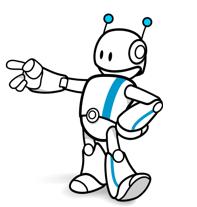
Chapitre 3 : Ports et registres Microcontrôleur AVR ATmega16 / ATmega32

**Introduction**

* L'AVR ATmega16/32 à 32 broches constituant quatre ports. Les ports sont énumérés ci-dessous:

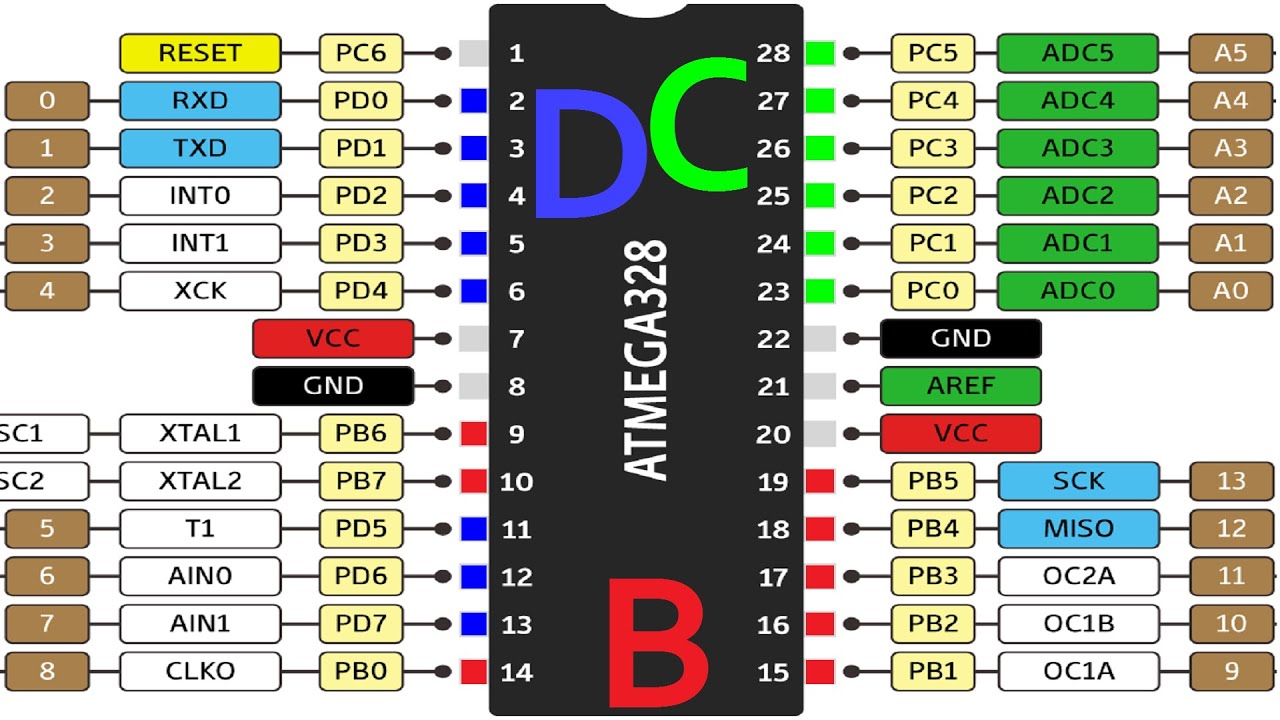
 PORT A , PORT B, PORT C, PORT D.

* Chaque port a 8 broches.
* Les broches de ces quatre ports peuvent être utilisées comme entrées / sorties à usage général.
* Ces broches peuvent être configurées en entrée ou en sortie à l'aide des trois registres d'E / S de chaque port :

1.   **DDRx Register:** ils sont utilisés pour décider de la direction des broches, (broches d’entrées ou broches de sortie ).

