Centre professionnel du Nord vaudois
Filière d'automatique et d'électronique
Modules complémentaires techniques

CPNV

Manette arcade

P1704_Manettes

SANDOZ Pierre-Olivier
PICOTTE Alexandre
01/07/2018

Table des matières

Introduction :

Le code de la manette arcade se compose d'un code, qui gère les boutons appuyés (slave), à fournir aux manettes et d'un code à fournir à un arduino leonardo ou micro qui se charge de simuler un clavier lorsqu'il est branché à un ordinateur ou un raspberry (master). Les manettes ont le code qui gère les boutons, le code pour leonardo ou micro se charge de récupérer les données des manettes pour les convertir en touche de clavier pour pouvoir jouer à des jeux rétros, il s'agit de l'arduino de la « console ». La communication entre les arduinos se fait par I2C, par le biais de la libraire Wire.

Header et communication I2C :

La communication se fait par I2C et pour vérifier que les différents arduinos reçoivent bien des instructions ou des requêtes nous avons ajouté un header au début de chaque communication. Celui-ci intervient comme un mot de passe que les arduinos doivent donner pour transmettre des informations, un arduino qui reçoit une communication va vérifier si les deux premières bytes correspondent au header choisi, si oui alors l'arduino traite les données qu'il reçoit si non il ignore ces données.

```
Serial.println("data received");
while (0 < Wire.available()) // slave may send less than requested
{
  for (int i = 0; i < 2; i++)
  {
    test_header[i] = Wire.read();
  }
}
for (int j = 0 ; j < 2; j++)
{
  if (header[j] != test_header[j])
  {
    confirm_header = false;
  }
  if (confirm_header)
  {
    test_header[j] = 0b00000000;
  }
}

if (confirm_header)
{
  //executing instructions
}
```

Code Manette :

Recherche d'addresse :

Lors de sa mise sous tension, l'arduino, dans lequel on a téléversé le code qui gère les boutons, va chercher à obtenir une adresse auprès de son master l'arduino de la console, tant qu'il n'a pas d'adresse il va envoyer en permanence une demande par I2C, il y à alors trois cas possibles :

```
if (!redefine_adresse)
{
  Serial.println("debut recherche");
  Wire.requestFrom(leonardo, 3, true);
  Serial.println("fin recherche");
  delay(1000);
  while (1 < Wire.available()) //slave may send less than requested
  {
    for (int i = 0; i < 2; i++)
    {
      test_header[i] = Wire.read();
    }
  }
  for (int j = 0 ; j < 2; j++)
  {
    if (header[j] != test_header[j])
    {
      confirm_header = false;
    }
  }
  if (confirm_header)
  {
    while (Wire.available())
    {
      adresse = Wire.read();
    }
    configuration(adresse);
  }
  confirm_header = true;
}
```

1. Le master ne répond pas, dans ce cas le slave va continuer à effectuer des demandes d'adresse

2. Le master lui donne une adresse de joueurs actifs, soit 1 soit 2, le slave refait alors son setup avec sa nouvelle adresse

```
void configuration(int new_adresse)
{
  if (new_adresse <= 2)
  {
    Wire.begin(new_adresse);
    Wire.onReceive(receiveEvent); // register event
    Wire.onRequest(requestEvent);// request event
    redefine_adresse = true;
    waiting = false;
    player = true;
    Serial.print("adresse:");
    Serial.println(new_adresse);
    activate_pin();
  }
```

3. Le master lui donne l'adresse lorsqu'il y a trop de joueurs, le slave va alors arrêter de faire des demandes, Si une place se libère le master enverra un message au slave pour lui demander de refaire une demande d'adresse.

```
else
{
  Wire.begin(new_adresse);
  Wire.onReceive(receiveEvent); // register event
  Wire.onRequest(requestEvent);// request event
  redefine_adresse = true;
  waiting = true;
  player = false;
  Serial.println("en attente");
}
```

Ce n'est que lorsque le slave reçoit une adresse de joueurs actifs que le reste du programme va se lancer, notamment l'activation de ces inputs.

```
// active les inputs
void activate_pin()
{
  pinMode(bouton1, INPUT_PULLUP);
  pinMode(bouton2, INPUT_PULLUP);
  pinMode(bouton3, INPUT_PULLUP);
  pinMode(bouton4, INPUT_PULLUP);
  pinMode(bouton5, INPUT_PULLUP);
  pinMode(bouton6, INPUT_PULLUP);
  pinMode(bouton7, INPUT_PULLUP);
  pinMode(bouton8, INPUT_PULLUP);
  pinMode(bouton9, INPUT_PULLUP);
  pinMode(bouton10, INPUT_PULLUP);
  pinMode(bouton11, INPUT_PULLUP);
  pinMode(bouton12, INPUT_PULLUP);
}
```

