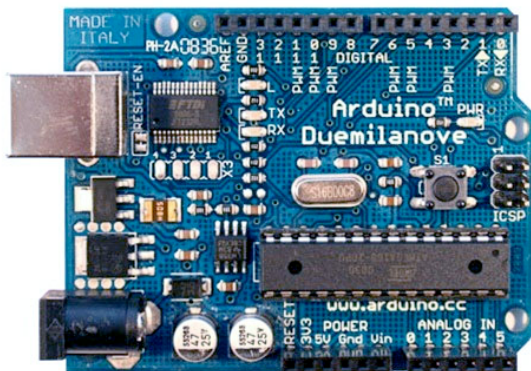


# arduino programmeer manual



**A. Kompanje**

V1.0 april 2009

Op internet vind je dit document ook met als naam: arduino programming notebook geschreven in het engels. Dit document is herschreven naar het Nederlands en aangepast met flowcharts en ander bruikbaar materiaal.

## **Arduino Programmeer manual**

De informatie is onder andere verkregen door:

<http://www.arduino.cc>

<http://www.arduino.nu>

<http://www.wiring.org.co>

<http://www.antratek.nl>

Aan deze tutorial is ook geschreven door::

Brian W. Evans

Paul Badger

Massimo Banzì

Hernando Barragán

David Cuartielles

Tom Igoe

Daniel Jolliffe

Todd Kurt

David Mellis

and others

**Uitgegeven: april 2009**

Deze tutorial valt onder de licentie: Creative Commons Attribution-Share Alike 2.5.

Een kopie van deze licentie staat op:

<http://creativecommons.org/licenses/by-sa/2.5/>

# inhoud

<b>structuur</b>	
structuur	6
setup()	6
loop()	6
functies	7
{ } accolades	7
; puntkomma	8
/*... */ blok commentaar	8
// regel commentaar	8
<b>variabelen</b>	
variabelen	9
variabelen declareren	9
variable bereik	10
<b>datatypes</b>	
byte	11
int	11
long	11
float	11
arrays	12
<b>rekenen</b>	
rekenen	13
samengestelde opdrachten	13
vergelijken van getallen	14
logische berekeningen	14
<b>constanten</b>	
constanten	15
true/false	15
high/low	15
input/output	15

<b>vergelijkingen</b>	
if	16
if... else	17
for	18
while	19
do... while	19
<b>digitale i/o</b>	
pinMode(pin, mode)	20
digitalRead(pin)	21
digitalWrite(pin, value)	21
<b>analoge i/o</b>	
analogRead(pin)	22
analogWrite(pin, value)	22
<b>timer</b>	
delay(ms)	23
millis()	23
<b>rekenen</b>	
min(x, y)	23
max(x, y)	23
<b>random</b>	
randomSeed(seed)	24
random(min, max)	24
<b>seriëel</b>	
serial.begin(rate)	25
Serial.println(data)	25
<b>Bijlage 1</b>	
digital output	26
digital input	27
analoge ingang	29
relais	30
externe voeding	31
i2c	32
servo	34
LCD Display	37
schema Arduino Duemilanove	41
datasheet Atmel 168/328	42