2. **PINx:** Ils sont utilisés pour lire le niveau logique sur les ports.   
3.   **PORTx registers:** ils servent à définir la logique sur les broches HIGH ou LOW.   
(Où **x** peut être A, B, C ou D en fonction des registres de ports à adresser).

* Chaque broche est également associée à une fonctionnalité spéciale.



**Registres de ports E / S :**

Trois registres d'E / S sont associés à chaque port :

**DDRx** : Registres de direction de données

* Ce sont des registres 8 bits.
* Ceux-ci sont utilisés pour configurer les broches des ports en entrée ou en sortie.
* L'écriture d'un 1 dans les bits de ce registre définit ces broches comme sortie.
* L'écriture d'un zéro sur les bits de ce registre définit ces broches comme des entrées.
* Tous les bits de ces registres peuvent être lus et écrits.
* La valeur initiale de ces bits est zéro.

Exemple :

* Définition du port D comme port de sortie:  DDRD = 0xFF;

L'écriture 1 sur les bits de ce registre configure les broches correspondant à ces broches en tant que broches de sortie.

Ici, nous avons configuré les 8 broches du port D en tant que broches de sortie en écrivant 1 sur chacune d’elles. (0xFF = 0b11111111).

* Définition du port D comme port d’entrée:  DDRD = 0x00;   
  L'écriture de 0 sur les bits de ce registre configure les broches correspondant à ces broches en tant que broches de sortie.    
  Ici, nous avons configuré les 8 broches du port D en tant que broches d’entrée en écrivant 0 sur chacune d’elles. (0x00 = 0b00000000).

PORTx  : registres de données

* Ce sont des registres 8 bits.
* Ceux-ci sont utilisés pour placer les broches des ports dans l'état logique haut ou bas.
* Écrire un 1 dans ce registre met une logique HIGH (5V) sur ces broches.
* Tandis que l'écriture d'un zéro dans les bits de ce registre place une logique BASSE (0V) sur ces broches.
* Tous les bits de ces registres peuvent être lus et écrits.
* La valeur initiale de ces bits est zéro.

Exemple:

Nous allons utiliser le registre PORTx du port D pour écrire une valeur 0x55 sur le port D. PORTD = 0x55;

PINx: Registres d'adresses des broches d' entrée

* Ce sont des registres 8 bits.
* Ils sont utilisés pour lire les valeurs sur les broches spécifiques du port.
* Ces bits sont en lecture seule et ne peuvent pas être écrits.

Exemple:

Nous lirons la valeur sur le port D dans une variable de 8 bits nommée 'valeur\_port'. valeur\_port = PIND;



**Gestion des entrées / sorties digitales par ARDUINO :**

**pinMode(pin,mode) :** pin = numero sur connecteur, mode = INPUT / OUTPUT / INPUT\_PULLUP.

**digitalWrite(pin,value) :** pin = numero sur connecteur, value = HIGH/LOW.

**int digitalRead(pin) :** pin = numero sur connecteur, retourne la valeur.

 L'exemple le plus simple, fourni par ARDUINO, consiste à faire clignoter la LED raccordée à la

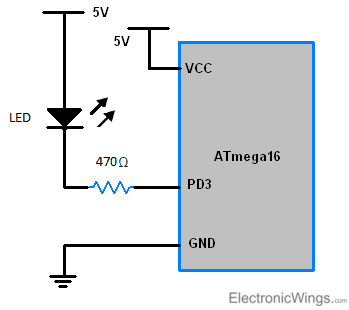
broche PB5 du microcontrôleur, broche également disponible sur les connecteurs de la carte et

portant le numéro 13.

