
#define SCLK_6_MHZ      1
#define SCLK_4_MHZ      2
#define SCLK_3_MHZ      3
#define SCLK_2_4_MHZ    4
#define SCLK_2_MHZ       5
#define SCLK_1_5_MHZ    7
#define SCLK_1_2_MHZ    9
#define SCLK_1_MHZ      11

```

### Πώς δουλεύει το SPI αναλυτικά

To Serial Peripheral Interface (SPI) είναι μια σειριακή διεπαφή επικοινωνίας μεταξύ δύο συσκευών. Στην περίπτωσή μας, η εν λόγω διεπαφή χρησιμοποιείται στην επικοινωνία ανάμεσα στην πλακέτα του μικροεπεξεργαστή και το ραδιόφωνο. Πιο συγκεκριμένα, η «επικοινωνία» αφορά στην δυνατότητα καθορισμού από τον μικροεπεξεργαστή των κύριων παραμέτρων ελέγχου του ραδιοφώνου, καθώς και την μεταφορά δεδομένων από και προς το τελευταίο.

Ο microcontroller λειτουργεί ως SPI master device και είναι εκείνος που εκκινεί όλες τις μεταφορές δεδομένων πάνω στο δίαυλο SPI. Σε επίπεδο κώδικα, ο MCU καθορίζεται ως master ενεργοποιώντας το bit 6 του καταχωρητή SPI0CFG (Master Enable flag -MSTEN). Αντίθετα, το ραδιόφωνο, που είναι slave συσκευή, αποκτά πρόσβαση στον δίαυλο μόνο κατόπιν αίτησης του master. Μέσω του SPI, ο μικροεπεξεργαστής αναθέτει τιμές στους καταχωρητές ελέγχου του ραδιοφώνου ρυθμίζοντας με τον τρόπο αυτό τις παραμέτρους λειτουργίας του τελευταίου. Φυσικά, μέσω του διαύλου πραγματοποιείται και η μεταφορά οποιονδήποτε άλλων δεδομένων (αποστολή δεδομένων από τον επεξεργαστή στο εξωτερικό περιβάλλον μέσω του ραδιοφώνου και λήψη δεδομένων που ακολουθούν την αντίστροφη πορεία).

Για την αποστολή δεδομένων προς το slave device, ο master γράφει δεδομένα στον καταχωρητή SPI0DAT τα οποία κατόπιν μεταφέρονται στο buffer μετάδοσης. Εάν ο shift-register που χρησιμοποιείται για την εισαγωγή των δεδομένων στο σειριακό δίαυλο είναι άδειος, τότε το byte μεταφέρεται από το transmit-buffer στον shift-register και εκκινείται η μεταφορά των δεδομένων.

Στο τέλος της μετάδοσης, το hardware θέτει αυτόματα το bit SPIF (SPI0CN.7) ίσο με λογικό 1 για να δηλώσει το πέρας της μεταφοράς. Επιπλέον, αν τα interrupts είναι ενεργοποιημένα, τότε παράγεται μία εξαίρεση κατά την ενεργοποίηση του SPIF flag. Στον δεδομένο κώδικα, μετά από κάθε εγγραφή δεδομένων στον καταχωρητή SPI0DAT, καλείται η συνάρτηση «spi\_wait» η οποία πίπτει σε βρόγχο μέχρις ώτου ενεργοποιηθεί το SPIF flag. Κατόπιν απενεργοποιεί το SPIF flag και επιστρέφει. Μετά την επιστροφή της συνάρτησης, είναι δυνατή η εγγραφή νέων δεδομένων στον καταχωρητή SPI0DAT.

Καθώς ο master μεταφέρει δεδομένα προς το slave device μέσω της γραμμής MOSI, το slave device μεταφέρει δεδομένα πίσω στον master μέσω της γραμμής MISO. Δηλαδή, η μετάδοση δεδομένων είναι full-duplex. Επιπλέον, το SPIF flag ενεργοποιείται για να δηλώσει τόσο την ολοκλήρωση της αποστολής δεδομένων από τον master στον slave αλλά και την ολοκλήρωση της λήψης δεδομένων που ακολούθησαν την αντίστροφη πορεία. Μάλιστα, για την ανάγνωση ενός (λήψη δεδομένων από) καταχωρητή στο slave, ο master γράφει την διεύθυνση του καταχωρητή-προέλευσης, και σε δεύτερο κύκλο στέλνει την τιμή 0 (βλ. συναρτήσεις «halSpiReadStatus» και «halSpiReadReg») ενώ παράλληλα παραλαμβάνει τα επιψυμητά δεδομένα. Ο καταχωρητής SPIODAT χρησιμοποιείται δηλαδή, τόσο για την αποστολή όσο και για τη λήψη δεδομένων: η εγγραφή δεδομένων σε αυτόν, οδηγεί τα τελευταία στο transmit-buffer και εκκινεί την μεταφορά εφόσον μιλάμε για τη συσκευή master, ενώ η ανάγνωση δεδομένων από τον ίδιο καταχωρητή επιστρέφει τα περιεχόμενα του receive-buffer.

Τα δεδομένα από τον slave προς τον master μεταφέρονται σειριακά με πρώτο το MSB και τοποθετούνται στον shift-register του master. Μετά την ολοκλήρωση της λήψης, το ληφθέν byte που έχει εισαχθεί πλήρως στον shift-register, μεταφέρεται στο receive buffer όπου είναι προσπελάσιμο από τον επεξεργαστή διαβάζοντας τον καταχωρητή SPIODAT.

Κατά τη διάρκεια των μεταφορών στο δίαυλο SPI, το CSn bit πρέπει να παραμένει απενεργοποιημένο. Τυχόν άκαιρη ενεργοποίησή του στο μέσο μεταφοράς, ακυρώνει την τελευταία. Όταν το σήμα CSn απενεργοποιείται, το MCU πρέπει να περιμένει πριν την έναρξη της μεταφοράς δεδομένων με την εγγραφή του header byte, μέχρις ώτου απενεργοποιηθεί το σήμα SO. Βέβαια, πέρα από τις περιπτώσεις που το ραδιόφωνο είναι σε μία από τις καταστάσεις SLEEP ή XOFF, το SO pin θα ακολουθήσει άμεσα την απενεργοποίηση του CSn.

Το SPI εν γένει υποστηρίζει αρκετά διαφορετικά modes λειτουργίας. Στην περίπτωσή μας χρησιμοποιείται το 4-wire single-master mode το οποίο καθορίζεται με την ενεργοποίηση του bit SPIOCN.3. Στο εν λόγω mode, το σήμα NSS χρησιμοποιείται ως pin εξόδου και μπορεί να χρησιμοποιηθεί ως slave select signal για το ένα και μοναδικό SPI-Slave device. Ουσιαστικά είναι αυτό το σήμα που συνδέεται στην ίδια γραμμή με το CSn pin του ραδιοφώνου και ελέγχει την μεταφορά των δεδομένων. Η τιμή του καθορίζεται από το bit 2 του καταχωρητή SPIOCN (NSSMD0).

### • Byte κατάστασης chip

Όταν το header byte, data byte, ή μία εντολή στέλνεται από τον μικροελεγκτή στην διεπαφή SPI, τότε ένα status byte που υποδεικνύει την κατάσταση του ραδιοφώνου, στέλνεται από το ραδιόφωνο στο pin SO. Αυτό το byte κατάστασης περιέχει σήματα κατάστασης, χρήσιμα για τον μικροελεγκτή. Το πρώτο bit είναι το σήμα CHIP\_RDYn. Αυτό το σήμα πρέπει να ‘κατέβει’ σε λογικό ’0’, πριν από την πρώτη θετική ακμή του SCLK. Το σήμα αυτό υποδεικνύει ότι ο χρύσταλλος του ΡΑΔΙΟΦΩΝΟΥ λειτουργεί. Τα bits 6, 5 και 4 εμπεριέχουν την τιμή της κατάστασης του ραδιοφώνου (οι καταστάσεις και οι 3-bit κωδικοί τους επεξηγήθηκαν στην εισαγωγή). Σημειώνεται πως ο ταλαντωτής και η ισχύς στον ψηφιακό πυρήνα του ραδιοφώνου είναι ενεργά στην κατάσταση IDLE, ενώ όλα τα άλλα modules είναι ανενεργά. Η συχνότητα και η επιλογή του καναλιού πρέπει να ανανεώνονται όταν το ραδιόφωνο βρίσκεται σε αυτήν την κατάσταση. Η κατάσταση RX θα είναι ενεργή όταν το ραδιόφωνο βρίσκεται στην φάση λήψης δεδομένων. Αντίστοιχα η κατάσταση TX θα είναι ενεργή όταν το ραδιόφωνο μεταδίδει δεδομένα.

Τα τρία τελευταία bits στο byte κατάστασης ονομάζονται FIFO\_BYTES\_AVAILABLE. Για λειτουργίες ανάγνωσης όταν το bit εγγραφής/ανάγνωσης, R/W, στο header byte είναι '1', το πεδίο FIFO\_BYTES\_AVAILABLE περιέχει τον αριθμό των διαθέσιμων bytes που είναι διαθέσιμα προς ανάγνωση στην RX FIFO του ραδιοφώνου. Για λειτουργίες εγγραφής, όταν το bit εγγραφής/ανάγνωσης R/W στο header byte είναι '0', το πεδίο FIFO\_BYTES\_AVAILABLE περιέχει τον αριθμό των bytes που έχουν απομείνει προς εγγραφή στην TX FIFO.

- **Πρόσβαση στους καταχωρητές**

Οι καταχωρητές αρχικοποίησης/ελέγχου (configuration registers) του ραδιοφώνου είναι το ποινητημένοι στις διευθύνσεις από το 0x00 μέχρι το 0x2E. Στο εργαστήριο όμως χρησιμοποιούμε το συμπληρωματικό λογισμικό Smart RF Studio, ώστε να παράγουμε τις βέλτιστες ρυθμίσεις. Όταν γράφουμε στους καταχωρητές αυτούς, το byte κατάστασης αποστέλλεται στο pin SO, κάθε φορά που ένα header byte ή ένα byte δεδομένων μεταδίδεται στο SI pin. Όταν διαβάζουμε από τους καταχωρητές, το byte κατάστασης αποστέλλεται στο pin SO, κάθε φορά που ένα header byte μεταδίδεται στο pin SI.

Καταχωρητές με διαδοχικές διευθύνσεις μπορούν να προσπελαστούν με αποδοτικό τρόπο θέτοντας το burst bit στο header byte ίσο με '1'. Τα bits διεύθυνσης ορίζουν την εναρκτήρια διεύθυνση σ' έναν εσωτερικό μετρητή διεύθυνσεων. Αυτός ο μετρητής αυξάνεται κατά 1 κάθε 8 παλμούς ρολογιού. Η πρόσβαση burst, είτε εγγραφής είτε ανάγνωσης, μπορεί να τερματιστεί "ανεβάζοντας" το CSn σε λογική κατάσταση '1'.

Για διευθύνσεις καταχωρητών από 0x30-0x3D, το burst bit χρησιμοποιείται για να επιλέξουμε μεταξύ καταχωρητών κατάστασης (burst bit=1) και καταχωρητών εντολών (burst bit=0). Εξαιτίας των παραπάνω η burst πρόσβαση δεν είναι διαθέσιμη για καταχωρητές κατάστασης ή εντολών, και σαν αποτέλεσμα πρέπει να προσπελαύνονται ένας την φορά. Στο σημείο αυτό σημειώνουμε ότι οι καταχωρητές κατάστασης είναι διαθέσιμοι μόνο προς ανάγνωση.

- **Ανάγνωση πάνω από την διεπαφή SPI**

Όταν διαβάζουμε πεδία καταχωρητών μέσω SPI, υπάρχει η πιθανότητα ενημέρωσής τους από το ραδιόφωνο. Συνεπώς, υπάρχει η πιθανότητα μη έγκυρης ανάγνωσης ενός καταχωρητή του ραδιοφώνου.

- **Εντολές**

Οι εντολές μπορούν να θεωρηθούν σαν οδηγίες μήκους ενός byte προς το ραδιόφωνο. Αυτές οι εντολές χρησιμοποιούνται για την απενεργοποίηση του ταλαντωτή κρυστάλλου, ενεργοποιούν την φάση λήψης κλπ. Οι καταχωρητές εντολών μπορούν να προσπελαστούν με την μεταφορά ενός header byte.

Κατά την εγγραφή σε καταχωρητή εντολών, το byte κατάστασης αποστέλλεται στο pin SO από το ραδιόφωνο. Μία εντολή μπορεί να ακολουθείται από οποιαδήποτε άλλη πρόσβαση στο SPI bus, χωρίς την 'ανόρθωση' του CSn. Ωστόσο, εάν πρόκειται για την εντολή SRES, όμως πρέπει να περιμένουμε να 'κατέβει' ξανά το SO προτού μπορέσουμε να δηλώσουμε το επόμενο header byte. Οι εντολές εκτελούνται άμεσα με εξαίρεση τις SPWD και SXOFF, που εκτελούνται με την πρώτη ανόρθωση του CSn.

- **Πρόσβαση στις στοίβες FIFO**

Η 64-bit FIFO στοίβα TX και η 64-bit FIFO στοίβα RX μπορει να προσπελαστεί μέσω της διεύθυνσης 0x3F. Όταν το bit R/W είναι '0', μπορούμε να προσπελάσουμε την TX FIFO, ενώ όταν είναι '1' την RX FIFO. Η TX FIFO είναι διαθέσιμη μόνο προς εγγραφή ενώ η RX FIFO είναι διαθέσιμη μόνο προς ανάγνωση. Το burst bit χρησιμοποιείται για να καθορίσει αν η πρόσβαση στην FIFO είναι ενός byte ή burst πρόσβαση. Η πρόσβαση ενός byte προϋποθέτει ένα header byte με μηδενικό burst bit. Μετά το byte πληροφορίας αναμένεται ένα νέο header byte. Ως εκ τούτου, το pin CSn μπορεί να παραμείνει "κατεβασμένο". Αντιθέτως η burst πρόσβαση προϋποθέτει ένα header byte, ακολουθούμενο από διαδοχικά bytes πληροφορίας, η μεταφορά των οποίων τερματίζεται από την 'ανόρθωση' του CSn pin.

Όταν γράφουμε στην TX FIFO, το byte κατάστασης είναι η έξοδος για κάθε νέο byte πληροφορίας στο pin SO. Αυτό το byte κατάστασης μπορεί να χρησιμοποιηθεί για την ανίχνευση υποχείλισης της TX FIFO κατά την εγγραφή δεδομένων. Στο σημείο αυτό να σημειώσουμε ότι το byte κατάστασης εμπεριέχει το πλήθος των διαθέσιμων bytes, πριν την εγγραφή του τρέχοντος byte στην TX FIFO. Αυτό σημαίνει ότι όταν μεταδοθεί στο SI το τελευταίο byte που χωράει στην TX FIFO, το byte κατάστασης που εκείνη την στιγμή αποστέλλεται στο SO θα δείχνει ότι υπάρχει ένα διαθέσιμο byte στην TX FIFO.

Η TX FIFO πρέπει να εκκαθαριστεί με την δήλωση της εντολής SFTX. Παρομοίως η εντολή SFRX αδειάζει την RX FIFO. Μία εντολή σαν τις παραπάνω μπορεί να δηλωθεί στις καταστάσεις: IDLE, TXFIFO\_UNDERFLOW ή RXFIFO\_OVERFLOW ή όταν το ραδιόφωνο περιέλθει στην κατάσταση SLEEP.

- **Πρόσβαση στον PATABLΕ**

Η διεύθυνση 0x3E χρησιμοποιείται για την προσπέλαση του καταχωρητή PATABLΕ, που σχετίζεται με την ισχύ εκπομπής (power control για τον power amplifier). Το PATABLΕ είναι ένας πίνακας των 8-byte, όμως δεν χρησιμοποιούνται όλες οι εγγραφές του. Οι εγγραφές (bytes) που θα χρησιμοποιηθούν επιλέγονται από την 3-bit ποσότητα FREND0.PA\_POWER.

Σε περίπτωση 2-FSK, GFSK ή MSK διαμόρφωσης, γίνεται χρήση μόνο της πρώτης εγγραφής του πίνακα. Αντιθέτως, χρησιμοποιούνται οι δύο πρώτες εγγραφές του πίνακα, όταν επιλέγεται διαμόρφωση OOK. Εφόσον το PATABLΕ είναι ένας 8-byte πίνακας, γράφεται και διαβάζεται από την χαμηλότερη προς την υψηλότερη ρύθμιση, 1 byte την φορά. Ένας byte μετρητής index χρησιμοποιείται κάθε φορά για τον έλεγχο της πρόσβασης στον πίνακα.

Η πρόσβαση στο PATABLΕ είναι ενός byte ή πολλαπλή burst (εξαρτάται από το burst bit). Όταν χρησιμοποιούμε burst πρόσβαση, ο μετρητής index αυξάνει και όταν πάρει την τιμή 7, ξανα-ξεκινάει από το 0. Το bit R/W καθορίζει αν η πρόσβαση είναι πρόσβαση εγγραφής (R/W = 0) ή πρόσβαση ανάγνωσης (R/W = 1).

Σημειώνουμε ότι εάν ένα byte είναι εγγεγραμμένο στο PATABLΕ και πρόκειται να διαβαστεί η τιμή του, τότε το CSn πρέπει να 'ανορθωθεί' (λογική κατάσταση '1') πριν την πρόσβαση ανάγνωσης, έτσι ώστε να μηδενιστεί ο μετρητής index. Το περιεχόμενο του PATABLΕ, εκτός από το πρώτο byte, χάνεται όταν το ραδιόφωνο μπει σε κατάσταση SLEEP.

Στο σημείο αυτό αναφέρουμε τις καταστάσεις του ραδιοφώνου σε μορφή state diagram.

**Περιγραφή των καταστάσεων**

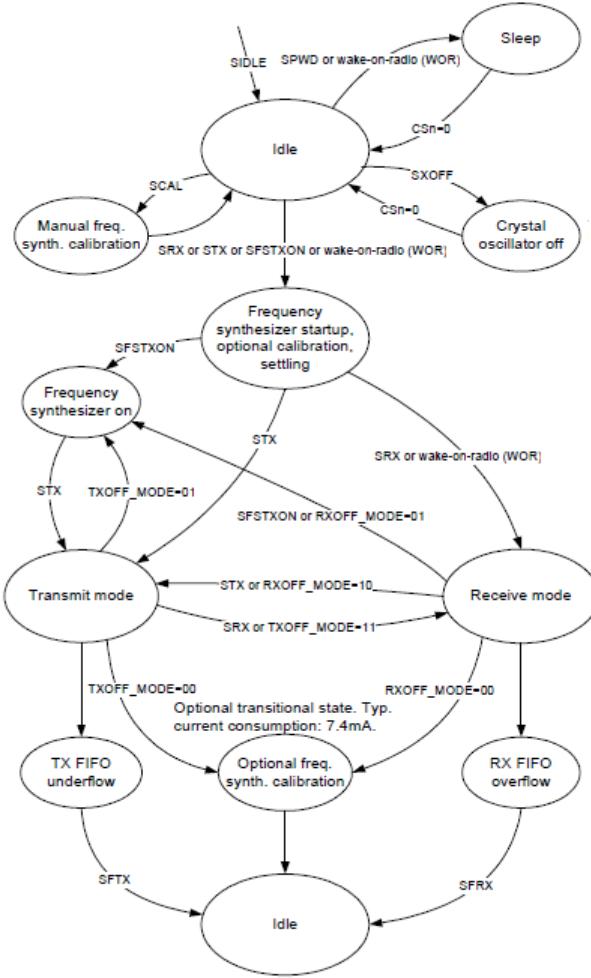


Figure 3: Απλοποιημένο διάγραμμα καταστάσεων.

- Sleep: Η κατάσταση που απαιτεί την λιγότερη ενέργεια. Οι τιμές των περισσότερων καταχωρητών διατηρούνται.
- Idle: Κατάσταση αρχικοποίησης, όταν το ραδιόφωνο δεν λειτουργεί ούτε ως πομπός ούτε ως δέκτης.
- Manual Frequency Synthesizer Calibration: Calibrating synthesizer upfront-Μεταβατική κατάσταση.
- Crystal Oscillator OFF: Διατηρούνται όλες οι τιμές των καταχωρητών.
- Frequency Synthesizer Start up, Optional Calibration, Settling: Ενεργοποιείται ο frequency synthesizer (optionally calibrated) και τοποθετείται στην σωστή συχνότητα.
- Frequency Synthesizer On: Ενεργοποιεί τον frequency synthesizer και είμαστε έτοιμοι για να ξεκινήσει η αποστολή. Η αποστολή ξεκινάει σχεδόν αμέσως μετά την λήψη του command strobe STX.

- Transmit Mode: Λειτουργία πομπού.
- Receive Mode: Λειτουργία δέκτη.
- TX FIFO underflow: Παύση της διαδικασίας μετάδοσης. Είσοδος σε αυτήν την κατάσταση όταν στην μέση ενός πακέτου αδειάσει η στοίβα TX FIFO.
- RX FIFO overflow: Παύση της διαδικασίας λήψης και είσοδο σε αυτήν την κατάσταση όταν υπερχειλίσει η RX FIFO.
- Optional Frequency Synthesizer Calibration: Προαιρετική μεταβατική κατάσταση.

Η κάθε κατάσταση από τις παραπάνω απαιτεί διαφορετική ενεργειακή κατανάλωση. Έτσι μπορούμε να κάνουμε μεταβάσεις σωστής διάρκειας και να ανταποκριθούμε στις ενεργειακές απαιτήσεις της εργασίας που καλούμαστε να υλοποιήσουμε κάθε φορά.

### Δήλωση των πρωτοτύπων των συναρτήσεων

Οι συναρτήσεις όμως περιγραφούν αναλυτικά παρακάτω.

