Notice sur le fonctionnement du dragino

Glossaire :

Fichier .c : « Un fichier .c est un fichier de code source écrit dans le langage de programmation C.Il contient des fonctions, des variables et d'autres constructions utilisées pour créer unprogramme qui peut être compilé en code machine pour l'exécution sur un ordinateur. Le fichier.c est généralement édité à l'aide d'un éditeur de texte ou d'un environnement dedéveloppement intégré (IDE) puis compilé à l'aide d'un compilateur C, tel que GCC, pourgénérer un fichier exécutable. »

Fichier .h : « Un fichier .h est un fichier d'en-tête en C. Il contient généralement des déclarationsde variables, de fonctions et de structures de données utilisées par d'autres fichiers source dansun projet. Les fichiers d'en-tête sont généralement inclus dans les fichiers source C à l'aide del'instruction #include. Les fichiers d'en-tête permettent de regrouper les déclarations communesdans un seul emplacement pour une utilisation répétée dans plusieurs fichiers source, ce quifacilite la maintenance et la modification du code.»

Driver:«Un "driver" (pilote en français) est un logiciel qui permet à un système d'exploitation de communiquer avec un périphérique matériel spécifique. Dans notre projet il s'agit defichier .c et .h qui permettentde communiquer et d'utiliser un type de sonde particulier.

Main.c:«Le fichier "main.c" est un fichier qui contient généralement la fonction principale, appelée "main", qui est le point d'entrée du programme. Lorsqu'un programme en C est exécuté, l'exécution commence par la fonction "main". Les autres fonctionset programmessont appelésà partir du main.

Introduction :

Ceci est une notice comprenant des informations sur le fonctionnement du dragino en vue d'une modification du code afin de modifier ou d'ajouter un mode de fonctionnement.

Cette notice montre les fichiers importants qui devront être modifié.

Les fichiers à modifier sont de types différents : le main, le bsp, les drivers et les commandes AT.

Main :

le main est le fichier principal du système. C'est lui qui va lancer toutes les fonctions contenues dans les autres fichiers. C'est le fichier le plus important pour le système. Dans notre cas, il nous a servis à initialiser les gain et offset des capteurs HC2A et HYT939. De plus c'est dans le main que se trouve la fonction qui place les données utiles dans la trame qui sera envoyé par protocole LoraWan sur TTN

main.c

```
851         AppData.Buff[i++] = (int)(sensor_data.hyt_sens[j].hum * 10);
852       }
853     }
854
855   else if (mode == 21)
856     {
857       AT_PRINTF("\r\nTEMPERATURE=%.2f  HUMIDITE =%.2f \r\n",sensor_data.hc2a_t.temp,sensor_data.hc2a_t.hum );
858       AppData.Buff[i++] = (batteryLevel_mV >> 8); // level of battery in mV
859       AppData.Buff[i++] = batteryLevel_mV & 0xFF;
860
861       AppData.Buff[i++] = 0x00;
862       AppData.Buff[i++] = 0x00;
863
864       AppData.Buff[i++] = 0x00;
865       AppData.Buff[i++] = 0x00;
866
867       AppData.Buff[i++] = (int)0x54;
868
869
870       AppData.Buff[i++] = (int)(sensor_data.hc2a_t.temp * 10) >> 8;
871       AppData.Buff[i++] = (int)(sensor_data.hc2a_t.temp * 10);
872       AppData.Buff[i++] = (int)(sensor_data.hc2a_t.hum * 10) >> 8;
873       AppData.Buff[i++] = (int)(sensor_data.hc2a_t.hum * 10);
874       //AppData.Buff[i++] = (int)(sensor_data.hc2a_t.gain);
875       //AppData.Buff[i++] = (int)(sensor_data.hc2a_t.offset);
876
877     }
```

Il appelle aussi les fonctions contenues dans le bsp afin d'actualiser les données

Bsp :

Le bsp est le fichiers qui va s'occuper de faire le lien entre le main et les drivers. Ce fichier sert à appeler les fonctions contenues dans les drivers pour récupérer les données des sondes et pour ensuite les envoyer.

