Licence Professionnelle SEICOM

Nom : Prénom :

M3-2 - Communications sans fil - WPAN - Zigbee

TP N°1

Table des matières

1 - Objectifs du TP N° 0

- Mettre en oeuvre les principaux éléments d'un réseau ZigBee.

Préparation préalable au TP : Cours/TD Communication sans fils - LR WPAN - 802.15.4 - ZigBee

Condition de réalisation :

- Travail en binôme
- Pour chaque étudiant: une station de travail Ubuntu
- Pour chaque binôme: deux afficheurs ZigBee, un coordinateur et un programmateur JTAG

2 - Caractéristiques de l'afficheur multifonction ZigBee

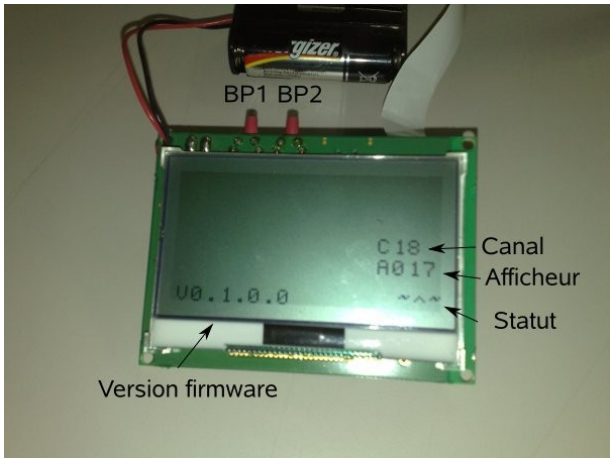


Photo afficheur coté écran LCD

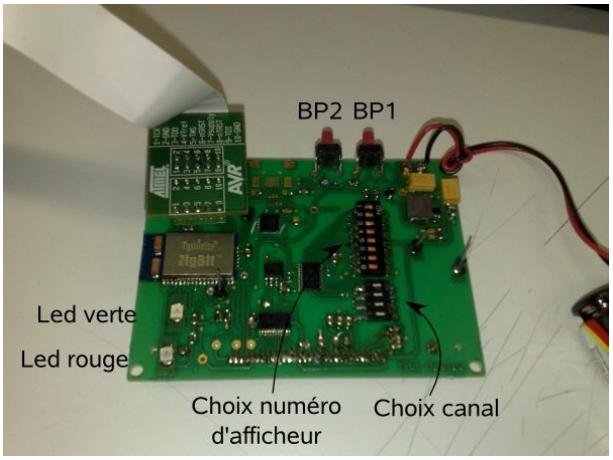


Photo afficheur coté composants

Les caractéristiques des modules du kit de développement Meshnetics qui serviront de coordinateurs sont données dans le TD.

3 - Mise en place de l'environnement de développement

Votre poste de travail est composé de :

- un ordinateur démarré sur Ubuntu,
- un programmateur JTAGICE MKII qui sera relié au PC par le port USB d'un côté, et au composant (afficheur ou coordinateur) qu'il faudra flasher de l'autre,
- un afficheur multifonction ZigBee développé par la société A3IP (www.a3ip.com),
- un coordinateur ZigBee, module du kit de développement ZigBee Meshnetics (www.meshnetics.com).

Sur votre Bureau, l'archive afficheur.tar.gz contient les fichiers nécessaires au TP (selon l'afficheur, il faudra utiliser le new_font ou old_font).

Décompressez l'archive et importez les fichiers dans le projet Afficheur.

Pour éditer les fichiers sources, compiler et flasher les composants ZigBit, vous pouvez utiliser :

– un IDE (eclipse, sublimeText, etc.)

ou

– éditer/compiler/flasher « à la main » avec gedit/make/avrdude (mise en oeuvre plus rapide)

Pour ce TP, on privilégiera la procédure à la main qui est la suivante :

- On édite avec gedit les fichier.c ou Makefile,

- ***make*** pour la compilation
- la commande ***sudo avrdude -V -c jtag2 -p m1281 -P usb -U flash:w:"afficheur.hex"*** pour flasher les composants ZigBit (AVR ATMEGA1281) avec le programmateur.
- Flashez l'afficheur.
- Pour allumez l'afficheur, appuyez quelques secondes sur BP1 jusqu'à ce que les deux leds s'allument puis s'éteignent.

✔ **Q3.1:** Que constatez-vous lorsque vous appuyez sur le bouton BP2 de l'afficheur ?

✔ **Q3.2:** Complétez le programme afficheur.c pour éteindre la led verte lorsque l'on relâche le bouton BP2.