|  |  |
| --- | --- |
| Exemple ARDUINO UNO | Faire clignoter la LED (PB5) sans utiliser les fonctions ARDUINO |
| void setup()  {  pinMode(13, OUTPUT); //broche PB5 en sortie  }  void loop()  {  digitalWrite(13,HIGH); // set PB5  delay(200); // délai 200 ms  digitalWrite(13,LOW); // clear PB5  delay(1000); // délai 1 s  } | // LED <- PB5 : LED allumée quand PB5=1  void setup() {  **// configurer broche PB5 en sortie**  **DDRB |= B100000;** // B100000 représente la constante 0x20  // OU DDRB|=0x20; // équivalent  // OU DDRB |= 32; // équivalent quoique moins clair  **PORTB &= ~(1<<PORTB5);** // **PORTB.5 <- 0**  }  void loop() {  **PORTB |= (1<<PORTB5); // PORTB.5 <- 1**  delay(200); // 200 ms  **PORTB &= ~(1<<PORTB5); // PORTB.5 <- 0**  delay(1000); // 1s  } |

Exemple 1

Écrivons un code pour programmer la broche 3 du port D en tant que broche de sortie et l’utiliser pour piloter une LED. Nous allons basculer la LED avec un peu de retard.



**Schéma de circuit de l'exemple 1**

**(Source de l'image: www.electronicwings.com)**

Comme indiqué dans la figure ci-dessus, la DEL est connectée à la broche 3 du port D. L'anode de la DEL est connectée à 5V.

La plupart des LED nécessitent environ 10 mA de courant pour briller correctement.

Si un courant supérieur au courant nominal maximum de la LED la traverse, la LED sera définitivement endommagée.

Le courant nominal maximum pour les LED est généralement compris entre 20 et

30 mA.

Afin de limiter le courant traversant la LED, nous connectons une résistance en série avec elle.

En supposant que la LED présente une chute de tension directe de 0,7 V et qu'un courant de

10 mA soit suffisant pour qu'elle s'allume correctement, la valeur de la résistance requise peut être calculée comme suit :

\ mathbf {R = \ frac {Tension fournie à la LED - Chute de tension directe de la LED} {DEL parcourant le courant} = \ frac {5V - 0.7V} {10mA}}

Donc, \ mathbf {R = \ frac {4.3V} {10mA} = 430 \ Omega}

La valeur standard la plus proche de la résistance disponible est 470Ω. Donc, cette valeur est utilisée dans le circuit.

/\*

Output Pin Programming to Drive LED

www.electronicwings.com

\*/

#define F\_CPU 8000000UL

#include <avr/io.h>

#include <util/delay.h>

int main(void)

{

DDRD = 0xFF; /\* Making all 8 pins of Port D as output pins \*/

/\*

In order to make only PD3 as output while keeping direction of other pins as it is, do the following

DDRD = DDRD | (1<<3);

This is equivalent to writing one to only PD3 while keeping the other pins unchanged

\*/

while(1)

{

PORTD = PORTD | (1<<3); /\* Making PD3 high. This will make LED ON \*/

\_delay\_ms(100);

PORTD = PORTD & (~(1<<3)); /\* Making PD3 low. This will make LED OFF \*/

\_delay\_ms(100);

/\* Do not keep very small delay values. Very small delay will lead to LED appearing continuously ON due to persistence of vision \*/

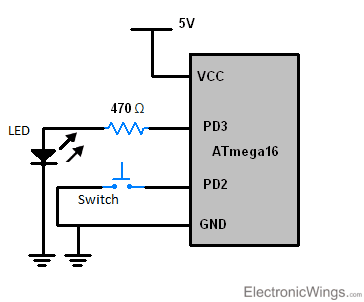
}

return 0;

}

 Exemple 2

Écrivons un code pour lire le statut de la broche en entrée. Nous indiquerons cet état sur LED.



**Schéma de circuit de l'exemple 2**

**(Source de l'image: www.electronicwings.com)**

Dans cet exemple,

* Nous utilisons switch comme entrée. Le commutateur est connecté à PD2 du contrôleur.
* Nous utilisons des voyants comme indicateur de l’état de la broche PD2. La LED est connectée à PD3.