Gestion des touches :

Lorsque l'arduino est un joueur actif, il va alors lancer le reste de son programme, qui consiste à attendre une demande du master pour regarder quelles sont ses boutons enclenchés.

```
// s'active lorsque l'arduino recoit une requete de son master
void requestEvent()
{
  button(bouton1,  0, four, not_four);
  button(bouton2,  0, three, not_three);
  button(bouton3,  0, two, not_two);
  button(bouton4,  0, one, not_one);
  button(bouton5,  1, eight, not_eight);
  button(bouton6,  1, seven, not_seven);
  button(bouton7,  1, six, not_six);
  button(bouton8,  1, five, not_five);
  button(bouton9,  1, four, not_four);
  button(bouton10, 1, three, not_three);
  button(bouton11, 1, two, not_two);
  button(bouton12, 1, one, not_one);
  //Serial.println(header[0]);
  //Serial.println(header[1]);
  Wire.write(header, 2);
  Wire.write(adresse);
  Wire.write(trame, 2);
}
```

Lors d'une demande le slave va regarder chaque input déclaré et regarder si l'input est LOW ou HIGH, il va ainsi composer deux bytes d'informations qui contient les états de tous les boutons.

```
static byte trame[2] = {0b00000000, 0b00000000};
```

On n'a que 12 boutons donc on utilise que 12 bits, dans notre il s'agit des bits entourés en orange.

A chaque bit de chaque byte est assigné un bouton et selon son état le bit vaut 0 ou 1, 1pour appuyé sinon 0. Cette gestion se fait au moyen de la fonction *button*.

```
// permet d'attribuer les bits selon l'état des boutons
void button(int bouton, int seq_trame, byte place, byte not_place)
{
  if (digitalRead(bouton) == LOW)
  {
    delay(10);
    Serial.println("button pressed");
    trame[seq_trame] = trame[seq_trame] | place;
  }
  else
  {
    delay(10);
    trame[seq_trame] = trame[seq_trame] & not_place;
  }
}
```

- bouton = quel bouton il faut regarder l'état
- seq_trame = s'il s'agit du premier ou deuxième byte de la trame
- place = aide pour mettre le bit désirer à 1 si bouton est LOW
- not_place = aide pour mettre le bit désirer à 0 si bouton est HIGH

```
// permet d'aider à remplir les bits de chaque bytes
#define zero  0b00000000
#define one   0b00000001
#define two   0b00000010
#define three 0b00000100
#define four  0b00001000
#define five  0b00010000
#define six   0b00100000
#define seven 0b01000000
#define eight 0b10000000

#define not_zero  0b11111111
#define not_one   0b11111110
#define not_two   0b11111101
#define not_three 0b11111011
#define not_four  0b11110111
#define not_five  0b11101111
#define not_six   0b11011111
#define not_seven 0b10111111
#define not_eight 0b01111111
```

Code console :

Le code console va s'occuper de gérer les joueurs, demander aux slaves l'état de leurs boutons ainsi que simuler un clavier et transmettre des touches de clavier selon l'état des boutons des joueurs

Attribution d'adresse :

Le code du master ne commence réellement que lorsqu'il enregistre au moins un joueur, donc lors de sa mise sous tension il va attendre une demande d'un slave pour lui transmettre une adresse, s'il y a déjà trop de joueurs alors le master va mettre les joueurs de trop en attente et lorsqu'une place se libère leur enverra un message pour leur demander de refaire leur demande d'adresse. Les joueurs actifs ont les adresse 1 et 2. L'attribution d'adresse passe par la fonction *requestEvent*.

```
void loop()
{
  if (nb_player > 0)
  {
    if (player_coming)
    {
      delay(5000);
      player_coming = false;
    }

    …
```

```
void requestEvent()
{
  Serial.println("request event");
  for (int i = 0; i < sizeof(Player) / sizeof(bool); i++)
  {
    if (!Player[i])
    {
      if (!is_empty)
      {
        slave_adresse = i + 1;
        is_empty = true;
        Player[i] = true;
        Wire.write(header, 2);
        Wire.write(slave_adresse);
        nb_player++;
        player_coming = true;
        //delay(10);
      }
    }
  }
  if (!is_empty)
  {
    Wire.write(header, 2);
    Wire.write(adresse_wait);
    player_waiting = true;
    player_coming = true;
  }
  //Serial.println(nb_player);
  is_empty = false;
}
```

