ABSA-NT

Constructeur d'équipements pour la formation électronique

ABSA-NT

WIRELESS – Hydra TP ZigBee

WIRLESS-HYDRA :

PLATEFORME TELECOMMUNICATIONS

Création d'une application Zigbee

Communication Zigbee entre 2 Plateformes Wireless-HYDRA

Sommaire

Travaux pratiques :

Application ZigBee sur plateforme Wireless-Hydra

Préambule

ZigBee est un protocole de communication de haut niveau permettant la communication de petites radios, à consommation réduite, basée sur la norme IEEE 802.15.4 pour les réseaux à dimension personnelle (Wireless Personal Area Networks : WPAN).

Ratifiées le 14 décembre 2004, les spécifications de ZigBee 1.0 sont disponibles auprès des membres de la communauté industrielle ZigBee Alliance.

Cette technologie a pour but la communication de courte distance telle que le propose déjà la technologie Bluetooth, tout en étant moins chère et plus simple. À titre d'exemple, les nœuds ZigBee classiques nécessitent environ 10% du code nécessaire à la mise en œuvre de nœuds Bluetooth ou de réseaux sans fil, et les nœuds ZigBee les plus élémentaires peuvent ainsi descendre jusqu'à 2%.

ZigBee est conçu pour interconnecter des unités embarquées autonomes comme des capteurs/actionneurs, à des unités de contrôle ou de commande. Des travaux sont également menés pour des applications domotiques sans fil (pilotage d'éclairage, commande de volets roulants, alarmes, maintenance des personnes à domicile…). Le monde industriel est depuis quelques temps vivement intéressé par ZigBee pour toutes les applications où un moyen de communication, sans fil bas débit et très basse consommation, est nécessaire. On va élaborer au cours de ce TP, une communication ZigBee entre 2 cartes permettant de gérer des Leds, envoyer un message ….

Introduction

Pilotée par un microcontrôleur PIC 32 et programmable en C et en C++, la carte Wireless – Hydra réalisée par ABSA-NT intègre les composants et modules de communication nécessaires permettant à l'utilisateur de se familiariser avec les principaux protocoles de communication.

Le but de ce TP est de permettre aux étudiants de créer des fonctions visant à mettre en place une communication ZigBee entre deux cartes possédant les modules MRF24J40.
Voici l'adresse du datasheet de ce composant
http://ww1.microchip.com/downloads/en/DeviceDoc/39776C.pdf

Cette dernière peut émettre, recevoir message, commander une autre carte et vice versa, voir l'état d'un capteur (capteur de température) ou d'un actionneur (bouton poussoir).

NB : Il faut impérativement lire et se munir du manuel MPLAB à chaque TP. Toute la partie création de projet et ouverture d'un projet existant se trouve dans le manuel

Le module utilisé est le MRF24J40.

Retrouver toutes les informations nécessaires au TP sur le site

http://www.absa-nt.com rubrique TP/MPLAB-Manuel-Wireless

BUT :

Ce TP va permettre aux étudiants de se familiariser avec la communication radio de type RX/TX (ZigBee)

Objectif

-Être capable de mettre en œuvre une communication sans fil entre les 2 cartes Wireless-Hydra.

Ce projet comporte deux types de fichiers :

- Un project.h (header) qui inclura les bibliothèques suivantes.
- "../../libs/lcd_lib/ssd1963.h" librairie permettant d'afficher quelques chose à l'écran.
- "../../libs/touch_lib/touch.h" librarie pour l'écran LCD.
- "../../libs/mrf24j60_lib/MRF24J40.h" librairie contenant tous les paramètres pour utiliser le module de communication ZigBee
- Un project.c qui inclura le project.h qui devra être capable d'écouter tous les paquets reçus et doit transmettre une réponse une fois que l'on va appuyer sur le bouton demander
- Le fichier zigbee.h qui sera donné contiendra ce qui suit : Ce sont toutes les bibliothèques, les structures et fonctions qui seront détaillées sur zigbee.c (où on va faire les fonctions de ce TP)

```
#ifndef ZIGBEE_H
#define ZIGBEE_H
#include "../../libs/lcd_lib/ssd1963.h"//lib permettant d'afficher à l'ecran LCD
#include "../../libs/touch_lib/touch.h"//Gère la position tactile du LCD
#include "../../libs/task_lib/task.h"//contient les structure des taches
#include "../../libs/menu_lib/application.h"//contient les structures des paramètres des bouton
//de la taille, de la position ...sur l'écran LCD
#include "../../libs/mrf24j60_lib/MRF24J40.h" // librairies faisant du module zigbee utilsé

#define TASK_ID_RXZIGBEE "task_rx_zb_id" // sturcture d'émission
#define TASK_ID_TXZIGBEE "task_tx_zg_id" // structure de reception

#define TASK_ID_WIFI "task_wifi_id" // structure du wifi

void ZBAppInit(struct Task * task); //fonction d'initialisatio de l'appli zigbee
void ZBAppRXTask(struct Task * task);//fonction qui gere la tache reception de message
void ZBAppTXTask(struct Task * task);//fonction qui gere la tache envoi de message
void ZBAppDestroy(struct Task * task);////fonction qui gere l'arret de l'application zigbee

void ZBApp(struct Button *butt);// fonction des boutons necessaires
void ZBExit(struct Button *butt);// fonction pour sortir de l'application

#endif  /* ZIGBEE_H */
```