# voorwoord

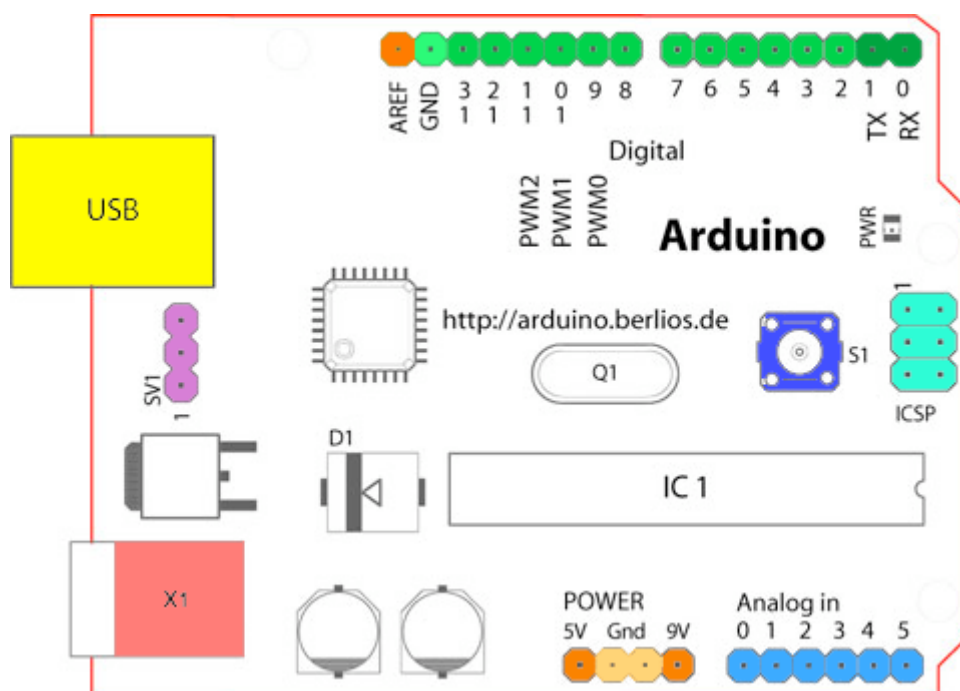
Dit document beschrijft de Arduino microcontroller met een gemakkelijk te leren commandostructuur.

Om de start simpel te maken worden een aantal ingewikkelde begrippen en commando's niet besproken, zodat we hier kunnen spreken over een echte beginners manual. Wil je meer weten en heb je diepgang nodig koop dan een boek over de taal C of verdiep je in materiaal dat diverse websites aanbieden.

Zoals hierboven al is gemeld is de basis structuur van de taal die gebruikt wordt bij de Arduino geschreven in C. Ook komen diverse bibliotheken aan bod die geschreven zijn om jou het werk makkelijker te maken. Er komen veel voorbeelden aan bod die je dagelijks zult gebruiken. In de bijlagen staan nog diverse (kleine), maar zeer bruikbare, voorbeelden aangevuld met de gebruikte schema's.

Dit document had echter niet geschreven kunnen worden zonder de community en de makers rond het Arduino project. Al het originele materiaal en de laatste updates vind je daarom ook op de Arduino website op:

<http://www.arduino.cc>



## structuur

De basisstructuur van de Arduino programmeertaal is erg simpel. Het bestaat uit minstens twee gedeeltes (blokken). Deze twee gedeeltes (blokken), of functies vormen een aantal statements (programma commando's). Voorbeeld:

```
void setup()
{
  statements;
}
void loop()
{
  statements;
}
```

Waarbij setup() de voorbereiding is en loop() de uitvoering. Beide functies zijn nodig om een programma te kunnen laten werken.

In de setup functie worden variabelen gedeclareerd en bepaald welke pinnen ingang of uitgang worden. De setup functie wordt slechts eenmaal doorlopen.

De loop functie volgt na de setup functie en wordt vaak oneindig herhaalt. De loop functie leest vaak de inputs en laat afhankelijk daarvan bepalen wat de outputs moeten doen. Eigenlijk komt het er op neer dat de loop functie de motor van het programma is dus daar waar al het werk moet gebeuren.

## setup()

De setup() functie wordt één keer aangeroepen wanneer het programma start. Het wordt gebruikt om pin modes te initialiseren of begint met seriële communicatie. De syntax ziet er als volgt uit:.

```
void setup()
{
  pinMode(pin, OUTPUT);          // maak de 'pin' als uitgang
}
```

## loop()

Nadat de setup() functie aangeroepen is volgt de loop() functie. Deze doet precies wat de naam zegt en loopt constant te meten om vervolgens te reageren door veranderingen aan te brengen. De loop functie is een oneindige loop. De syntax:

```
{
  digitalWrite(pin, HIGH);      // zet 'pin' aan
  delay(1000);                  // Eén seconde pauze
  digitalWrite(pin, LOW);       // zet 'pin' uit
  delay(1000);                  // Eén seconde pauze
}
```

## functies

Een functie is een blok met code dat een naam heeft gekregen en waarin instructies staan die uitgevoerd moeten worden wanneer de functie aangeroepen wordt. De functies `void setup()` en `void loop()` zijn al genoemd, maar andere functies kunnen ook. Er zijn ook op maat gemaakte functies die de het aantal opdrachten in een programma reduceren, waardoor het geheel overzichtelijker is.