✔ **Q3.2:** Que se passe-t'il si vous appuyez simultanément sur les boutons BP1 puis BP2 ?

4 - Configuration des paramètres réseau

- Configurez les afficheurs pour qu'ils aient le numéros indiqué par leur pastille et écoutent sur le canal suivant :

N° de coordinateur	***N° de canal***	***PanID***
1	11	0xAAAAAAAAAAAAAAAALL
2	13	0xABABABABABABABLL
3	15	0xBBBBBBBBBBBBBBLL
4	17	0xBCBCBCBCBCBCBCLL
5	19	0xCCCCCCCCCCCCCCLL
6	21	0xCDCDCDCDCDCDCDLL

Pour l'afficheur :

- N° afficheur = valeur micro-switch "afficheur" (8bits)
- N°canal = 11 + valeur micro-switch "canal" (4bits)
- PanID : réglé dans le Makefile

Pour le coordinateur :

- N° canal : réglé dans le Makefile
- PanID : réglé dans le Makefile

Statuts de l'afficheur :

Symbole Statut Afficheur	**Signification**
~^~	Recherche réseau
~~	Jonction réseau réussie
~>~	Émission en cours
~<~	Réception en cours
~Zz	sommeil
~UP	réveil

✔ **Q4.1:** Vérifiez que les paramètres affichés au démarrage de l'afficheur sont les bons.

✔ **Q4.2:** Quel est le statut de l'afficheur lorsqu'il n'y a pas de coordinateur dans l'environnement radio ?

5 - Connexion du coordinateur

- Branchez le coordinateur sur un port USB du PC en utilisant un cordon approprié.
- Sur votre Bureau, l'archive coordinateur.tar.gz contient les fichiers nécessaires au TP.
- Décompressez l'archive et importez les fichiers dans le projet Coordinateur.
- Configurez le Makefile avec les paramètres de votre réseau (voir §5).

Q5.1: Compilez et flasher le coordinateur. Vérifiez que vous obtenez le même fonctionnement qu'au paragraphe précédent.

- Lancez minicom (minicom -s) et configurez-le pour qu'il utilise le port USB sur lequel est connecté le coordinateur (*ls /dev/ttyUSB** ou *tail -f /var/log/syslog* | *grep usb* pour savoir quel port est utilisé lorsque l'on se connecte). Le coordinateur envoie régulièrement le nombre d'afficheurs connectés au réseau :

```
0.1.0.0
Coordinator started
Network started on channel : 18
nb children : 0
nb children : 0 ...
```

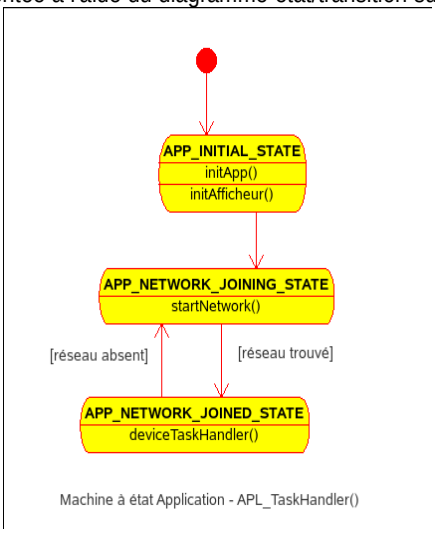
Messages du coordinateur sur la liaison série :

Message Coordinateur	Signification
nb children	nombre d'afficheur
!^ 9	jonction réseau réussie de l'afficheur 9
@9 - rssi:-63 - lqi:207	rssi (en dBm) et lqi (0-255) du afficheur 9
<~ @9 - req:1 - par:0 - rssi:-63	requête de l'afficheur 9 (requête 1 - paramètre 0 - rssi dBm)
~> @9	réponse envoyée vers l'afficheur 9
!~> OK	réponse correctement envoyée

- **Q5.2:** Vérifiez que les afficheurs rejoignent bien le réseau quand on les allume. Quel le statut affiché sur l'écran LCD lorsqu'un afficheur a rejoint le réseau ?
- **Q5.3:** Relevez et commentez les principales informations recueillies sur la liaison série du coordinateur (minicom) en fonction de la distance entre coordinateur et les afficheurs (il peut être nécessaire d'éteindre/allumer les afficheurs pour rafraichir plus rapidement les valeurs rssi/lqi).
- **Q5.4:** Que se passe-t'il sur l'afficheur lorsque le coordinateur est hors de portée radio ? Quelle sont les valeurs de rssi/lqi à la limite de portée radio ?
- **Q5.5:** Combien de temps, le coordinateur garde-t'il en mémoire la présence d'un afficheur sur le réseau (approximativement) ?