Remarque :

 Dans ce code, nous allons montrer l'état de la broche sur la LED. Nous ne montrerons pas l'état de l'interrupteur sur la LED.   
Lorsque le commutateur est ouvert, la broche est HAUTE, le voyant est allumé.   
Lorsque l’interrupteur est enfoncé, la broche est BAS, le voyant est éteint.

/\*

Input Pin Programming

www.electronicwings.com

\*/

#define F\_CPU 8000000UL

#include <avr/io.h>

#include <util/delay.h>

int main(void)

{

DDRD = DDRD | (1<<3); /\* Make PD3 as output pin \*/

DDRD = DDRD & (~(1<<2)); /\* Make PD2 as input pin \*/

PORTD = PORTD | (1<<2); /\* Enable pull-up on PD2 by writing 1 to it \*/

int pin\_status;

while(1)

{

pin\_status = PIND & (1<<2); /\*Read status of pin PD2 \*/

if(pin\_status) /\* Transmit status of pin PD2 on to pin PD3 to drive LED. \*/

{

PORTD = PORTD | (1<<3); /\* Switch is open, pin\_status = 1, LED is ON \*/

}

else

{

PORTD = PORTD & (~(1<<3)); /\* Switch is closed, pin\_status = 0, LED is OFF \*/

}

}

return 0;

}

Comprendre les broches d'E / S

Toutes les broches du port ont des résistances de tirage sélectionnables individuellement avec une résistance indépendante de la tension d'alimentation. Toutes les broches ont des diodes de protection sur VCC et sur la terre.

États Pin

* Une broche de port peut avoir quatre états différents en fonction de l'état des bits correspondant à cette broche dans le registre de direction des données (DDRx) et le registre de données (PORTx) de ce port spécifique.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DDxn** | **PORTxn** | **E/S** | **Pull-up** | **Commentaire** |
| 0 | 0 | **E** | Non | État (Z ) |
| 0 | 1 | **E** | Oui | Pxn sera source de courant Pull-up |
| 1 | 0 | **S** | Non | Sortie basse (évier) |
| 1 | 1 | **S** | Non | Sortie haute (Source) |
| x est un port et peut être A, B, C ou D  n est le numéro d'identification personnel du port et peut être 0, 1, 2, 3, 4, 5, 6 ou 7 | | | | |

* Comme indiqué dans le schéma équivalent des broches d'E / S, chaque broche de port possède une résistance de tirage interne. Cette résistance interne à toutes les broches peut être désactivée en écrivant 1 dans le bit PUD du registre SFIOR. Lorsque cela est fait, les broches sont à (états à haute impédance) si DDRxn = 0 et PORTxn = 1.

