ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

LANGAGE INTERPRÉTÉ SPÉCIALISÉ

PIERRE-YVES ROCHAT, EPFL

RÉV 2016/01/22

MOTIVATION

Animer une enseigne à LED consiste en une suite d'opérations sur les groupes LED. Animer un afficheur matriciel consiste aussi à envoyer des séquences graphiques. Dans les deux cas, une jolie animation ne se limitera pas à quelques étapes, mais pourra vite devenir longue. Les programmes correspondant vont donc avoir tendance à devenir longs, ce qui va rendre leur lecture fastidieuse et qui risque aussi de remplir rapidement la mémoire du microcontrôleur.

Une technique souvent utilisée consiste à **inventer** un *langage* pour décrire ce qui se passe sur l'enseigne ou l'afficheur et programmer les animations dans ce langage.

LANGAGE ARDUINO

Prenons l'exemple très simple. Pour décrire une animation sur une enseigne, deux ordres suffisent pour décrire les actions :

- mettre un groupe de LED à une certaine intensité
- attendre un certain temps.

Dans le cas simple de sorties tout-ou-rien, voici les procédures Arduino qui vont suffire :

- digitalWrite() de l'Arduino convient pour donner un état à une sortie
- delay() pour une attente.

En observant la taille d'un petit programme sur Energia et en ajoutant des appels à ces procédures, on constate que :

– digitalWrite() prend 8 octets en mémoire
– delay() prend 10 octets en mémoire.

En prenant par exemple un microcontrôleur MSP430G2213, disposant d'une mémoire flash de 2kB (2048 octets), on sera limité à moins de 80 pas de programme, constitué de paires digitalWrite() - delay(). En constatant qu'un simple chenillard dans les deux sens sur 8 bits en prend déjà 16, c'est réellement limitatif !

```
loop() {
  digitalWrite (P2_0, 1); delay (100);
  digitalWrite (P2_1, 1); delay (100);
  digitalWrite (P2_2, 1); delay (100);
  digitalWrite (P2_3, 1); delay (100);
  digitalWrite (P2_4, 1); delay (100);
  digitalWrite (P2_5, 1); delay (100);
  digitalWrite (P2_6, 1); delay (100);
  digitalWrite (P2_7, 1); delay (200);
  digitalWrite (P2_7, 0); delay (100);
  digitalWrite (P2_6, 0); delay (100);
  digitalWrite (P2_5, 0); delay (100);
  digitalWrite (P2_4, 0); delay (100);
  digitalWrite (P2_3, 0); delay (100);
  digitalWrite (P2_2, 0); delay (100);
  digitalWrite (P2_1, 0); delay (100);
  digitalWrite (P2_0, 0); delay (300);
}
```

Bien entendu, les instructions permettant l'accès direct aux registres du microcontrôleur permettent d'économiser la place en mémoire. L'instruction P1OUT |= (1<<0); <--- --- > prend 4 octets. C'est déja mieux ! Mais cherchons une autre solution.

Inventer un langage

Une solution élégante est d'inventer un langage. Il aura les deux même instructions :

– **Mettre une intensité sur une sortie**. Paramètres : numéro de la sorte et intensité (0 ou 1)
– **Attendre**. Paramètre : durée de l'attente

Le programme pourrait alors se présenter sous forme d'un tableau. Nous avons utilisé ici un tableau d'octets. Le programme pour notre chenillard se présenterai alors de la manière suivante :

```
uint8_t Animation[] = { // définition d'un tableau d'octets
  Sortie0+On, Attente+10,
  Sortie1+On, Attente+10,
  Sortie2+On, Attente+10,
  Sortie3+On, Attente+10,
  Sortie4+On, Attente+10,
  Sortie5+On, Attente+10,
  Sortie6+On, Attente+10,
  Sortie7+On, Attente+20,
  Sortie7+Off, Attente+10,
  Sortie6+Off, Attente+10,
  Sortie5+Off, Attente+10,
  Sortie4+Off, Attente+10,
  Sortie3+Off, Attente+10,
  Sortie2+Off, Attente+10,
  Sortie1+Off, Attente+10,
  Sortie0+Off, Attente+30,
  Fin
}
```

Sa taille n'est que de 33 octets. Voici les définitions nécessaire pour que ce tableau se compile correctement :

```
#define On 0b01000000
#define Sortie0 0
#define Sortie1 1
#define Sortie2 2
#define Sortie3 3
#define Sortie4 4
#define Sortie5 5
#define Sortie6 6
#define Sortie7 7

#define Attente 0b10000000
#define Fin 0b1111111
```

LANGAGE BINAIRE

Voici la description binaire de notre langage :

```
// Description des instructions :
// b7 b6 b5 b4 b3 b2 b1 b0  : instructions sur 8 bits
// -----------------------
//  0 i0 s5-s4-s3-s2-s1-s0  : met une intensité sur une sortie
//  1 d6-d5-d4-d3-d2-d1-d0  : attente
// -----------------------
//
```

```
// Sorties sur 6 bits (maximum 64 sorties)
// Intensité sur 1 bit (On ou OFF)
// Durée sur 7 bits, exprimée en dixième de seconde (0 à 12.6 secondes)
```

Ceux qui ont déjà programmé en assembleur trouveront une grande similitude avec la description des instruction en binaire !

On voit que des choix ont été faits pour utiliser au mieux les instructions, qui sont des champs de 8 bits. Le bit de poids fort b7 détermine s'il s'agit d'une instruction pour définir l'intensité ou pour l'attente. Ensuite, les 7 bits restant se répartissent selon l'instruction : une intensité et un numéro de sortie pour l'action sur une sortie, une valeur en dixième de seconde pour l'attente. L'usage de la milliseconde comme unité aurait été trop limitative, étant donné que seuls 7 bits sont à disposition.

Interpréteur

Il reste à écrire une procédure qui va interpréter notre langage et le traduire en instructions pour un microcontrôleur. En voici un exemple :

Exemple plus complexe