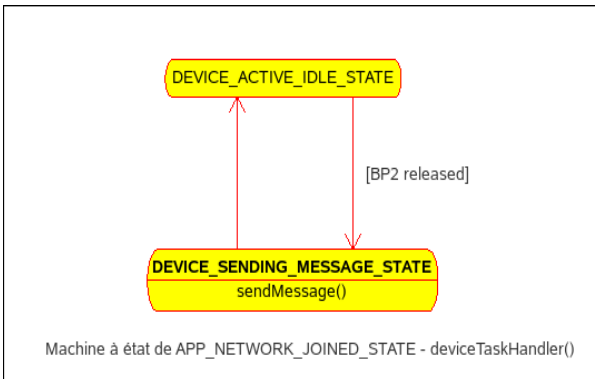
6 - Emission d'une requête depuis l'afficheur

La machine à état de l'afficheur peut se représentée à l'aide du diagramme état/transition suivant :



Dans l'état APP_NETWORK_JOINED_STATE, c'est à dire l'état nominal, le comportement de l'afficheur va suivre une autre machine à état. Cette machine à état imbriquée dans l'état APP_NETWORK_JOINED_STATE est plus ou moins complexe en fonction de ce que l'on demande à l'afficheur (émettre un message, recevoir une réponse, s'endormir, etc.).

On va commencer avec une machine à état simple : on souhaite que l'afficheur émette une requête de demande d'heure au coordinateur lorsque l'on relâche BP2.



Le message à transmettre est une structure composée de deux enregistrements : une requête et un paramètre pour cette requête.

```
typedef struct
{
    uint8_t requete;
    uint8_t param;
} PACK AppMessageOut_t;
```

Dans le cas d'une requête de demande d'heure on fixe la valeur de requête à 1 (REQ_HEURE) et param à 0 (PAR_HEURE).

On donne la fonction d'émission à compléter (sendMessage()) et son callback (APS_DataConf()).

```
/**************************************************************************
  Description:  Send the application message
  Parameters:   none
  Returns:      none
**************************************************************************/
static void sendMessage(void)
{
  // A COMPLETER
      GPIO_0_clr();

      appDataTtransmissionState = APP_DATA_TRANSMISSION_BUSY_STATE;
```

```
      sprintf(stringLCD, "   ");
      _lcdPutString(stringLCD,font_7x11,107,56);
      sprintf(stringLCD, "~>~");
      _lcdPutString(stringLCD,font_7x11,107,56);

      // A COMPLETER
      appMessageOutBuffer.message.requete = ;
      appMessageOutBuffer.message.param = ;

      // prepare and send APS Data Request
      apsDataReq.dstAddrMode          = APS_SHORT_ADDRESS;                    // Short addressing mode
      // A COMPLETER
      apsDataReq.dstAddress.shortAddress = ;            // Destination node short address

      apsDataReq.dstEndpoint          = APP_ENDPOINT;                         // Destination endpoint
      apsDataReq.profileId            = simpleDescriptor.AppProfileId;        // Profile ID
      apsDataReq.clusterId            = APP_CLUSTER_ID;                       // Destination cluster ID
      apsDataReq.srcEndpoint          = APP_ENDPOINT;                         // Source endpoint
      apsDataReq.asduLength           = sizeof (AppMessageOut_t);             // ASDU size
      apsDataReq.asdu                 = (uint8_t *) &appMessageOutBuffer.message;    // ASDU pointer as an application message
      apsDataReq.txOptions.acknowledgedTransmission = 0;                      // Acknowledged transmission enabled
      apsDataReq.radius               = 0;                                    // Default radius
      apsDataReq.APS_DataConf         = APS_DataConf;                         // Confirm handler
      APS_DataReq(&apsDataReq);                                               // Data Request sending
}

/*************************************************************************
  Description:  Data sent handler
  Parameters:   conf - APS Data Confirm primitive
  Returns:      none
*************************************************************************/
static void APS_DataConf(APS_DataConf_t *conf)
{
   GPIO_0_set();
   appDataTtransmissionState = APP_DATA_TRANSMISSION_IDLE_STATE; // Data transmission entity is idle
   switch (conf->status)
   {
   case APS_SUCCESS_STATUS :
      appDeviceState = appDeviceState = DEVICE_ACTIVE_IDLE_STATE;
      sprintf(stringLCD, "   ");
      _lcdPutString(stringLCD,font_7x11,107,56);
      sprintf(stringLCD, "~~~",conf->status);
      _lcdPutString(stringLCD,font_7x11,107,56);
      break;


   default:
      // List of status error code :
      // 0xE9 : MAC_NO_ACK_STATUS
      // 0xA3 : APS_ILLEGAL_REQUEST_STATUS

      sprintf(stringLCD, "   ");
      _lcdPutString(stringLCD,font_7x11,107,56);
      sprintf(stringLCD, "!%02X",conf->status);
      _lcdPutString(stringLCD,font_7x11,107,56);
      appDeviceState = DEVICE_SENDING_MESSAGE_STATE;    // Data transmission wasn't successfully finished. Retry.
      break;
   }

   SYS_PostTask(APL_TASK_ID);                   // Application task posting
}
```