```
void spi_wait(void);
void halSpiWriteReg(BYTE addr, BYTE value);
void halRfWriteRfSettings(RF_SETTINGS *pRfSettings);
void halRfSendPacket(BYTE *txBuffer, UINT8 size);
void halSpiWriteBurstReg(BYTE addr, BYTE *buffer, BYTE count);
void halSpiStrobe(BYTE strobe);
BYTE halSpiReadStatus(BYTE addr);
BYTE halSpiReadReg(BYTE addr);
void halSpiReadBurstReg(BYTE addr, BYTE *buffer, BYTE count);
BOOL halRfReceivePacket(BYTE *rxBuffer, UINT8 *length);
void halWait(UINT16 timeout);
void intToAscii(UINT32 value);
void SYSCLK_Init (void);
void PORT_Init (void);
void Timer2_Init (int counts);
INTERRUPT_PROTO(Timer2_ISR, INTERRUPT_TIMER2);
```

### Διαφορά τύπων μνήμης xdata και pdata

Η 8051 αρχιτεκτονική του μικροελεγκτή που χρησιμοποιούμε στο εργαστήριο προσφέρει δύο τύπους μνήμης για προσπέλαση δεδομένων: xdata και pdata: το xdata αναφέρεται σε οποιαδήποτε τοποθεσία στον 64-Byte χώρο διευθύνσεων μιας μνήμης δεδομένων. Το μοντέλο μεγάλης μνήμης - "large memory model"- αποθηκεύει μεταβλητές σε αυτόν τον χώρο μνήμης. Το pdata αναφέρεται σε ακριβώς μία σελίδα (256-Bytes) μνήμης δεδομένων. Το μοντέλο συμπυκνωμένης μνήμης - "compact memory model"- τοποθετεί μεταβλητές σε αυτόν τον χώρο μνήμης.

### Δήλωση των Global μεταβλητών