La fonction bsp_sensor_init initialise les ports à utiliser et teste si des sondes sont connectés au dragino . La fonction bsp_sensor_read va utiliser les fonctions spécifiques au driver correspondant au mode sélectionné.

Le fichier bsp.h contient la structure des données des capteurs. Toutes les variables de tous les capteurs sont inclus dans cette structure. On peut également y insérer une structure provenant

d’un driver. Si vous souhaitez créer un nouveau driver il faudra déclarer les nouvelles variables dans la structure.

```
typedef struct{

  uint8_t   in1;/*GPIO Digital Input 0 or 1*/
  float temp1;//DS18B20-1
  float temp2;//DS18B20-2
  float temp3;//DS18B20-3
  float oil;  //oil float
  float ADC_1; //ADC1
  float ADC_2;  //ADC2
  float temp_sht;
  float hum_sht;
  uint16_t illuminance;
  uint16_t distance_mm;
  uint16_t distance_signal_strengh;
  int32_t Weight;
  hyt_sensor hyt_sens[10];
  hc2a_sensor  hc2a_t;
  /**more may be added*/
} sensor_t;
```

Dans le main, un sensor_data de type sensor_t est déclaré en tant que variable globale. Sensor_data contient les valeurs qui seront envoyé dans la trame Lora.

Driver :

Le driver est le fichier qui va utiliser les sondes. De manière générale, il contient toutes les fonctions relatives aux sondes associées. C’est le fichier le plus difficile à créer car on ne peut pas tester le fonctionnement d’une sonde sans lui. Dans le cas du mode 21, le driver utilise un port virtuel pour communiquer avec la sonde à l’instar du mode 2 avec le capteur de distance tf-mini plus.

C’est le fichier qui devrait vous poser le plus de problème lors du développement du projet.

Commande AT :

Les commandes AT sont les commandes qui vont permettre de communiquer avec le dragino. Le système ne possède pas de débugger donc le seul moyen d’agir sur le dragino est via les commandes AT. Ces commandes permettent d’afficher des valeurs ou de modifier des paramètres comme le mode de fonctionnement du système.

La partie des commandes est séparé en 3 fichiers : command.c , at.c et at.h qui contiennent respectivement les déclarations des commandes AT, les différentes fonctions liés à ces commandes et les explications des différentes fonctions. Ce sont les 3 fichiers à modifier pour ajouter des commandes AT

Une explication plus approfondie se trouve dans la documentation programmeur.

Exemple :

```
    {
    .string = AT_MOD,
    .size_string = sizeof(AT_MOD) - 1,
#ifndef NO_HELP
    .help_string = "AT"AT_MOD ":Get or Set the work mode\r\n",
#endif
    .get = at_MOD_get,
    .set = at_MOD_set,
    .run = at_return_error,
  },
```

Figure 1: déclaration de la commande AT lié au mode de fonctionnement (command.c)

```
ATEerror_t at_MOD_set(const char *param)
{
  uint8_t workmode;
  if (tiny_sscanf(param, "%d", &workmode) != 1)
  {
    return AT_PARAM_ERROR;
  }
  if ( (workmode>=1 && workmode<=9)|| workmode==20 || workmode==21 || workmode==42 )
  {
    mode=workmode;
    PPRINTF("Attention:Take effect after ATZ\r\n");
  }
  else
  {
    PPRINTF("Mode of range is 1 to 9 or 40 to 42 \r\n");
    return AT_PARAM_ERROR;
  }

  return AT_OK;
}

ATEerror_t at_MOD_get(const char *param)
{
  print_d(mode);
  return AT_OK;
}
```

Figure 2: fonction de lecture et d'écriture du mode (at.c)

```
#define AT_MOD          "+MOD"
```

Figure 3 : déclaration dans at.h

```
/**
  * @brief  Print temperature and humidity of hc2a sensor
  * @param  Param string of the AT command
  * @retval AT_OK if OK, or an appropriate AT_xxx error code
  */
ATEerror_t at_valhc2a_get(const char *param);
```

Figure 4 : information relatives à une fonction (at.h)