- ✔ **Q6.1:** Copiez le code des fonctions sendMessage() et APS_DataConf(). Complétez la fonction sendMessage().
- ✔ **Q6.2:** Complétez la fonction buttonsReleased() pour qu'elle change l'état de la machine à état de l'afficheur (vers DEVICE_SENDING_MESSAGE_STATE) et rappelle le traitement de la machine à état, si l'afficheur est connecté au réseau et si il est dans l'état DEVICE_ACTIVE_IDLE_STATE.
- ✔ **Q6.3:** Compilez et flasher l'afficheur avec cette nouvelle version. Que constatez-vous sur la liaison série du Coordinateur ? Donnez la limite de portée radio approximative pour la communication sur le réseau (en dBm).

7 - Émission de la réponse depuis le coordinateur

On désire modifier le code du Coordinateur pour émettre la réponse à la requête REQ_HEURE en envoyant le nombre de milliseconde depuis la mise sous tension du coordinateur (voir fonction HAL_GetSystemTime()).

On donne la fonction d'émission à compléter (sendMessageTargetChild()) :

```
static void sendMessageTargetChild()
{
   GPIO_2_clr();
   GPIO_1_set();
   appCoordState = COORD_ACTIVE_IDLE_STATE;
   if (APP_DATA_TRANSMISSION_IDLE_STATE == appDataTtransmissionState)
   {
      appDataTtransmissionState = APP_DATA_TRANSMISSION_BUSY_STATE;

#ifdef _DBG_
      sprintf(strBuf, "~> @%d",targetChild);
      printComMsg(strBuf);
#endif

      // A COMPLETER
      appMessageOutBuffer.message.reponse = ;
      appMessageOutBuffer.message.date = ;

      apsDataReq.dstAddrMode          = APS_SHORT_ADDRESS;                    // Short addressing mode
      // A COMPLETER
      apsDataReq.dstAddress.shortAddress = ;

      apsDataReq.dstEndpoint          = APP_ENDPOINT;                  // Destination endpoint
      apsDataReq.profileId            = simpleDescriptor.AppProfileId;        // Profile ID
      apsDataReq.clusterId            = APP_CLUSTER_ID;                // Destination cluster ID
      apsDataReq.srcEndpoint          = APP_ENDPOINT;                  // Source endpoint
      apsDataReq.asduLength           = sizeof (AppMessageOut_t);             // ASDU size

      apsDataReq.asdu                 = (uint8_t *) &appMessageOutBuffer.message;     // ASDU pointer as an application message

      apsDataReq.txOptions.acknowledgedTransmission = 0;                  // Acknowledged transmission enabled
      apsDataReq.radius               = 0;                        // Default radius
      apsDataReq.APS_DataConf         = APS_DataConf;                  // Confirm handler

      APS_DataReq(&apsDataReq);
   }
}
```

Q7.2: Copiez et complétez le code de la fonction sendMessageTargetChild().

Q7.3: Complétez le code de la fonction APS_DataInd() chargée d'initialiser la variable globale targetChild avec l'adresse de l'afficheur qui a envoyé la requête REQ_HEURE si targetChild est égal à zéro (pas d'autre demande en cours).

Q7.4: Compilez et flasher le coordinateur. Vérifiez que le coordinateur reçoit bien la requête et renvoie une réponse.

8 - Réception et affichage de la date sur l'afficheur

Q8.1: Complétez la fonction de réception coté Afficheur pour afficher la date reçue du coordinateur.

Indication : Pour copier le message entrant vers la structure à afficher, vous pouvez utiliser

memcpy(&appMessageIn,indData→asdu,indData→asduLength);

Avec indData qui contient la valeur de la date reçue.

9 - Agrandir le réseau

Q9.1: Configurez correctement les éléments du réseau afin d'augmenter le nombre d'afficheurs sur le réseau en vous regroupant avec d'autres binômes.

10 - Améliorations

Q10.1: Proposez des améliorations pour endormir l'afficheur, le réveiller toutes les 30s ou lorsque l'on appuie sur BP2 pour rafraichir la date.