```

BYTE xdata txBuffer[] = {20, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14, 15, 16, 17, 18, 19};
//symbol sequence to be transmitted
BYTE xdata rxBuffer[61]; // Length byte
// + 2 status bytes are not
//stored in this buffer
//buffer for the received symbolstream
UINT8 xdata mode = MODE_NOT_SET; //=0
BYTE xdata asciiString[11];

```

### Αρχικοποίηση ραδιοφώνου

Στο σημείο αυτό, ρυθμίζουμε το ραδιόφωνο, δίνοντας τιμές στα κατάλληλα bits των καταχωρητών που ορίζουν την λειτουργία του (configuration registers), με χρήση του τύπου δεδομένων code (υποδεικνύεται ότι η συγκεκριμένη δομή θα αποθηκευτεί σε ένα διαφορετικό κομμάτι της μνήμης και θα αντιμετωπιστεί ως κώδικας). Μερικές βασικές παράμετροι είναι η συχνότητα (ή το κανάλι εκπομπής), η διαμόρφωση, η ισχύς εκπομπής, η δυνατότητα διόρθωσης σφαλμάτων μετάδοσης (FEC), η δυνατότητα data whitening, δηλ. η δυνατότητα τα δεδομένα μετάδοσης να μοιάζουν με λευκός θόρυβος (έτσι ώστε να αποφεύγονται μεγάλες σειρές συνεχόμενων '1' ή '0'), η δυνατότητα καθορισμού του μήκους του πακέτου πληροφορίας προς μετάδοση κλπ. Μερικά παραδείγματα ακολουθούν:

- επιλογή καναλιού,  
καταχωρητής: CHANNR  
bits: 0-7  
(CHANNR = 0 × 02 δίνει επιλογή 2ου καναλιού)  
Η τιμή του καναλιού (channel number) πολλαπλασιάζεται με την παράμετρο channel spacing και προστιθέμενο στην συχνότητα βάσης μας δίνει την ακριβή συχνότητα του καναλιού.
- τύπος διαμόρφωσης,  
καταχωρητής: MDMCFG2  
bits: 4-6  
(MDMCFG2 = 0 × 73 δίνει διαμόρφωση MSK)

Με την χρήση του συμπληρωματικού λογισμικού SmartRf Studio, το οποίο μπορούμε να προμηθευτούμε από την ιστοσελίδα του κατασκευαστή του ραδιοφώνου, παίρνουμε εύκολα τις κατάλληλες τιμές για όλους τους configuration καταχωρητές, ανάλογα με την εφαρμογή.