Functies moeten eerst gedeclareerd worden in hun type.

Dat ziet er als volgt uit:

```
type functionName(parameters)
{
    statements;
}
```

Een voorbeeld: De volgende type integer functie `delayVal()` wordt gebruikt om een vertraging in te bouwen bij het uitlezen van een waarde van een aangesloten potentiometer. Als eerste wordt een locale variabele `v` gedeclareerd die vervolgens een waarde krijgt tussen 0 en 1023. In een volgende regel is een functie `v /= 4`; Hier wordt `v` door 4 gedeeld omdat de maximale waarde in een byte 255 kan zijn. Met de return opdracht gaat het programma terug naar zijn hoofdprogramma.

```
int delayVal()
{
    int v;                // maak een tijdelijke variabele 'v'
    v = analogRead(pot); // lees de potentiometer waarde
    v /= 4;               // converter 0-1023 naar 0-255
    return v;            // return definitieve waarde
}
```

### **{ } krullende haakjes (accolade)**

Krullende haakjes geven het begin of het einde aan van een functieblok zoals je ook tegenkomt bij de `void loop()`. Kijk maar:

```
type functie()
{
    statements;
}
```

Het eerste opende accolade (gekrulde haakje) **{** moet altijd afgesloten worden door een (gekruld gesloten) accolade **}**. Het aantal accolades is dus altijd een even getal. Let daar goed op want één accolade te weinig en een heel programma kan stoppen met werken. Vind dan de ontbrekende accolade maar eens terug.

## **; puntkomma**

Een puntkomma moet gebruikt worden na elke ingevoerde opdracht. Net zoals de gekrulde haakjes zal bij het ontbreken van een puntkomma een error verschijnen. Voorbeeld:

```
int x = 13;    // declareer variable 'x' as the integer 13
```

**Opmerking:** Vaak is het zo dat het ontbreken van een puntkomma er voor zorgt dat de Arduino software niet wil compileren en een error aangeeft op een andere plek dan waar de puntkomma vergeten is. Dat wordt dus lastig zoeken.

## **/\*... \*/ blok commentaar**

Blok commentaar zijn gebieden met tekst die door het programma genegeerd worden. Tussen de /\* en \*/ staat meestal uitleg over het programma of over de code die daar staat. Het mooie van "blok commentaar" is dat het over meerdere regels geplaatst kan worden.

```
/* dit is een blok met commentaar
Vergeet niet het einde van het
Commentaar aan te geven
*/
```

Commentaar wordt nooit mee geprogrammeerd in de microcontroller. Het neemt dus geen geheugenruimte in beslag. Natuurlijk wordt het wel bewaard in het Arduino programma (Windows/Linux/Mac).

## **// regel commentaar**

Een regel die begint met // en eindigt met tekst of code zal op die regel genegeerd worden. Ook dit neemt geen geheugen in de microcontroller in beslag. Code voor in de regel gevolgd door // zal wel uitgevoerd worden alles wat na de // komt op die regel echter niet.

```
// Dit is commentaar op een regel en onderstaande wordt
// niet uitgevoerd omdat het vooraf gaat met //.
// A = B + 3
```

Regel commentaar wordt vaak gebruikt om de genoemde opdrachten uit te leggen.

**Opmerking:** Als een programma niet werkt zoals het zou moeten werken dan is het vaak handig om een gedeelte van het programma uit te schakelen door stukken van je programma te voorzien van //. Je kunt dan stapsgewijs onderzoeken waar de fout zit.



