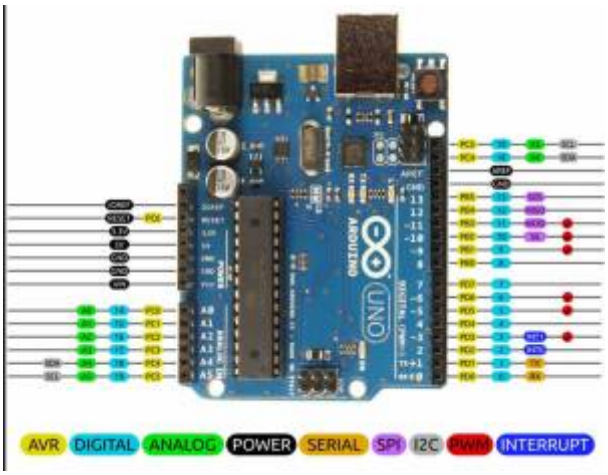
Programmation des Ports sur Arduino



La doc sur les Ports du Blog Eskimon

La Doc de référence Arduino

Registres de port

Les registres de port permettent une manipulation de niveau inférieur et plus rapide des broches d'E / S du microcontrôleur sur une carte Arduino. Les puces utilisées sur la carte Arduino (les ATmega8 et ATmega168) ont trois ports:

```
B (broche numérique 8 à 13)
C (broches d'entrée analogiques)
D (broches numériques 0 à 7)
```

Chaque port est contrôlé par trois registres, qui sont également des variables définies dans le langage Arduino. Le registre DDR détermine si la broche est une entrée ou une sortie. Le registre PORT contrôle si la broche est HIGH ou LOW, et le registre PIN lit l'état des broches INPUT configurées pour entrer avec pinMode (). Les cartes des puces ATmega8 et ATmega168 montrent les ports. La nouvelle puce Atmega328p suit exactement le brochage de l'Atmega168.

Les registres DDR et PORT peuvent être à la fois écrits et lus. Les registres PIN correspondent à l'état des entrées et ne peuvent être lus.

PORTD mappe sur les broches numériques Arduino 0 à 7

```
DDRD - Le registre de direction des données du port D - lecture / écriture
PORTD - Le registre de données du port D - lecture / écriture
PIND - Le registre des broches d'entrée du port D - lecture seule
```

PORTB correspond aux broches numériques Arduino 8 à 13 Les deux bits hauts (6 et 7) correspondent aux broches en cristal et ne sont pas utilisables

```
  DDRB - Le registre de direction des données du port B - lecture / écriture
  PORTB - Le registre de données du port B - lecture / écriture
  PINB - Le registre des broches d'entrée du port B - lecture seule
```

PORTC correspond aux broches analogiques Arduino 0 à 5. Les broches 6 et 7 ne sont accessibles que sur l'Arduino Mini

```
  DDRC - Le registre de direction des données du port C - lecture / écriture
  PORTC - Le registre de données du port C - lecture / écriture
  PINC - Le registre des broches d'entrée du port C - lecture seule
```

Chaque bit de ces registres correspond à une seule broche; Par exemple, le bit faible de DDRB, PORTB et PINB fait référence à la broche PB0 (broche numérique 8). Pour un mappage complet des numéros de broches Arduino aux ports et aux bits, consultez le diagramme de votre puce: ATmega8 , ATmega168 . (Notez que certains bits d'un port peuvent être utilisés pour des choses autres que les entrées / sorties; faites attention de ne pas changer les valeurs des bits de registre qui leur correspondent.) Exemples

En se référant à la carte des broches ci-dessus, les registres PortD contrôlent les broches numériques Arduino 0 à 7.

Vous devez cependant noter que les broches 0 et 1 sont utilisées pour les communications série pour la programmation et le débogage de l'Arduino, donc le changement de ces broches doit généralement être évité sauf si nécessaire pour les fonctions d'entrée ou de sortie série. Sachez que cela peut interférer avec le téléchargement ou le débogage du programme.

```
  DDRD est le registre de direction pour le port D (broches numériques
Arduino 0-7). Les bits de ce registre contrôlent si les broches de
  PORTD sont configurées comme entrées ou sorties, par exemple:
```

```
  DDRD = B11111110; // définit les broches Arduino 1 à 7 comme sorties, la
broche 0 comme entrée
  DDRD = DDRD | B11111100; // c'est plus sûr car il définit les broches 2 à
7 comme sorties
                  // sans changer la valeur des broches 0 & 1, qui sont RX &
TX
```

See the bitwise operators reference pages and Le didacticiel Bitmath dans le Playground PORTD est le registre de l'état des sorties. Par exemple; PORTD = B10101000; sets digital pins 7,5,3 HIGH

Vous ne verrez cependant que 5 volts sur ces broches si les broches ont été définies comme sorties en utilisant le registre DDRD ou avec pinMode ().

PIND est la variable du registre d'entrée. Il lira toutes les broches d'entrée numériques en même temps. Pourquoi utiliser la manipulation de port?