```

RF_SETTINGS code rfSettings = {
    0x0A, // FSCTRL1   Frequency synthesizer control.
    0x00, // FSCTRL0   Frequency synthesizer control.
    0x5D, // FREQ2     Frequency control word, high byte.
    0x93, // FREQ1     Frequency control word, middle byte.
    0xB1, // FREQ0     Frequency control word, low byte.
    0x2D, // MDMCFG4   Modem configuration.
}

```

```

0x3B,    // MDMCFG3   Modem configuration.
0x73,    // MDMCFG2   Modem configuration.
0x22,    // MDMCFG1   Modem configuration.
0xF8,    // MDMCFG0   Modem configuration.
0x0C,    // CHANR     Channel number.
0x01,    // DEVIATN   Modem deviation setting
           // (when FSK modulation is enabled).
0xB6,    // FREND1    Front end RX configuration.
0x10,    // FREND0    Front end TX configuration.
0x18,    // MCSMO     Main Radio Control State
           // Machine configuration.
0x1D,    // FOCCFG    Frequency Offset
           // Compensation Configuration.
0x1C,    // BSCFG     Bit synchronization Configuration.
0xC7,    // AGCCTRL2  AGC control.
0x00,    // AGCCTRL1  AGC control.
0xB0,    // AGCCTRL0  AGC control.
0xEA,    // FSCAL3    Frequency synthesizer calibration.
0x0A,    // FSCAL2    Frequency synthesizer calibration.
0x00,    // FSCAL1    Frequency synthesizer calibration.
0x11,    // FSCAL0    Frequency synthesizer calibration.
0x59,    // FTEST     Frequency synthesizer calibration.
0x88,    // TEST2     Various test settings.
0x31,    // TEST1     Various test settings.
0x0B,    // TEST0     Various test settings.
0x07,    // FIFOTHR   RXFIFO and TXFIFO thresholds.
0x29,    // IOCFG2    GD02 output pin configuration.
0x06,    // IOCFG0D   GD00 output pin configuration.
0x04,    // PKTCTRL1  Packet automation control.
0x05,    // PKTCTRL0  Packet automation control.
0x00,    // ADDR      Device address.
0xFF,    // PKTLEN    Packet length.
};


```

Η ισχύς της εξόδου θα είναι 0dBm. Αν θέταμε

BYTE code paTable = 0xFF

τότε η ισχύς της εξόδου θα γίταν 1dBm.

```

// PATABLE (0 dBm output power)
BYTE code paTable = 0xFE;

```

## Συνάρτηση main

Ακολουθεί η συνάρτηση main του πομπού. Παρακάτω αναλύουμε τα όσα υλοποιεί.

- Δήλωση ενός ακεραίου (32-bit) όσον αφορά το πλήθος των απεσταλμένων πακέτων και ενός ακόμα (8-bit) για το μήκος του κάθε πακέτου προς αποστολή.
- Κλήση των συναρτήσεων:
  - CLOCK\_INIT()
  - PORT\_init()
  - SPI\_INIT() με όρισμα SCKL\_6\_MHz

Σημειώνεται πως με την συγκεκριμένη αρχικοποίηση, η σύνδεση του περιφερειακών του SPI bus στα pins P0.0-P0.3 του μικροελεγκτή, ταυτίζεται με το Σχήμα 2.

- Περιμένουμε περίπου 41μsec για να ρυθμιστούν τα ηλεκτρονικά του ραδιοφώνου.
- Αποθηκεύουμε στον data register SPI0DAT ένα SRES command strobe. Η εντολή αυτή θέτει τους καταχωρητές του ραδιοφώνου στις default τιμές τους και το ραδιόφωνο περνάει σε κατάσταση IDLE. Αναλυτικά:
  - Strobe CSn low/high (μέσω του CSn ≡ pin NSSMD0 ≡ SPI0CN.2).
  - Παραμένει το CSn ‘ανεβασμένο’ για τουλάχιστον 41μs
  - ‘Κατεβαίνει’ το CSn και περιμένει το SO ≡ P0.1 να ‘κατέβει’ (CHIP\_RDYn).
  - Μεταφορά του SRES strobe στην γραμμή του SI, μέσω SPI bus.
  - Όταν ολοκληρωθεί η μεταφορά στο SPI bus και ενεργοποιηθεί το σχετικό interrupt flag (το οποίο απενεργοποιείται στην συνάρτηση spi\_wait()), το RESET ολοκληρώνεται και το ραδιόφωνο είναι στην κατάσταση IDLE.
- Κλήση της συνάρτησης halRfWriteRfSettings(), με όρισμα την δομή των κατάλληλων τιμών για τους configuration registers του ραδιοφώνου.
- Κλήση της συνάρτησης halSpiWriteReg() με ορίσματα CCxxx0.PATABLE και paTable, ώστε να εγγραφεί το paTable (πίνακας των ρυθμίσεων του power amplifier) στον αντίστοιχο καταχωρητή.
- Ενεργοποιούμε όλα τα interrupts θέτοντας το bit EA του καταχωρητή IE ίσο με ‘1’.
- Κλήση της συνάρτησης halRfSendPacket() με όρισμα την προς αποστολή ακολουθία και το μήκος αυτής (σε bytes). Η συνάρτηση αυτή μεταδίδει τα σύμβολα που είναι αποθηκευμένα στον txBuffer.
- Αποθηκεύουμε στην μεταβλητή packetsSent το πλήθος των απεσταλμένων πακέτων πληροφορίας.
- To LED που συνδέεται με το pin P2.2 αλλάζει κατάσταση.

Η αποστολή πακέτων πληροφορίας, όπως περιγράφηκε παραπάνω, είναι συνεχής. Αυτό σημαίνει πως το εν λόγω LED θα κάνει ουσιαστικά toggling, κάθε φορά που αποστέλλονται τα bytes πληροφορίας που βρίσκονται στον πίνακα txBuffer.

```
void main (void) {

    UINT32 packetsSent = 0;
    UINT8 length;

    CLOCK_INIT();      // Initialize clock
    PORT_Init ();       // Initialize crossbar and GPIO
    SPI_INIT(SCLK_6_MHZ); // Initialize SPI

    NSSMDO = 1;    // you need to wait
                   // ~41 usecs, before CC2500 responds
    halWait(1);
    NSSMDO = 0;
    halWait(1);
    NSSMDO = 1;
    halWait(41);

    //now reset the radio;
    do {
        NSSMDO = 0;
        while (P0_1);
        //this is necessary for
        //the specific cc2500 radio
        // see cc2500 manual page 21
        SPIODAT = CCxxx0_SRES;
        //reset cc2500 chip (see cc2500.h)
        spi_wait();
        //wait until data have been tranmited
        //through the spi interface
        NSSMDO = 1;
        //transmission complete
        // inform slave node (i.e. the cc2500)
    } while (0);

    halRfWriteRfSettings(&rfSettings);
    halSpiWriteReg(CCxxx0_PATABLE, paTable);

    EA = 1; // Enable global interrupts
    while (1){
        //TX mode
        halRfSendPacket(txBuffer, sizeof(txBuffer));
    }
}
```

```

    intToAscii(++packetsSent);

    //endof TX mode
    halWait(30000);
    halWait(30000);
    LED = ~LED;
}
}

```

### Ακολουθεί η συνάρτηση SYSCLK\_Init

Η συνάρτηση αυτή διαιρεί το ρολόι από τον καταχωρητή CLKSL με παράγοντα 8, σύμφωνα με τις τιμές των bits OSCICN.1-0. Συνεπώς, η συχνότητα του system clock του μικροελεγκτή, ορίζεται στα  $24/8 = 3\text{MHz}$ .

Έπειτα, ενεργοποιείται ο ανιχνευτής έλλειψης ρολογιού Missing Clock Detector (MCD), με την ανάθεση RSTSRC = 0x04. Ο ανιχνευτής είναι ένα κύκλωμα που πυροδοτείται από το system clock. Αν περάσουν περισσότερα από 100μs μεταξύ των θετικών ακμών του system clock, επιβάλλεται reset.