Précautions à prendre lors de la configuration des E / S

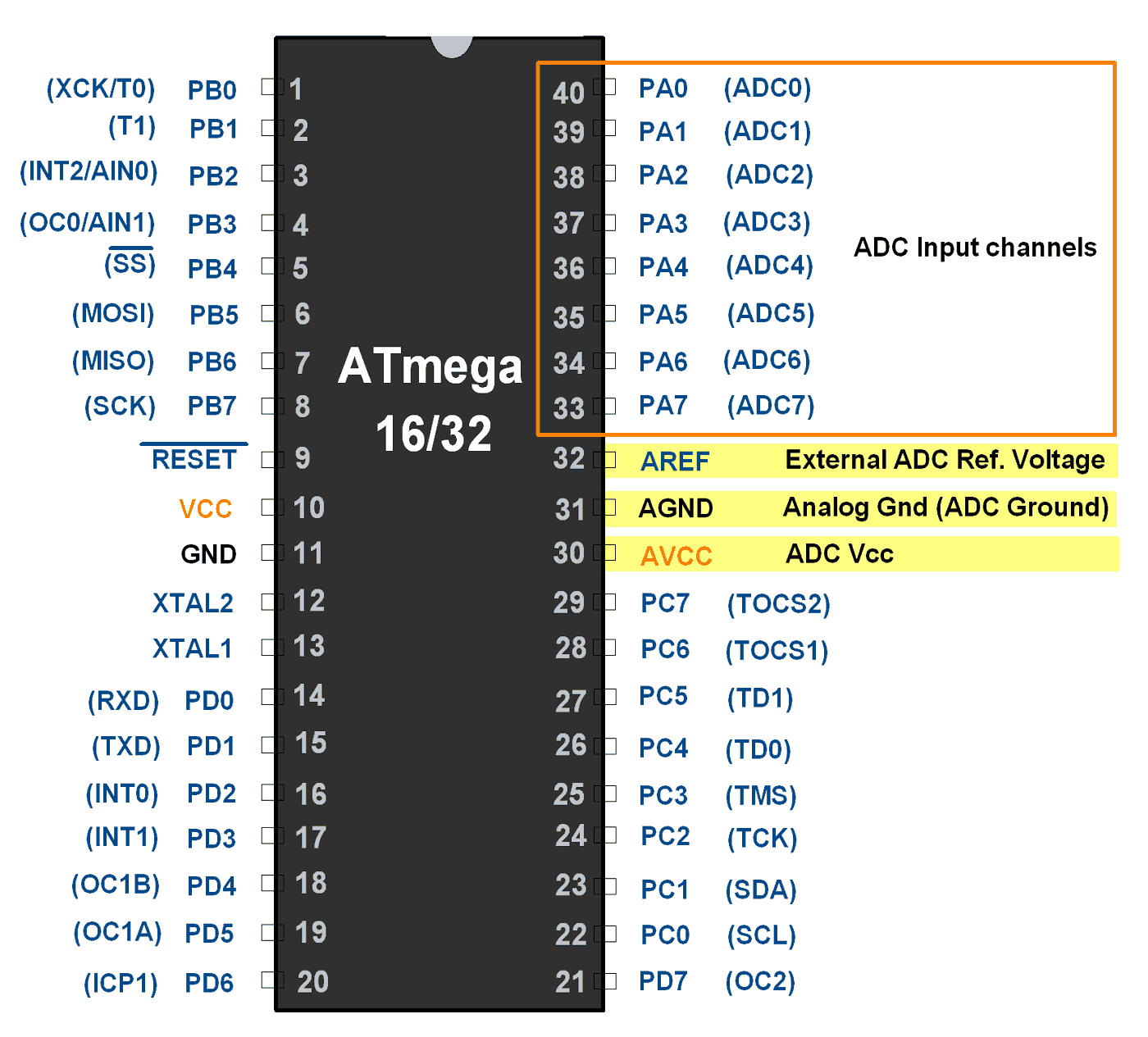
* Lors de la configuration de broches en tant que broches de sortie avec la logique HIGH, assurez-vous que la broche n'est pas directement connectée à la terre.
* Lors de la configuration de broches en tant que broches de sortie avec une logique LOW, assurez-vous que la broche n'est pas directement connectée à Vcc.
* Lors de la configuration des broches en tant que broches d'entrée, il faut garder à l'esprit la structure de tension interne et effectuer les connexions en conséquence.

Chapitre 4 : ADC dans AVR ATmega16 / ATmega32

Introduction

Le convertisseur analogique-numérique (ADC) est l’appareil le plus largement utilisé dans les systèmes intégrés spécialement conçu pour l’acquisition de données. Dans la série AVR ATmega, un contrôleur ADC 10 bits est normalement intégré au microcontrôleur.

ATmega16 / 32 prend en charge huit canaux ADC, ce qui signifie que nous pouvons connecter huit entrées analogiques à la fois. Les canaux ADC 0 à 7 sont présents sur PORTA. c'est-à-dire les broches n ° 33 à 40.



**ADC Broches de ATmega 16/32**

**(Source de l'image: www.electronicwings.com)**

Le microcontrôleur dispose d’un convertisseur analogique-numérique 10 bits, ce qui signifie que nous aurons une sortie numérique comprise entre 0 et 1023.