Q1) fonction d'initialisation de la communication radio.

Faire une fonction permettant d'initialiser la communication radio. Cette fonction doit afficher un message de bienvenue, doit dire si le module ZigBee détecte une autre carte envoyant des messages radios et dois etre capable d'initialiser une adresse long(MyLongAddress), court(MyShortAddress) avec un ID(MyPANID) qui permettra donc de reconnaitre l'(es) émetteur(s) (maitre) et le(s) récepteur(s) (esclave) de la communication et permettra surtout de savoir si on détecte une communication radio venant du module de communication MRF24J40.

Pour résoudre cette application, vous pourrez suivre les étapes suivantes.

- Lire MRF24J40.c et comprendre le rôle des fonctions surtout ceux qui sont données pour élaborer la communication sur Le TP ZigBee. Le programme MRF24J40.c se trouve sur le projet **WirelessDevBoard.X**, plus particulièrement dans app/lib/

```
// cette fonction retourne un booléen qui est:

// =1 si communication radio allumé
// =0 si communication radio est eteint

BOOL RadioInit(void) // cold start radio init
{
    BOOL radio;

    memset((void*) &RadioStatus, 0, sizeof (RadioStatus));// remplissage des n pemier octet
    // de la zone memoire pointé par l'adresse de RadioStatus avec la taille de celui ci
    RadioStatus.MyPANID = MY_PAN_ID; // recupere l'ID
    RadioStatus.MyShortAddress = MY_SHORT_ADDRESS;// recupere l'adresse court
    RadioStatus.MyLongAddress = MY_LONG_ADDRESS;// recupere l'adresse long

    RadioStatus.Channel = 11; // commencement depuis channel 11

    radio = initMRF24J40();
    // intialisation du composant radio depuis RadioStatus.Channel
    //RFIE = 1;                                 // enable radio interrupts

    return radio;
}
```

- ssd1963_PutText pour afficher quelques choses à l'écran (fonction déjà expliquée dans MPLAB-Manuel-Wireless)
- SetPromtParameters (fonction déjà expliquée dans MPLAB-Manuel-Wireless)
- RadioInitP2P se trouvant dans MRF24J40.c, donnée et commentée ci-après

```
// INITs Tx point de structure simple à point à point ( per to per) entre une
// seule paire de périphériques qui utilisent la même adresse
// Après l'appel, vous pouvez envoyer des paquets en remplissant simplement :
// TxPayload [] avec la charge utile
// Tx.payloadLength ,
// Puis appelant RadioTXPacket ( )


void RadioInitP2P(void) {
    Tx.frameType = PACKET_TYPE_DATA;
    Tx.securityEnabled = 0;
    Tx.framePending = 0;
    Tx.ackRequest = 1;
    Tx.panIDcomp = 1;
    Tx.dstAddrMode = SHORT_ADDR_FIELD;
    Tx.frameVersion = 0;
    Tx.srcAddrMode = NO_ADDR_FIELD;
    Tx.dstPANID = RadioStatus.MyPANID;
    Tx.dstAddr = RadioStatus.MyShortAddress;
    Tx.payload = txPayload;
}
```

L'ossature globale de la fonction peut être comme suit :

(utilisez MRF24J40.c , ssd1963_PutText , SetPromtParameters, RadioInitP2P qui sont déjà données avant)

```
void ZBAppInit(struct Task * task) {


  if (  ) // condition qui montre que la communication
  {

/* à compléter pour faire apparaitre ce qui se trouve ci-après à l'écran lorsqu' on détecte une connexion  */

   "Demo for MRF24J40 is running."

   "Note this application shows you ALL received "

   "packets (not just ones addressed to you)."

   Long Addr:                    Short Addr :                    PAN ID:


    ssd1963_PutText(10, 70, zb_config, White, Black);

  } else

/*À compléter  pour avoir ceci s'il n'y a pas de connexion */

   "MRF24J40 is not detected."


  RadioInitP2P(); // setup for simple peer-to-peer communication

}
```

Q2) Bouton Exit

Rajouter un bouton exit qui devra être affiché après avoir lancé l'application Zigbee et qui permettra donc de quitter l'application.