```

void SYSCLK_Init (void)
{
    OSCICN = 0x80; // Configure internal oscillator for
                    // its lowest frequency
    RSTSRC = 0x04; //Enable missing clock detector
}

```

### Ακολουθεί η συνάρτηση PORT\_Init

Τα σχόλια στον κώδικα επεξηγούν πλήρως τις αρχικοποιήσεις.

```

void PORT_Init (void)
{
    P2MDOUT = LED1_ | LED2_;
    // Enable two LEDs as a push-pull output at port 2

    P0MDOUT = SCLK_ | SO_ | SI_ | CSn_;
    // Enable push-pull output at port 0

    P2MDIN = (~0x01);
    // set P2.0 as analog input

    POSKIP = GD00_ | GD02_;
    // Crossbar skips the selected bits,
    // used as analog input/output

    XBR0 = BM_SPI0E;
    // SPI peripheral selected
}

```

```

XBR1 = BM_XBAR | BM_PCAOME_1 | BM_WEAKPUD;
// Enable crossbar, CEX0 routed to
// port pin and disable weak pull-ups
}

```

### Ακολουθούν οι συναρτήσεις σχετικές με τον Timer2

Πρόκειται για αρχικοποίηση του 16-bit Timer2. Όπως σε κάθε χρονιστή (timer), η τιμή του αυξάνεται κατά ένα, σε κάθε περίοδο του ρολογιού που πυροδοτεί τον συγκεκριμένο χρονιστή. Ο προγραμματιστής ορίζει το ρολόι του χρονιστή, την αρχική του τιμή και την τιμή που 'φορτώνεται' στον χρονιστή, όταν ο καταχωρητής του υπερχειλίσει, δηλ. όταν ο χρονιστής φτάσει την τιμή 0xFFFF και περάσει στην τιμή 0x0000.

Με την εντολή

```
CKCON &= ~0x30;
```

το ρολόι του χρονιστή ορίζεται με βάση το bit T2XCLK  $\equiv$  TMR2CN.0, το οποίο είναι αρχικοποιημένο σε τιμή '0', η οποία σύμφωνα με το εγχειρίδιο, ορίζει το ρολόι του χρονιστη 2, στην συχνότητα του system clock διαιρεμένου δια 12. Οι υπόλοιπες εντολές εξηγούνται στον κάδικα:

```

void Timer2_Init (int counts)
{
    TMR2CN = 0x00;
    // Stop Timer2; Clear TF2;

    CKCON &= ~0x30;
    // Timer2 clocked based on T2XCLK;
    // use SYSCLK/12 as timebase

    TMR2RL = -counts;
    // Init reload values

    TMR2 = 0xffff;
    // Set to reload immediately

    ET2 = 1;
    // Enable Timer2 interrupts

    TR2 = 1;
    // Start Timer2
}

```

Στην ρουτίνα χειρισμού του interrupt που προκαλεί ο χρονιστής 2, μηδενίζεται το interrupt flag και μεταβάλλεται η κατάσταση και των δύο LEDS.

```
INTERRUPT(Timer2_ISR, INTERRUPT_TIMER2)
{
    TF2H = 0;           // Clear Timer2 interrupt flag
    LED = !LED;         // Change state of LED
    LED2 = !LED2;       // Change state of LED2
}
```

Ακολουθεί η συνάρτηση `halwait`

Η συνάρτηση halWait χρησιμοποιείται για αναμονή τόσων με ΠΕΡΙΠΟΥ, όσων είναι το όρισμα της συνάρτησης (timeout).

Ακολουθεί η συνάρτηση `spi_wait`

Η συνάρτηση περιμένει interrupt από το SPI. Μόλις έρθει interrupt, μηδενίζει το interrupt flag και επιστρέφει. Υπενθυμίζεται πως το συγκεκριμένο flag παράγεται όταν ολοκληρωθεί μεταφορά δεδομένων από ή προς το ραδιόφωνο.

```
void spi_wait(void) {
    do {
        while (!SPIF);
        SPIF=0;
    } while (0);
}
```

### **Ακολουθεί η συνάρτηση halSpiWriteReg()**

Αρχικά η CSn γραμμή τίθεται από τον (master) μικροελεγκτή σε λογικό '0' (μέσω του σήματος NSSMDO) και στην συνέχεια, ο μικροελεγκτής περιμένει μέχρι το ραδιόφωνο αποκριθεί με 'μηδενισμό' της γραμμής MISO. Στην συνέχεια, ο μικροελεγκτής στέλνει διαδοχικά την τιμή της διεύθυνσης του καταχωρητή του ραδιοφώνου και στην συνέχεια την τιμή byte που θέλει να εγγράψει στον καταχωρητή αυτό. Η επικοινωνία ολοκληρώνεται με λογική τιμή '1' στην γραμμή CSn.

```
void halSpiWriteReg(BYTE addr, BYTE value) {
    NSSMDO = 0;
    while (P0_1);
    SPIODAT = addr;
    spi_wait();
    SPIODAT = value;
    spi_wait();
    NSSMDO = 1;
}
```

### **Ακολουθεί η συνάρτηση halRfWriteRfSettings()**

Η συνάρτηση χρησιμοποιεί την halSpiWriteReg που ορίσαμε παραπάνω για να αναθέσει στο ραδιόφωνο τις επιθυμητές ρυθμίσεις στους configuration registers. Ως όρισμα δέχεται έναν δείκτη (pointer) ο οποίος δείχνει στον πίνακα με τις τιμές bytes που περιέχουν τις ρυθμίσεις.

```
void halRfWriteRfSettings(RF_SETTINGS *pRfSettings) {

    // Write register settings
    halSpiWriteReg(CCxxx0_FSCTRL1, pRfSettings->FSCTRL1);
    halSpiWriteReg(CCxxx0_FSCTRL0, pRfSettings->FSCTRL0);
    halSpiWriteReg(CCxxx0_FREQ2, pRfSettings->FREQ2);
    halSpiWriteReg(CCxxx0_FREQ1, pRfSettings->FREQ1);
    halSpiWriteReg(CCxxx0_FREQ0, pRfSettings->FREQ0);
    halSpiWriteReg(CCxxx0_MDMCFG4, pRfSettings->MDMCFG4);
    halSpiWriteReg(CCxxx0_MDMCFG3, pRfSettings->MDMCFG3);
    halSpiWriteReg(CCxxx0_MDMCFG2, pRfSettings->MDMCFG2);
    halSpiWriteReg(CCxxx0_MDMCFG1, pRfSettings->MDMCFG1);
    halSpiWriteReg(CCxxx0_MDMCFG0, pRfSettings->MDMCFG0);
    halSpiWriteReg(CCxxx0_CHANR, pRfSettings->CHANR);
    halSpiWriteReg(CCxxx0_DEVIATN, pRfSettings->DEVIATN);
    halSpiWriteReg(CCxxx0_FREND1, pRfSettings->FREND1);
    halSpiWriteReg(CCxxx0_FREND0, pRfSettings->FREND0);
    halSpiWriteReg(CCxxx0_MCSMO , pRfSettings->MCSMO );
    halSpiWriteReg(CCxxx0_FOCCFG, pRfSettings->FOCCFG);
    halSpiWriteReg(CCxxx0_BSCFG, pRfSettings->BSCFG);
    halSpiWriteReg(CCxxx0_AGCTRL2, pRfSettings->AGCCTRL2);
    halSpiWriteReg(CCxxx0_AGCTRL1, pRfSettings->AGCCTRL1);
```