Requête au slave :

Dès que le master a au moins un slave il va alors émettre des requêtes auprès du slave pour lui demander l'état de ses boutons. Le master va procéder par étapes il va d'abord regarder dans liste de joueurs actifs et envoyé une requête à chacun.

```
for (int r1 = 0; r1 < sizeof(Player) / sizeof(bool); r1++)
{
  if (Player[r1])
  {
    Wire.requestFrom(r1 + 1, 5);
    while (3 < Wire.available())
    {
      for (int i = 0; i < 2; i++)
      {
        test_header[i] = Wire.read();
        //Serial.println(test_header[i]);
      }
    }
```

Le master va ensuite vérifier si les deux premières bytes reçues correspondent au header choisi, si oui il va traiter les données reçues si non il va effacer le joueur de son registre de joueurs actifs et s'il y a des joueurs en attente leur enverra un message pour les prévenir qu'une place c'est libéré.

```
else if(player_waiting&&!confirm_header)
{
  Player[r1] = false;
  nb_player--;
  Serial.println("player absent");
  Serial.println(nb_player);
}
confirm_header = true;
if (player_waiting)
{
  Serial.println("player waiting");
  if (nb_player < sizeof(Player) / sizeof(bool))
  {
    Wire.beginTransmission(adresse_wait);
    Wire.write(header, 2);
    Wire.endTransmission();

    Serial.println(nb_player);
  }
  player_waiting = false;
}
```

Détermination touches clavier :

Le master va lire chaque bit de chaque byte pour déterminer l'état des boutons des slaves. Chaque bit de chaque byte de chaque joueur à une touche de clavier attribué ainsi si un bouton est pressé alors la touche de clavier attribué sera aussi pressée par le master jusqu'à l'information que le bouton a été relâché.

C'est la méthode despair qui s'en charge.

```
if (confirm_header){
  while (2 < Wire.available()){
    header_adresse = Wire.read();
  }
  int k = 0;
  switch (header_adresse){
    case 1:
      while (0 < Wire.available()){
        player1[k] = Wire.read();
        k++;
      }
      despair(player_change1, player1, 1);
      break;
    case 2:
      while (0 < Wire.available()){
        player2[k] = Wire.read();
        k++;
      }
      despair(player_change2, player2, 2);
      break;
  }
}
```

```
void despair(boolean joueur, byte joueurarray[], int n)

for (int i1 = 0; i1 < sizeof(joueurarray) / sizeof(byte); i1++)
{
  if (i1 == 0)
  {
    for (int j1 = 0; j1 < 4; j1++)
    {
      je_suis_perdu = debut_ascii + n * ahhhh + j1;
      if (bitRead(joueurarray[i1], j1) == 1)
      {
        //Serial.println(je_suis_perdu);
        choose_char(je_suis_perdu, true);
      }
      else
      {
        choose_char(je_suis_perdu, false);
      }
    }
  }
  if (i1 == 1)
  {
    for (int j11 = 0; j11 < 8; j11++)
    {
      je_suis_perdu = debut_ascii + n * ahhhh + j11 + 4;
      if (bitRead(joueurarray[i1], j11) == 1)
      {
        //Serial.println(je_suis_perdu);
        choose_char(je_suis_perdu, true);
      }
      else
      {
        choose_char(je_suis_perdu, false);
      }
    }
  }
```

Despair permet de determiner un nombre selon le bit identifier, qui grace à choose char permet d'obtenir une touche du clavier

```
char choose_char(int number, bool press_release){
  switch (number){
    case 33:
      if (press_release) //Player 1 Right{
        Keyboard.press(right_arrow);
      }
      else{
        Keyboard.release(right_arrow);
      }
      break;
```

Il convient de faire attention aux touches de clavier choisie, car dans le cas d'une majuscule le leonardo fera d'abord un shift suivi de la lettre voulue ainsi lorsqu'il faudra configurer un émulateur la touche attribué ne pourra être que shift

Piste d'amélioration :

Il convient d'abord de faire remarquer que la communication I2C n'est pas faite pour de grandes distances et doit se partager entre les différents arduinos, il est alors préférable alors de changer de moyen de communication, par du bluetooth par exemple. Il peut être aussi envisager de changer la nature de la connexion entre l'ordinateur et le leonardo, pour l'instant le leonardo se fait passer pour un clavier mais on peut imaginer changer pour que le leonardo soit considérer comme un gamepad et dans ce cas là ou peut imaginer avoir un leonardo dans chaque manette qui serait relier à l'ordinateur