Q3) Communication et commande à distance

Faire une fonction qui aura 2 parties :

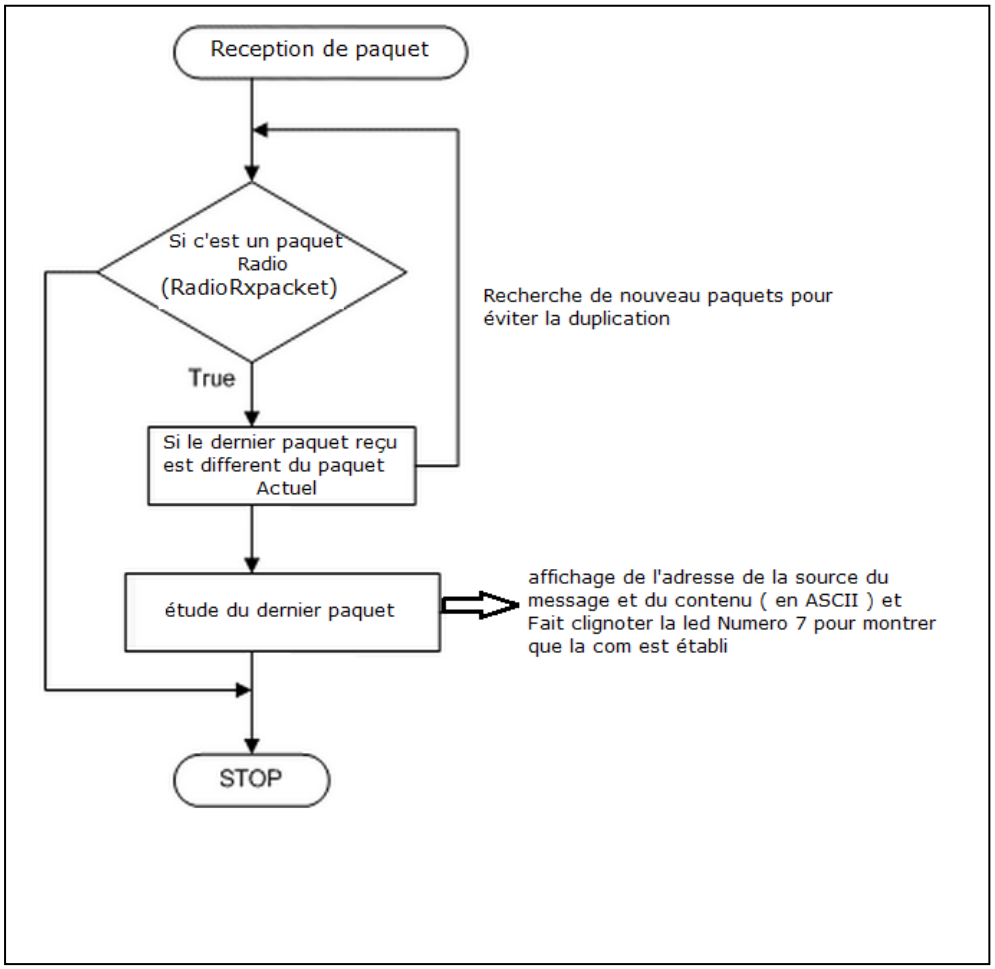
- Une partie communication : cette partie servira à capter les communications radio environnante, permettra ensuite d'afficher l'identifiant de l'émetteur et fera clignoter la LED7 si la communication entre les 2 cartes est établie.
- Une partie commande : Ici affiche l'état (appuyé ou pas) d'un bouton cliqué par l'utilisateur depuis le module émetteur dans le module récepteur et qui allumera la LED correspondante sur le récepteur

NB : on ne passera à la partie 2 qu'une fois avoir fini la première.