```

    halSpiWriteReg(CCxxx0_AGCTRL0, pRfSettings->AGCCTRL0);
    halSpiWriteReg(CCxxx0_FSCAL3, pRfSettings->FSCAL3);
    halSpiWriteReg(CCxxx0_FSCAL2, pRfSettings->FSCAL2);
    halSpiWriteReg(CCxxx0_FSCAL1, pRfSettings->FSCAL1);
    halSpiWriteReg(CCxxx0_FSCAL0, pRfSettings->FSCAL0);
    halSpiWriteReg(CCxxx0_FTEST, pRfSettings->FTEST);
    halSpiWriteReg(CCxxx0_TEST2, pRfSettings->TEST2);
    halSpiWriteReg(CCxxx0_TEST1, pRfSettings->TEST1);
    halSpiWriteReg(CCxxx0_TEST0, pRfSettings->TEST0);
    halSpiWriteReg(CCxxx0_FIFOTH, pRfSettings->FIFOTH);
    halSpiWriteReg(CCxxx0_IOCNG2, pRfSettings->IOCNG2);
    halSpiWriteReg(CCxxx0_IOCNG0, pRfSettings->IOCNG0);
    halSpiWriteReg(CCxxx0_PKTCTRL1, pRfSettings->PKTCTRL1);
    halSpiWriteReg(CCxxx0_PKTCTRL0, pRfSettings->PKTCTRL0);
    halSpiWriteReg(CCxxx0_ADDR, pRfSettings->ADDR);
    halSpiWriteReg(CCxxx0_PKTLEN, pRfSettings->PKTLEN);
}

```

### Ακολουθεί η συνάρτηση `halRfSendPacket()`

Η συνάρτηση αυτή χρησιμοποιείται για την μετάδοση ενός πακέτου με μήκος μέχρι 63 bytes. Ο μικροελεγκτής θα πρέπει να στείλει τα δεδομένα προς ασύρματη μετάδοση στο ραδιόφωνο, και στην συνέχεια να περιμένει όσο χρειάζεται για να μεταδοθούν ασύρματα τα δεδομένα από το ραδιόφωνο. Το σήμα που μεταδίδεται από το ραδιόφωνο στο pin GDO0, παίρνει την λογική τιμή '1' όταν το ραδιόφωνο στείλει την λέξη συγχρονισμού (sync word) και παραμένει εκεί μέχρι να σταλούν ασύρματα όλα τα δεδομένα. Τότε, το ραδιόφωνο θέτει την τιμή του GDO0 σε λογικό '0'. Σημειώνεται πως η εντολή (command strobe) CCxxx0-STX θέτει το ραδιόφωνο σε κατάσταση 'TX'.

```

void halRfSendPacket(BYTE *txBuffer, UINT8 size) {
    halSpiWriteBurstReg(CCxxx0_TXFIFO, txBuffer, size);
    halSpiStrobe(CCxxx0_STX);

    // Wait for GDO0 to be set -> sync transmitted
    while (!GDO0_PIN);

    // Wait for GDO0 to be cleared -> end of packet
    while (GDO0_PIN);
}

```

### Ακολουθεί η συνάρτηση `halSpiWriteBurstReg()`

Η συνάρτηση αυτή γράφει σε πολλαπλούς καταχωρητές, χρησιμοποιώντας burst— πρόσβαση στο SPI. Ως ορίσματα δέχεται την διεύθυνση του πρώτου CCxxx0 καταχωρητή που πρόκειται να προσπελαστεί, ένα δείκτη πίνακα ο οποίος περιέχει τα bytes που θα γραφτούν σε διαδοχικές διεύθυνσεις και τέλος, το πλήθος των bytes που θα γραφτούν ακολούθως στους CCxxx0 καταχωρητές του ραδιοφώνου. Η `halSpiWriteBurstReg` παρουσιάζει ομοιότητες στην λειτουργία της με την `halSpiWriteReg`, μόνο που εδώ γίνεται burst εγγραφή στους καταχωρητές.

```

void halSpiWriteBurstReg(BYTE addr, BYTE *buffer, BYTE count) {
    UINT8 i;
    NSSMDO = 0;
    while (P0_1);
    SPIODAT = addr | WRITE_BURST;
    spi_wait();
    for (i = 0; i < count; i++) {
        SPIODAT = buffer[i];
        spi_wait();
    }
    NSSMDO = 1;
}

```

### Ακολουθεί η συνάρτηση halSpiStrobe()

Η συνάρτηση αυτή δέχεται ως όρισμα μία εντολή και την γράφει στο CCxxx0 καταχωρητή. Λειτουργεί και αυτή όμοια με την halSpiWriteReg μόνο που στον καταχωρητή δεδομένων εγγράφεται ένα command strobe.

```

void halSpiStrobe(BYTE strobe) {
    NSSMDO = 0;
    while (P0_1);
    SPIODAT = strobe;
    spi_wait();
    NSSMDO = 1;
}

```

### Ακολουθεί η συνάρτηση halSpiReadStatus()

Η συνάρτηση αυτή δέχεται ως όρισμα την διεύθυνση του καταχωρητή κατάστασης (status register) που πρόκειται να προσπελαστεί. Στην συνέχεια διαβάζει και επιστρέφει την τιμή του καταχωρητή. Ας σημειωθεί πως επιστρέφεται ένα byte, αν και η τιμή του burst bit τίθεται σε λογικό '1' (γιατί;).

```

BYTE halSpiReadStatus(BYTE addr) {
    UINT8 x;
    NSSMDO = 0;
    while (P0_1);
    SPIODAT = (addr | READ_BURST);
    SPI_WAIT();
    SPIODAT = 0;
    SPI_WAIT();
    x = SPIODAT;
    NSSMDO = 1;
    return x;
}

```

### **Ακολουθεί η συνάρτηση halSpiReadReg()**

Η συνάρτηση αυτή δέχεται ως όρισμα την διεύθυνση του καταχωρητή CCxxx0 που πρόκειται να προσπελαστεί. Στην συνέχεια διαβάζει και επιστρέφει την τιμή του καταχωρητή.

```
BYTE halSpiReadReg(BYTE addr) {
    UINT8 x;
    NSSMDO = 0;
    while (P0_1);
    SPIODAT = (addr | READ_SINGLE);
    SPI_WAIT();
    SPIODAT = 0;
    SPI_WAIT();
    x = SPIODAT;
    NSSMDO = 1;
    return x;
}
```

### **Ακολουθεί η συνάρτηση halSpiReadBurstReg()**

Η συνάρτηση αυτή διαβάζει πολλαπλές διευθύνσεις, χρησιμοποιώντας burst πρόσβαση στο SPI. Ως ορίσματα δέχεται την διεύθυνση του πρώτου καταχωρητή που πρόκειται να προσπελαστεί, έναν δείκτη (pointer) προς ένα πίνακα από bytes στον οποίο θα καταχωρηθούν οι τιμές που θα διαβαστούν, και τέλος, το πλήθος των bytes που θα διαβαστούν. Η halSpiReadBurstReg μοιάζει στην λειτουργία της με την halSpiReadReg μόνο που εδώ γίνεται burst πρόσβαση στους καταχωρητές.

```
void halSpiReadBurstReg(BYTE addr, BYTE *buffer, BYTE count) {
    UINT8 i;
    NSSMDO = 0;
    while (P0_1);
    SPIODAT = (addr | READ_BURST);
    SPI_WAIT();
    for (i = 0; i < count; i++) {
        SPIODAT = 0;
        SPI_WAIT();
        buffer[i] = SPIODAT;
    }
    NSSMDO = 1;
}
```

### **Ακολουθεί η συνάρτηση halRfReceivePacket()**

Η συνάρτηση αυτή δέχεται ως ορίσματα έναν δείκτη για τον buffer στον οποίο θα αποθηκευτούν τα εισερχόμενα δεδομένα και έναν δείκτη σε μία μεταβλητή που περιέχει το μέγεθος του buffer. Το μήκος κάθε πακέτου δεν θα πρέπει να ξεπερνά το μέγεθος της στοίβας RX FIFO. Για να χρησιμοποιήσουμε την παραπάνω συνάρτηση θα πρέπει να εκμεταλευτούμε το σήμα GDO0, το οποίο τίθεται σε λογικό '1' όταν έχει ληφθεί το sync word και μηδενίζεται (λογικό '0'), όταν ολοκληρωθεί

η λήψη του πακέτου.

Επιπλέον το APPEND\_STATUS στον καταχωρητή PKTCTRL1 πρέπει να είναι ενεργοποιημένο. Η συνάρτηση υλοποιεί polling του GDO0. Αρχικά περιμένει για τον ορισμό του και στην συνέχεια για τον εκκαθαρισμό του. Αφότου εκκαθαριστεί το pin GDO0 ο καταχωρητής RXBYTES διαβάζεται για να επιβεβαιωθεί πως υπάρχουν εγγεγραμμένα bytes στην FIFO. Αυτό συμβαίνει επειδή το σήμα GDO0 θα υποδείξει οτι το sync word έχει ληφθεί ακόμη και εάν η FIFO έχει εκκαθαριστεί. Η συνάρτηση επιστρέφει true εάν το Cyclic Redundancy Check - CRC δεν ανακαλύψει λάθη, και false εάν όχι ή εάν δεν είχε τοποθετηθεί κανένα πακέτο στην στοίβα RX FIFO.