C'est-à-dire que lorsque l'entrée est 0V, la sortie numérique sera 0V et quand l'entrée sera 5V (Vref = 5V), nous obtiendrons la sortie numérique la plus élevée correspondant à 1023 pas, ce qui correspond à 5V.

Donc,

Si Vref = 5V alors 5/1023 = 4,88 mV.

Si Vref = 2,56 alors 2,56 / 1023 = 2,5 mV.

La sortie de données numériques sera donc V*out* = Vin / taille du pas.

ATmega16 / 32 ADC

* C'est ADC 10 bits
* Les données binaires de sortie converties sont conservées dans les registres ADCL (result Low) et ADCH (result High).
* ADC fournit une sortie 10 bits, donc (ADCH: ADCL), seuls 10 bits sont utiles sur 16 bits.
* Nous avons des options pour utiliser ces 10 bits comme bits supérieurs ou inférieurs.
* Nous avons également trois options pour Vref. 1. AVcc (Vcc analogique), 2. Interne 2.56 v 3. Externe A *ref* . .
* Le temps de conversion total dépend de la fréquence de l’horloge et de ADPS 0 : 2 (diviseur de fréquence).
* Si vous avez décidé d'utiliser la broche AVcc ou Vref comme référence de tension ADC, vous pouvez la rendre plus stable et augmenter la précision de l'ADC **en branchant un condensateur entre cette broche et GND** .

Registre ADC

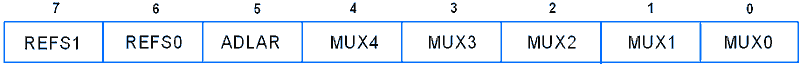
Dans AVR ADC, nous devons comprendre quatre registres principaux -

1. **ADCH: contient** un octet plus élevé de données converties numériques
2. **ADCL :**  contient l'octet inférieur de données converties numériques
3. **ADMUX:** registre de sélection du multiplexeur ADC
4. **ADCSRA:** registre de contrôle et d'état ADC

**Registre : ADCL, ADCH**

Les deux premiers registres contiennent les données numériques converties, sur 10 bits.

**ADMUX Inscrivez-vous**



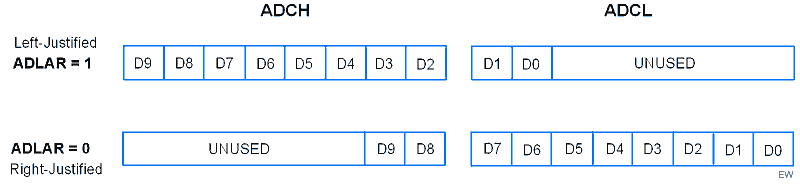
**Bit 7: 6 - REFS1: 0: Bits de sélection de référence**

Sélection de tension de référence pour ADC

| **REFS1** | **REFS0** | **Vref à ADC** |
| --- | --- | --- |
| 0 | 0 | Broche AREF |
| 0 | 1 | AVCC pin ie Vcc 5 V |
| 1 | 0 | Réservé |
| 1 | 1 | Interne 2.56v |

**Bit 5 - ADLAR: résultat de réglage gauche ADC**

Utilisez la sortie 10 bits en tant que bits supérieurs ou inférieurs dans ADCH & ADCL.



**Bits 4: 0 - MUX4: 0: Bits de sélection de canal analogique**

Nous pouvons sélectionner le canal d'entrée ADC0 à ADC7 en utilisant ces bits. Ces bits sont également utilisés pour sélectionner des entrées de comparateur (intégrées dans AVR) avec un gain différent. Nous couvrirons ces opérations de comparaison dans une autre partie.

La sélection du canal est très facile, il suffit de mettre le numéro du canal dans MUX4: 0.

Supposons que vous connectiez une entrée au canal ADC 2, alors mettez 00010 dans MUX4: 0.

Supposons que vous connectiez une entrée au canal ADC 5, alors insérez 00101 dans MUX4: 0.