À partir du didacticiel Bitmath

De manière générale, faire ce genre de chose n'est pas une bonne idée. Pourquoi pas? Voici quelques raisons:

Le code est beaucoup plus difficile à déboguer et à maintenir, et il est beaucoup plus difficile à comprendre pour les autres. Le processeur ne prend que quelques microsecondes pour exécuter le code, mais cela peut vous prendre des heures pour comprendre pourquoi cela ne fonctionne pas correctement et le réparer! Votre temps est précieux, non? Mais le temps de l'ordinateur est très bon marché, mesuré par le coût de l'électricité que vous l'alimentez. Il est généralement préférable d'écrire du code de la manière la plus évidente.

Le code est moins portable. Si vous utilisez digitalRead () et digitalWrite (), il est beaucoup plus facile d'écrire du code qui fonctionnera sur tous les microcontrôleurs Atmel, alors que les registres de contrôle et de port peuvent être différents sur chaque type de microcontrôleur.

Il est beaucoup plus facile de provoquer des dysfonctionnements involontaires avec un accès direct au port. Remarquez comment la ligne DDRD = B11111110; ci-dessus mentionne qu'il doit laisser la broche 0 comme broche d'entrée. La broche 0 est la ligne de réception (RX) sur le port série. Il serait très facile de provoquer accidentellement l'arrêt de votre port série en changeant la broche 0 en une broche de sortie! Maintenant, ce serait très déroutant lorsque vous êtes soudainement incapable de recevoir des données série, n'est-ce pas?

Alors vous vous dites peut-être, génial, pourquoi aurais-je envie d'utiliser ce truc alors? Voici quelques-uns des aspects positifs de l'accès direct au port:

Vous devrez peut-être pouvoir activer et désactiver les broches très rapidement, c'est-à-dire en quelques fractions de microseconde. Si vous regardez le code source dans lib / cibles / arduino / câblage.c, vous verrez que digitalRead () et digitalWrite () sont chacun environ une douzaine de lignes de code, qui sont compilées en quelques instructions machine. Chaque instruction machine nécessite un cycle d'horloge à 16 MHz, ce qui peut s'additionner dans les applications sensibles au temps. L'accès direct au port peut faire le même travail en beaucoup moins de cycles d'horloge.

Parfois, vous devrez peut-être définir plusieurs broches de sortie exactement en même temps. Appel digitalWrite (10, HIGH); suivi de digitalWrite (11, HIGH); fera passer la broche 10 à l'état HAUT plusieurs microsecondes avant la broche 11, ce qui peut perturber certains circuits numériques externes sensibles au temps que vous avez connectés. Vous pouvez également régler les deux broches à un niveau élevé exactement au même moment en utilisant PORTB | = B1100;

Si vous manquez de mémoire programme, vous pouvez utiliser ces astuces pour réduire la taille de votre code. Il faut beaucoup moins d'octets de code compilé pour écrire simultanément un tas de broches matérielles simultanément via les registres de port que d'utiliser une boucle for pour définir chaque broche séparément. Dans certains cas, cela peut faire la différence entre l'adaptation de votre programme dans la mémoire flash ou non!

Exemple de programme Arduino UNO pour faire clignoter 2 LEDS

2 LEDS branchées sur les broches: LED1 = - sur 2 et cmd+ sur 3 , LED2 = - sur 6 cmd+ sur 7

ports_Leds.ino

```
void setup() {
```

```
    // On positionne en sortie OUTPUT   ( 1 ) les broches 2 à 7  sur
les bits 2 à 7 ,
    // On part de la droite vers la gauche pour lire les n° des bits
donc des broches.
    DDRD = B11111110; // ATTENTION le bit 0 = 0 et le bit 1 = 1 sinon
pas de liaison série ....!!!
    // Le B majuscule en début de séquence indique un nombre Binaire
}

void loop(){

   // on met du 5 volts ou à 1 ( HIGH) les broches 3 et 7 on allume les
2 LEDS
   PORTD = B10001000; // toujours  0 sur les bits 0 et 1 ==> liaison
série...
   delay(1000); // on attend 1s
   PORTD = B00000000; // On eteind les 2 LEDS
   delay(1000); // attente 1s

}
```

*bitRead(x, n); bitRead() permet de lire l'état d'un bit dans un nombre entier. *bitWrite(x, n, b); bitWrite() permet d'écrire l'état d'un bit dans un nombre entier. *bitSet(x, n); bitSet() permet de mettre un bit à “1” dans un nombre entier. *bitClear(x, n); bitClear() permet de mettre un bit à “0” dans un nombre entier. *bit(n);bit() permet de retourner la valeur numérique correspondant au poids d'un bit :

*https://www.carnetdumaker.net/articles/quelques-fonctions-bien-pratiques-du-framework-arduino/

From:
http://chanterie37.fr/fablab37110/ - **Castel'Lab le Fablab MJC de Château-Renault**

Permanent link:
http://chanterie37.fr/fablab37110/doku.php?id=start:arduino:ports&rev=1607201030

Last update: **2023/01/27 16:08**