```
BOOL halRfReceivePacket(BYTE *rxBuffer, UINT8 *length) {
    BYTE status[2];
    UINT8 packetLength;

    halSpiStrobe(CCxxx0_SRX);

    // Wait for GD00 to be set
    // -> sync received
    while (!GD00_PIN);

    // Wait for GD00 to be cleared
    //-> end of packet
    while (GD00_PIN);

    // This status register is safe to read
    //since it will not be updated after
    // the packet has been received
    //(See the CC1100 and 2500 Errata Note)
    if ((halSpiReadStatus(CCxxx0_RXBYTES) & BYTES_IN_RXFIFO)) {

        // Read length byte
        packetLength = halSpiReadReg(CCxxx0_RXFIFO);

        // Read data from RX FIFO and store in rxBuffer
        if (packetLength <= *length) {
            halSpiReadBurstReg(CCxxx0_RXFIFO, rxBuffer, packetLength);
            *length = packetLength;

        // Read the 2 appended status bytes
        // (status[0] = RSSI, status[1] = LQI)
            halSpiReadBurstReg(CCxxx0_RXFIFO, status, 2);

        // MSB of LQI is the CRC_OK bit
        return (status[LQI] & CRC_OK);
    }
}
```

```
    } else {
        *length = packetLength;
        // Make sure that the radio is in IDLE
        // state before flushing the FIFO
        // (Unless RXOFF_MODE has been changed,
        // the radio should be in IDLE state at this point)
        halSpiStrobe(CCxxx0_SIDLE);
        // Flush RX FIFO
        halSpiStrobe(CCxxx0_SFRX);
        return FALSE;
    }
} else
    return FALSE;
}
```

### 3.2 Κώδικας AB\_rx.c: διαφορές με AB\_tx.c

Η διαφορά του κώδικα AB\_rx.c ο οποίος υλοποιεί την λήψη πακέτων σε σύγκριση με το κώδικα AB\_tx τον οποίο περιγράφαμε αναλυτικά παραπάνω είναι μόνο στο κομμάτι της συνάρτησης main.

```
void main (void) {

    UINT32 packetsSent = 0;
    UINT8 length;

    CLOCK_INIT();      // Initialize clock
    PORT_Init ();       // Initialize crossbar and GPIO
    SPI_INIT(SCLK_6_MHZ); // Initialize SPI

    NSSMDO = 1;    // you need to wait
                   // ~41 usecs, before CC2500 responds
    halWait(1);
    NSSMDO = 0;
    halWait(1);
    NSSMDO = 1;
    halWait(41);

    //now reset the radio;
    do {
        NSSMDO = 0;
        while (P0_1);
        //this is necessary for
        //the specific cc2500 radio
        //see cc2500 manual page 21
        SPIODAT = CCxxx0_SRES;
        //reset cc2500 chip (see cc2500.h)
        spi_wait();
        //wait until data have been tranmitted
        //through the spi interface
        NSSMDO = 1;
        //transmission complete
        //inform slave node (i.e. the cc2500)
    } while (0);

    halRfWriteRfSettings(&rfSettings);
    halSpiWriteReg(CCxxx0_PATABLE, paTable);

    EA = 1; // Enable global interrupts
    while (1){
```

```

////RX mode
length = sizeof(rxBuffer);
if (halRfReceivePacket(rxBuffer, &length)) {
    intToAscii(++packetsReceived);

    if (rxBuffer[10] == 10) {
        LED = ~LED;
    }

    mytestbyte = rxBuffer[0];
    mytestbyte = rxBuffer[1];
    mytestbyte = rxBuffer[2];
    mytestbyte = rxBuffer[3];
    mytestbyte = rxBuffer[4];
    mytestbyte = rxBuffer[5];
    mytestbyte = rxBuffer[6];
    mytestbyte = rxBuffer[7];
    mytestbyte = rxBuffer[8];
    mytestbyte = rxBuffer[9];