## Variabelen

Een variabele is een manier om een numerieke waarde te bewaren voor later gebruik in het programma. Zoals de naam variabele al aangeeft kan de waarde van een variabele ook regelmatig veranderen. Er bestaan ook zogenaamde constantes. Dat zijn variabelen die constant het zelfde blijven en dus nooit van waarde veranderen. Een variabele moet op een juiste manier gedeclareerd worden. In de code hier onder wordt een variabele gedeclareerd genaamd inputVariabele die vervolgens de waarde krijgt die gemeten wordt op de analoge input pin 2:

```
int inputVariabele = 0;           // declareer een variabele en geef
                                  // die de waarde 0
inputVariabele = analogRead(2); // geef de variabele de waarde die
                                  // gelezen wordt op de analoge pin 2
```

'inputVariabele' is de variabele zelf. Op de eerste regel staat dat er een variabele gedeclareerd wordt van het type int (int is de afkorting voor integer). De tweede regel krijgt de variabele de waarde toegewezen die gemeten wordt op de analoge pin 2. Later in het programma zal er wel wat met die variabele gedaan worden. Vaak zal de variabele getest worden of hij aan bepaalde condities voldoet.

**Een voorbeeld:** De volgende code test of de inputVariabele kleiner is dan 100. Als dat zo dan krijgt inputVariabele de waarde 0. Is de waarde hoger dan 100 dan houdt inputVariabele de waarde die hij al had. In de derde regel zie je dat een pauze gemaakt is die dus alleen maar optreedt als inputVariabele groter of gelijk is aan 100.

```
if (inputVariabele < 100) // test of de variabele kleiner
                          // is dan 100
{
inputVariabele = 0;       // zo ja, dan krijgt hij de waarde 0
}
delay(inputVariabele);   // De waarde van de variabele bepaalt
                          // de pauze
```

**Opmerking:** Geef variabelen een logische naam bijvoorbeeld tiltSensor of pushButton. Bekijken anderen jouw programma dan wordt het al een stuk makkelijker lezen. Je kunt elk woord voor variabelen gebruiken tenminste als het geen naam is die al gebruikt wordt in de Arduino omgeving.

## Variabelen declareren

Alle variabelen moeten eenmalig gedeclareerd worden voordat je ze kunt gebruiken. Er zijn verschillende types zoals int, long, float, etc.

## Variable bereik

Een variabele kan gedeclareerd worden in het begin van het programma voor void setup(). Soms heb je door omstandigheden een variabele in een programma niet nodig. Daarom kun je ook een variabele later in het programma wel of niet aanmaken al naar gelang hij nodig is.

De vaste variabele heet een globale variabele. Een globale variabele is dus een variabele die in een heel programma kunt oproepen. Deze variabele declareer je boven void setup().

Een locale variabele is een variabele die alleen gebruikt kan worden in een stukje van een programma. Het is een tijdelijke variabele. Zo'n stukje kan bijvoorbeeld in de void loop() zitten. De reden dat deze variabelen bestaan is dat de tijdelijk geheugen in beslag nemen en daardoor efficiënter met het geheugen van de microcontroller wordt omgesprongen.

**Opmerking:** Hoe meer variabelen je gebruikt in een microcontroller des te sneller zal het geheugen vol zitten. Dat kan natuurlijk niet de bedoeling zijn.

Het volgende voorbeeld zal duidelijk maken hoe de verschillende variabelen werken:

```
int value;                // 'value' is zichtbaar in het hele
                          // programma
void setup()
{
    // geen setup nodig
}
void loop()
{
    for (int i=0; i<20;)  // 'i' is alleen zichtbaar in de for loop
    {
        i++;              // i = i + 1
    }
    float f;              // 'f' is alleen zichtbaar in de inside
                          // loop
```

Op de volgende bladzijde worden de verschillende type variabelen beschreven.