**Registre ADCSRA:**

Registre ADCSRA

* **Bit 7 - ADEN: Activer ADC**

L'écriture de 1 sur ce bit active le CAN. En l'écrivant à zéro, le CAN est désactivé. La mise hors tension de l’ADC pendant qu’une conversion est en cours met fin à cette conversion.

* **Bit 6 - ADSC: démarrage de la Conversion de l’ ADC**

L'écriture de 1 sur ce bit commence la conversion.

* **Bit 5 - ADATE: Activation automatique du déclencheur ADC**

En écrivant 1 de ce bit, le déclenchement automatique de l’ADC est activé.

* **Bit 4 - ADIF: indicateur d'interruption ADC**

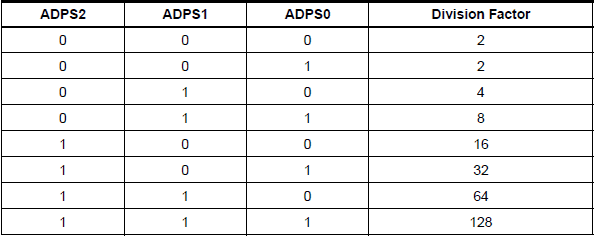
Ce bit est activé lorsqu'une conversion ADC est terminée et que les registres de données sont mis à jour.

* **Bit 3 - ADIE: Activer l'interruption ADC**

En écrivant 1 sur ce bit, l’interruption de conversion complète du convertisseur analogique-numérique est activée.

* **Bits 2: 0 - ADPS2: 0: Bits ADC Prescaler Select**

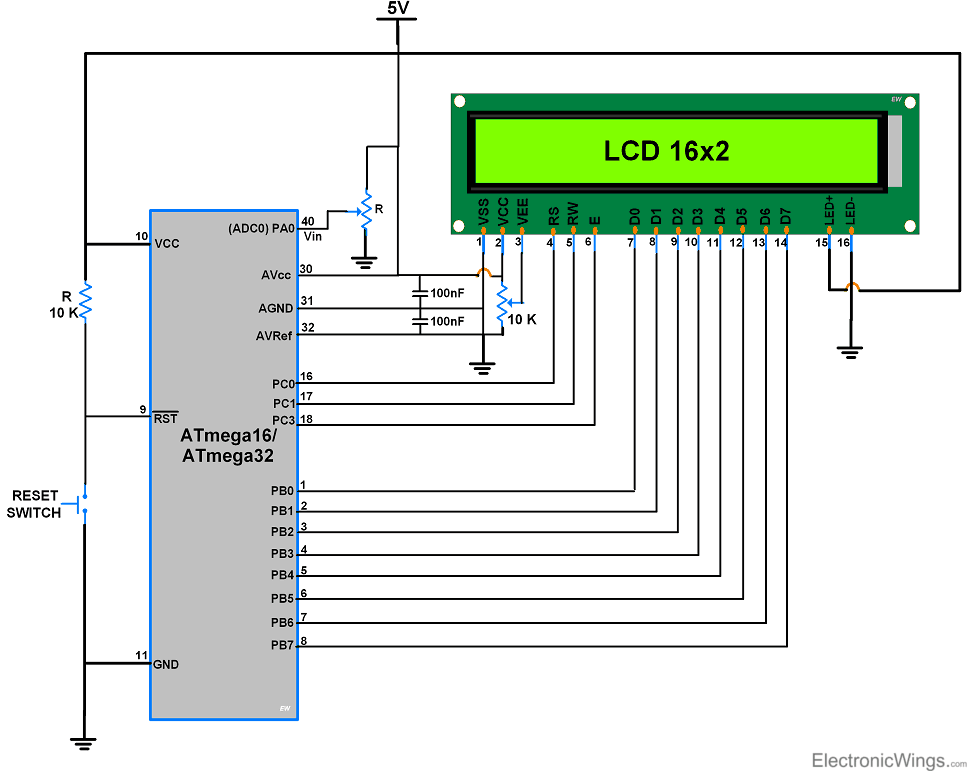
Ces bits déterminent le facteur de division entre la fréquence XTAL et l’horloge d’entrée du CAN.