Ossature de la méthode : Envoi, test, étude du message reçu. Partie 1

```
void ZBAppRXTask(struct Task * task) {

  static uint8_t lastFrameNumber;
```




```
} static char Button0PreviousState = 0;   // affiche l'état du bouton0 (appuyé ou pas)

  static char Button1PreviousState = 0;

  static char Button2PreviousState = 0;

  static char Button3PreviousState = 0;
```

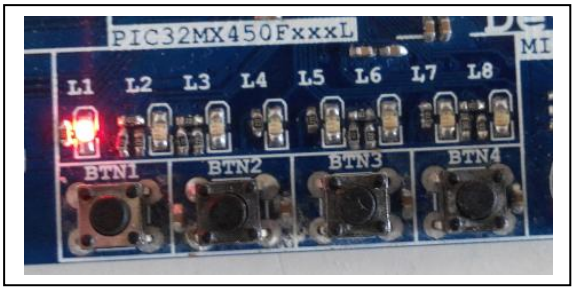
Partie 2 : (commande) Allumage de la LED n de la carte Wireless esclave, si on appuie sur BTN n de la carte Wireless Maitre.

Code à compléter pour allumer la LED1 de la carte esclave et afficher l'état de BTN1 de la carte maitre:

```
if (ButtonsGetState(0)) {
    if (   ) {
      Émetteur sending : Button1
      From address AA55 : Button1
        DelayUS(10000);
    }
    Button0PreviousState = 1;
    return;
  }
```

Exemple allume LED1 de la carte esclave si on appuie sur BTN1 de la carte Maitre

- Une fois la communication établie, on peut allumer la LED L1 du récepteur en appuyant sur BTN1 de l'émetteur par exemple.



Q4) Code à compléter pour commander toutes les LEDs et afficher l'état de tous les Boutons:

```
// envoi un message si on apue sur un bouton
   if (ButtonsGetState(0)) {
            if () {/ *à completer*/
               DelayUS(10000);
      }     /* à completer
      return;
   } else ;// bouton 1 est remise à zero
   if () {/ *à completer*/
         DelayUS(10000) ;
 if (  /* à completer*/ ) {
      }/* à completer
      return;
   } else bouton 2 est remise à zero
   if (  /* à completer*/ ) {
      if (  /* à completer*/)
 {/* à completer*/
  DelayUS(10000);
      }
 /* à completer*/
      return;
   } else
/* à completer*/
   } else  /* à completer*/ bouton 3 est remise à zero
}
.
.
.
```

Explications des librairies essentielles utilisées dans ce TP

Etant donné que le module ZigBee utilisé dans cette carte est le MRF24J40, on a mis tous les outils nécessaires à l'utilisation de ce module dans un programme appelée MRF24J40. On va aussi faire appel à la fonction spi.c. C'est dans ces programmes que l'on initialise toute la communication radio. On va détailler les fonctions mise en place dans le programme ci-après.

Il faut savoir que pour mettre en place cette communication, on va utiliser le SPI et deux structures RX et TX pour l'envoi et réception des trames radio.

La clé pour comprendre le SPI est qu'il ya seulement 1 ligne d'horloge, t donc tous les transferts sont toujours bidirectionnels. La taille maximale de la trame (envoyés et reçus) est un octet (8 bits). Le CLK (horloge) ne fonctionne (en mode Master) que lorsqu'on est en mode transmission.

On va utiliser le principe de PEPS (premier bit envoyé, premier bit sorti). Pour envoyer, vous stockez le message d'envoi dans la mémoire buffer TX, attendez que l'horloge s'actualise avant d'envoyer. Pour recevoir, il faut attendre que le dernier octet envoyé soit pointé pour pouvoir lire le message en entier.

Fonction mise en place pour le SPI

Envoi et réception de message :