}

}

}

```

Παρατηρούμε ότι καλείται η συνάρτηση halRfReceivePacket(), σε αντίθεση με τον κώδικα για την εκπομπή πακέτων, όπου χρησιμοποιείται η συνάρτηση halRfSendPacket(). Η συνάρτηση halRfReceivePacket() λαμβάνει τα αποθηκευμένα σύμβολα στον txBuffer. Η υλοποίηση των δύο αυτών συναρτήσεων έχει περιγραφεί αναλυτικά στο προηγούμενο κεφάλαιο.

Στην συνέχεια αποθηκεύουμε στην μεταβλητή packetsSent το πλήθος των απεσταλμένων πακέτων πληροφορίας. Έλοποιούμε έναν βρόγχο ελέγχου if, με σκοπό να ελέγχουμε αν στην θέση 10 του buffer των λαμβανόμενων συμβόλων είναι το σύμβολο της αντίστοιχης θέσης στον buffer των απεσταλμένων συμβόλων. Αν ο έλεγχος είναι αληθής τότε το LED στο pin P2.2 αλλάζει κατάσταση.

Η λήψη πακέτων πληροφορίας, όπως περιγράφηκε παραπάνω, είναι συνεχής. Αυτό σημαίνει πως το en λόγω LED θα κάνει ουσιαστικά toggling κατά την ορθή λήψη πακέτων πληροφορίας.

### 3.3 Δομή πακέτων στον αέρα: επιπλέον bits προς μετάδοση, εκτός των bits πληροφορίας

Είναι σημαντικό να κατανοηθεί ότι για να μεταδοθούν συγκεκριμένα bits πληροφορίας, πρέπει να μεταδοθούν επιπλέον bits, τα οποία θα χρησιμοποιηθούν για συγχρονισμό πακέτου, για περιγραφή πακέτου και για ανίχνευση λαθών. Οι επιμέρους παράμετροι καθορίζονται από τον χρήστη και

καθορίζονται από τους καταχωρητές ρύθμισης (configuration registers). Η δομή του πακέτου προς εκπομπή/λήψη αποτελείται από τα ακόλουθα κομμάτια και φαίνεται παρακάτω:

- Preamble
- Synchronization word
- Μήκος byte ή σταθερό προγραμματιζόμενο μήκος πακέτου
- Προαιρετικό byte διεύθυνσης
- Payload
- Προαιρετικά 2 byte CRC

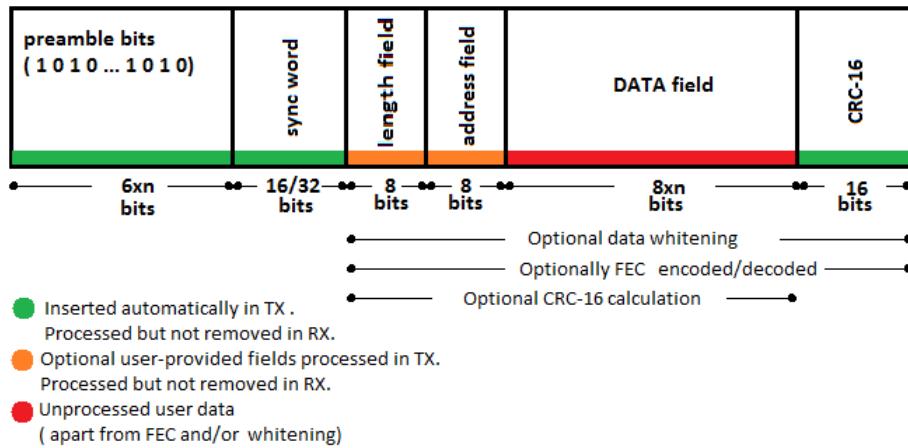


Figure 4: Δομή Πακέτου Λήψης.

Περισσότερες πληροφορίες στο εγχειρίδιο.

### 3.4 Εύκολη ρύθμιση ραδιοφώνου: Smart RF Studio

To Smart RF Studio Software συγκεκριμένο software προτείνεται ανεπιφύλακτα για την απόκτηση των βέλτιστων ρυθμίσεων των καταχωρητών του ραδιοφώνου μας (CC2500). Στο Fig. 3.4 παραθέτουμε ένα screenshot της Smart RF Studio διεπαφής:

Μετά από reset του ραδιοφώνου, όλοι οι καταχωρητές έχουν τις default τιμές τους. Οι βέλτιστες ρυθμίσεις των καταχωρητών μπορεί να διαφέρουν από τις default, και συνεπώς μετά από reset, οι καταχωρητές πρέπει να ρυθμιστούν. Το παραπάνω λογισμικό μας διευκολύνει στην κατάλληλη ρύθμιση των ραδιοφώνων και επομένως της ασύρματης ζεύξης, ανάλογα με την εφαρμογή.

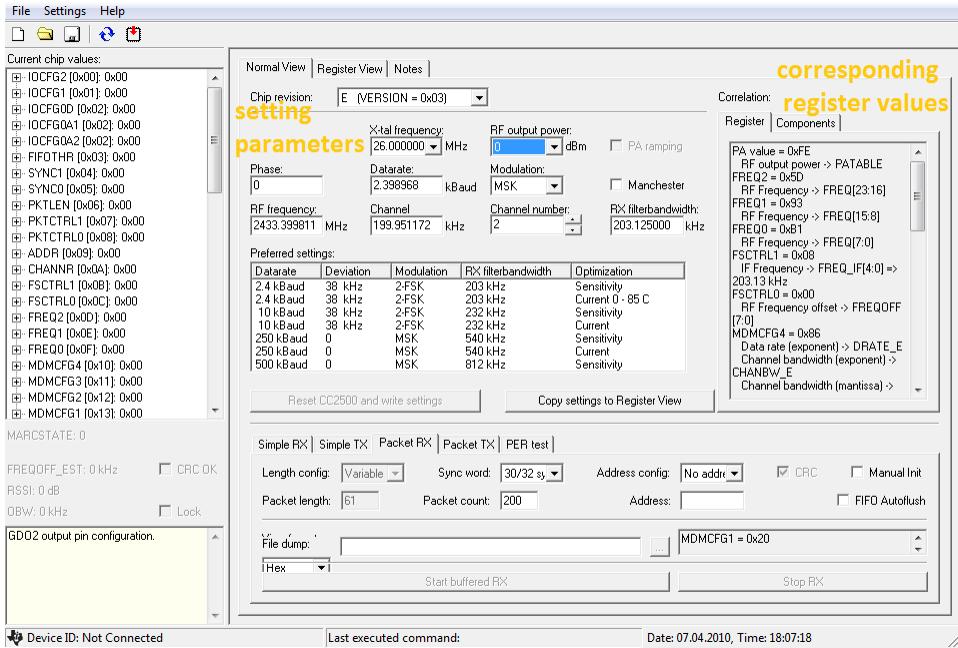


Figure 5: Εφαρμογή Smart RF Studio.

## 4 Ερωτήσεις

Οι παρακάτω ερωτήσεις απαιτούν αρκετή ενασχόληση σας με τον προγραμματισμό του μικροελεγκτή και του ραδιοφώνου, με βάση το λογισμικό που σας δόθηκε. Ξεκινήστε έγκαιρα!

1. Ως μηχανικός ψηφιακής ζεύξης, καλείστε να ρυθμίσετε την δομή και το μήκος του πακέτου που θα μεταδοθεί. Περιγράψτε την δομή, το μήκος και τα πεδία του πακέτου που θα χρησιμοποιήσετε και στην συνέχεια, τους σχετικούς καταχωρητές ρύθμισης που θα χρησιμοποιήσετε, με τις κατάλληλες τιμές. Προσοχή: εδώ μας ενδιαφέρουν οι καταχωρητές σχετικά με την δομή του πακέτου, και όχι όλοι οι καταχωρητές (π.χ. δεν μας ενδιαφέρει η διαμόρφωση, η συχνότητα ή η ισχύς εκπομπής).
2. Χρησιμοποιώντας την γνώση που αποκομίσατε από το παραπάνω ερώτημα, εξηγήστε την δομή του πακέτου στον δεδομένο κώδικα AB\_tx/rx.c.
3. Θέλετε η ζεύξη σας να λειτουργήσει στην μέγιστη ισχύ εκπομπής, σε ταχύτητα κάτω από 4 kilo-bits per second. Ποια είναι η μέγιστη εμβέλεια (απόσταση) της ζεύξης που πειραματικά παρατηρείτε; Περιγράψτε τους καταχωρητές ρύθμισης που χρησιμοποιείτε και φροντίστε η δυνατότητα Cyclic Redundancy Check (CRC) να είναι ενεργοποιημένη. Η επιλογή διαμόρφωσης αφήνεται στην διακριτική σας ευχέρεια. Προσέξτε να επιλέξετε το κανάλι που σας έχει δοθεί, έτσι ώστε να μην παρεμβάλλετε τους διπλανούς σας.
4. Επαναλάβετε το παραπάνω ερώτημα, χρησιμοποιώντας όμως την μέγιστη ταχύτητα μετάδοσης δεδομένων.

5. (Προαιρετικό - bonus) Δημιουργήστε νέο κώδικα, ώστε με το πάτημα ενός κουμπιού να αντιστρέψετε τους ρόλους σε κάθε πομπόδεκτη. Δηλαδή εάν ο ένας λειτουργεί σαν πομπός και ο άλλος σαν δέκτης, με το πάτημα του διακόπτη ο πρώτος θα πρέπει να λειτουργεί σαν δέκτης και ο τελευταίος σαν πομπός. Χρησιμοποιήστε LEDs για οπτικοποίηση της μεταφοράς πακέτων.
6. (Ping Pong) Υλοποιήστε την παραπάνω λειτουργικότητα, χωρίς πάτημα ενός κουμπιού: όταν ο δέκτης λάβει πακέτο, αλλάζει σε λειτουργία πομπού και μεταδίδει το πακέτο. Αντίστοιχα λειτουργεί και ο αρχικός πομπός, δηλ. αφού αποστείλει πακέτο, αλλάζει σε λειτουργία δέκτη και αναμένει πακέτο. Η μεταφορά και εναλλαγή ρόλων συνεχίζεται χωρίς διακοπή.
7. Η τιμή του Receiver Signal Strength Indicator (RSSI) είναι μία εκτίμηση του επιπέδου του σήματος στο επιλεγμένο κανάλι. Το ραδιόφωνο δίνει δυνατότητα RSSI. Στην ψηφιακή ζεύξη που σας δίνεται, ενεργοποιήστε την δυνατότητα αυτή στον δέκτη (αφού ανακαλύψετε την συγκεκριμένη λειτουργικότητα στο manual) και παρουσιάστε ένα ενδεικτικό demo της ορθής λειτουργίας του. [Ισως σας χρειαστεί η χρησιμοποίηση της σειριακής θύρας ή το αναβόσβημα ενός LED με συχνότητα ανάλογη του RSSI.]
8. (Προαιρετικό - bonus) Υλοποιήστε έναν ραδιο-τηλέγραφο με σήματα Morse, στην ψηφιακή σας ζεύξη!
9. (Συνεργασία) Μια σημαντική δυνατότητα των ραδιοφώνων που χρησιμοποιούμε είναι η εναλλαγή τους σε διαφορετικό κανάλι επικοινωνίας, χωρίς ιδιαίτερη καθυστέρηση, δηλ. το ραδιόφωνο μπορεί να λάβει σε μια συχνότητα και να μεταδώσει σε άλλη. Συνεργαστείτε με τον διπλανό πάγκο εργασίας, έτσι ώστε να ανταλλάξετε πακέτα επικοινωνίας (θυμηθείτε ότι ο κάθε πάγκος έχει το δικό του κανάλι επικοινωνίας, μεταξύ 0,2,...,10, στο λογισμικό που σας δόθηκε). Στην συνέχεια υλοποιήστε με συνεργασία όλων των πάγκων, μεταφορά πολλαπλών αλμάτων (multi-hop), ξεκινώντας από τον πάγκο 0 προς τον 2, στην συνέχεια στον 4 κ.ο.κ., μέχρι τον λήφη του πακέτου στον πάγκο 10.

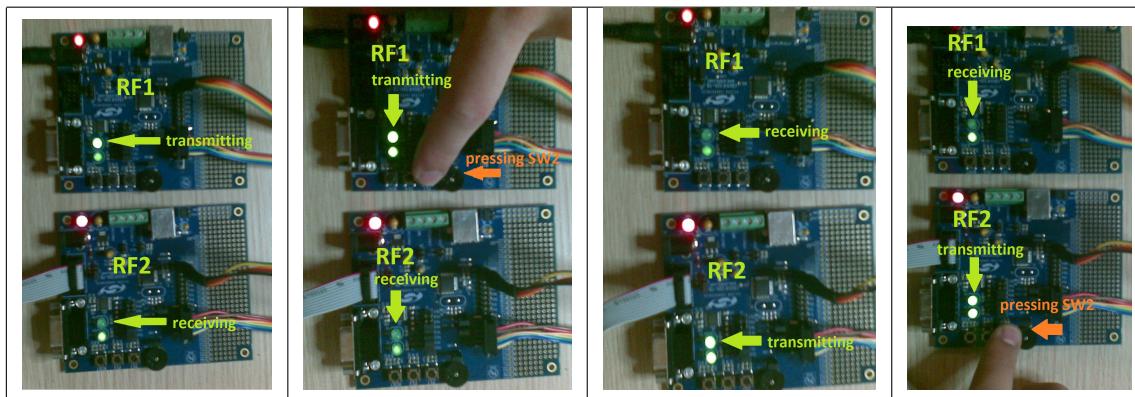


Figure 6: Σχηματικό διάγραμμα προτεινόμενης υλοποίησης