Pour notre AVR, Nous devons toujours veiller à ne pas dépasser la fréquence ADC supérieure à 200 KHz. Supposons que votre fréquence d'horloge de l'AVR soit de 8 MHz, alors nous devons utiliser le scrutateur 64 ou 128. Parce que cela donne 8 MHz / 64 = 125 KHz, ce qui est inférieur à 200 KHz.

Schéma

LCD16x2 est utilisé pour afficher la valeur convertie numérique du canal 0.



**Schéma de circuit pour l'utilisation de ADC0 d'ATmega 16/32**

**(Source de l’image : www.electronicwings.com)**

Étapes pour programmer ADC

1. Activer ADC avec l'écriture de 1 sur ADEN dans ADCSRA.
2. Créez une broche de canal ADC comme entrée.
3. Définissez le bit d'activation ADC dans ADCSRA, sélectionnez la vitesse de conversion à l'aide de ADPS2 : 0. Par exemple, nous allons sélectionner le diviseur 128.
4. Sélectionnez la tension de référence ADC en utilisant REFS1: REFS0 dans le registre ADMUX. Par exemple, nous utiliserons AVcc comme tension de référence.
5. Sélectionnez le canal d'entrée ADC à l'aide de MUX4: 0 dans ADMUX, par exemple, nous utiliserons le canal 0. Donc, notre valeur dans le registre ADCSRA = 0x87 et ADMUX = 0x40.
6. Commencez la conversion en définissant le bit ADSC dans ADCSRA.
7. Attendez la fin de la conversion en interrogeant le bit ADIF dans le registre ADCSRA.
8. Une fois que le bit ADIF est devenu HIGH, lisez les registres ADCL et ADCH pour obtenir une sortie numérique.

 Programme

#define F\_CPU 8000000UL

#include <avr/io.h>

#include <util/delay.h>

#include <stdlib.h>

#include "LCD\_16x2\_H.h"

ADC\_Init()

{

DDRA=0x0; /\* Make ADC port as input \*/

ADCSRA = 0x87; /\* Enable ADC, fr/128 \*/

ADMUX = 0x40; /\* Vref: Avcc, ADC channel: 0 \*/

}

ADC\_Read(char channel)

{

int Ain,AinLow;

ADMUX=ADMUX|(channel & 0x0f); /\* Set input channel to read \*/

ADCSRA |= (1<<ADSC); /\* Start conversion \*/

while((ADCSRA&(1<<ADIF))==0); /\* Monitor end of conversion interrupt \*/

\_delay\_us(10);

AinLow = (int)ADCL; /\* Read lower byte\*/

Ain = (int)ADCH\*256; /\* Read higher 2 bits and

Multiply with weight \*/

Ain = Ain + AinLow;

return(Ain); /\* Return digital value\*/

}

int main()

{

char String[5];

char String[5];

int value;

ADC\_Init();

LCD\_Init(); /\* Initialization of LCD \*/

LCD\_String("ADC value"); /\* Write string on 1st line of LCD \*/

while(1)

{

LCD\_Command(0xc4); /\* LCD16x2 cursor position \*/

value=ADC\_Read(0); /\* Read ADC channel 0 \*/

itoa(value,String,10); /\* Integer to string conversion \*/

LCD\_String(String);

LCD\_String(" ");

}

return 0;

}

**PROGRAMMATION DE L'ADC À L'AIDE DE L'ARDUINO**

* 1. la fonction analogRead(pin) va nous permettre de lire la valeur lue sur une entrée analogique de l'Arduino.
  2. on a une valeur retournée par la fonction comprise entre 0 et 1023.

Référencées :

1/ Fiche technique ATmega16 et ATmega32

2/ www.electronicwings.com

3/