1) unsigned char **spiPutGetByte**(unsigned char c) : Cette fonction écrit un octet à travers Le port de sorti du SPI et lit un octet à travers le port d'entrée du SPI. Le paramètre c donne l'octet à écrire à travers le port du SPI et la fonction retourne l'octet déjà lu.

```
unsigned char spiPutGetByte(unsigned char c) {
#if defined(SPI_SOFT)
    volatile unsigned char ret;
    volatile unsigned int mask;
    //SPI Mode 0. CS active à l'état bas. le clock fonctionne en frond montant.
    SOFT_SPI_CLK_LOW;

    //Activer la communication SPI . L'activation du signal SPI
    //doit être pulsé à l'état bas pour chaque octet envoyé !

    //Assurer un délai minimum de 500 ns entre
    //le front descendant du signal active du SPI
    // et le front montant de l'horloge du SPI
    //SPIDelay(SPI_BIT_DELAY);
    unsigned int delay = SPI_BIT_DELAY;
    while (delay--)
        asm("nop");
    mask = 0x80; //initialise et lit 7 bit
    ret = 0; //Initialise et litun octet avec 0
    do {
        //printf("%X ", mask);
        if (c & mask) SOFT_SPI_MOSI_HIGH;
        else SOFT_SPI_MOSI_LOW;
        //sorti de l'horloge du bit courant sur la ligne de sorti du SPI
        SOFT_SPI_CLK_HIGH; //met l'horloge du SPI à l'état haute
        //SPIDelay(SPI_BIT_DELAY);
```

```
        SOFT_SPI_CLK_HIGH; //met l'horloge du SPI à l'état haut //SPIDelay(SPI_BIT_DELAY);
        delay = SPI_BIT_DELAY;
        while (delay--)
            asm("nop");
        if (SOFT_SPI_MISO_READ) ret |= mask; //lit le bit courant
        //SPIDelay(SPI_BIT_DELAY); //Ensure minimum delay of 500nS between SPI Clock high and SPI Clock Low
        //Assurer un délai minimum de 500 ns entre le front montant du signal active et l' état bas de horlc
        delay = SPI_BIT_DELAY;
        while (delay--)
            asm("nop");
        SOFT_SPI_CLK_LOW; //met l'horloge du spi en route
        mask = mask >> 1; //Shift mask so that next bit is written and read from SPI lines
        //décale à droite le masque pour que le prochain bit écris et lu provient directement du ligne SPI
        //SPIDelay(SPI_BIT_DELAY); //Assure un délai minimum de 1000ns entre les bits
        delay = SPI_BIT_DELAY;
        while (delay--)
            asm("nop");
    } while (mask != 0);//Assure un délai minumum de 750ns entre les front descendant du signal d'horloge SF
    //et activer le front montant du SPI//Désactive la communication SPI
    return ret;// l'activation du signal SPI doit etre pulsé à l'état bas à cahque octet envoyer
#else
    //SpiChnGetRov(SPI_CHANNEL2, TRUE);
    SpiChnPutC(SPI_CHANNEL2, c); // envoi les données du maitre dans le canal de com
    return SpiChnGetC(SPI_CHANNEL2); // reçoi les données
#endif
}
```

Les fonctions suivantes existent dans le programme MRF24J40.c

- spiPut et spiGet : ses 2 fonctions servent simultanément à lire et à écrire un octet à travers le SPI. Il utilise tous les 2 spiPutGetByte déjà expliqué en haut.
- highRead : sert à lire les donnés radio des adresses long
- highWrite : sert à écrire les donnés radio des adresses long
- lowRead : sert à lire les donnés radio des adresses courts
- toTXfifo : écrit et compte les octets consécutifs de la source

avec le PEPS consécutifs à partir du «registre»

- initMRF24J40 : -active le module radio, récupère les canaux.

  -prend environ une période de 0.37 ms du PIC32 à 20 MHz et 10Mhz pour l'horloge du module SPI

- S'il retourne 0, cela signifie qu'il n'y a pas de radio ; Si c'est 1, la radio est remise à zéro

- RadioInit : Cette fonction retourne un booléen qui est:

  -Egal à 1 si communication radio allumé

  -Egal à 0 si communication radio est éteint

- RadioSetAddress : initialise l'adresse courte et la PANID
- RadioSetChannel : met en route un canal radio et retourne un booléen pour montrer s'il y' a un signal ou non. On y initialise aussi la bande passante permise qui se situe entre la fréquence 2400 MHz à 2483.5 MHz, la fréquence central e (2405+5(k-11) MHz, pour k=canal 11 à 26). Pour info, le canal 26 se trouve à la fréquence de 2480 MHz en pleine puissance
- RadioSetSleep : met l'émetteur-récepteur en marche/arrêt, initialise la Puissance radio, met RX:28.4 mA, TX: 65.8 mA ...
- RadioEnergyDetect : Fait un seul balayage de (128 us) du canal courant. Retourne la RSSI(puissance du signal)
- RadioTXRaw : Envoie un paquet brut déjà configuré par la structure Tx . on n'effectue aucune vérification d'erreur ici.
- RadioTXPacket : règle automatiquement l'adresse de réception du paquet
- RadioTXResult : retourne le statut du dernier paquet transmit.
- RadioWaitTXResult : Attente du statut de TX
- RadioRXPacket : Retourne et compte les paquets reçus, attend, traite et gère la pile pour bien recevoir le paquet
- CNZBInterrupt : Gestionnaire d'interruptions
- ZigBeeDisable : Désactive le module ZigBee après Initialisation