### **byte**

Byte bewaart een 8-bit numerieke waarde zonder een decimale punt met een bereik van 0-255.

```
byte Button1 = 180; // declareert 'Button1' als een byte type
```

### **int**

Integers zijn primaire datatypes om getallen te bewaren zonder een decimale punt een 16-bit waarde met een bereik van 32767 tot -32768.

```
int Count4 = 1500; // declareert 'Count4' als een integer type
```

**Opmerking:** Een integer variabele kan niet groter zijn dan 32767. Verhoog je 32767 met 1 dan wordt het een negatief getal: -32768.

### **long**

Datatype voor erg grote getallen, zonder een decimale punt (een soort uitgebreide integer) een 32-bit waarde met een bereik van 2,147,483,647 tot -2,147,483,648.

```
long someVariabele = 90000; // declareert 'someVariabele'  
                          // als een long type
```

### **float**

Een datatype voor getallen met een decimale punt. Dus met getallen achter de komma. Floating datatypes nemen meer geheugen in gebruik dan een integer en worden opgeslagen als een 32-bit waarde met een bereik van 3.4028235E+38 tot -3.4028235E+38.

```
float someVariabele = 3.14; // declareert 'someVariabele' als  
                          // een float-point type
```

**Opmerking:** Floating-point getallen zijn of hoeven in principe niet gelijk te zijn als je ze met elkaar vergelijkt. Met het rekenen aan floating getallen heeft de microcontroller veel meer tijd nodig dan bij byte of integer getallen.

## Arrays

Een array is a verzameling van verschillende waarden die benaderd kunnen worden door een indexnummer. Je kunt er elke waarde in kwijt en je kunt het oproepen door de naam van de variabele en de indexnummers. Arrays zijn standaard met nullen gevuld en het eerste indexnummer van een array begint ook met een 0.

```
int myArray[] = {waarde0, waarde1, waarde2...}
```

Het is mogelijk om een array te declareren naar type en grootte en dan later de waarden in te vullen op basis van een index-positie:

```
int myArray[5];          // declareert integer array met 6 positions
myArray[3] = 10;        // vul de 4e index positie met de waarde 10
```

Om de waarde terug te krijgen:

```
x = myArray[3];          // x is nu gelijk aan 10
```

Arrays worden vaak gebruikt in loops waarin tellers zitten die telkens met 1 opgehoogd worden zodat ze makkelijk traceerbaar zijn. Het volgende voorbeeld gebruikt een array om een LED te laten knipperen. Er wordt een loop gebruikt. De teller begint op 0, schrijft die waarde op de index positie in de array flicker[], in dit geval 180 op de PWM pin 10, wacht 200 ms en gaat vervolgens naar de volgende index positie.

```
int ledPin = 10;          // LED on pin 10
byte flicker[] = {180, 30, 255, 200, 10, 90, 150, 60}; // array van 8
void setup()              // different values
{
  pinMode(ledPin, OUTPUT); // vul de OUTPUT pin
}
void loop()
{
  for(int i=0; i<7; i++)   // loop gelijke nummers
  { // of values in array
    analogWrite(ledPin, flicker[i]); // schrijf index waarde
    delay(200);           // pause 200ms
  }
}
```

**Opmerking:** Voor de werking van PWM staat een voorbeeld in de bijlage. Je kunt door pulsen te geven de indruk wekken dat een LED feller of minder fel licht geeft.

## Rekenen

Rekenkundige bewerkingen zijn bijvoorbeeld optellen, aftrekken, vermenigvuldigen en delen. Het is altijd een resultaat van twee getallen (operands). Voorbeelden:

```
y = y + 3;  
x = x - 7;  
i = j * 6;  
r = r / 5;
```

De berekening wordt uitgevoerd afhankelijk van het gekozen datatype. Als er gekozen is voor een integer dan zal het resultaat  $9 / 4 = 2$  zijn in plaats van 2.25. Pas ook op bij rekenkundige bewerkingen dat er een "overflow" error kan komen bij te grote getallen. Een byte kan tot maximaal 255, zodat een variabele die gedeclareerd is als een byte de optelling  $250 + 23$  niet kan onthouden.

Zijn de getallen die je gaat bewerken van twee verschillende types, dan wordt het grootste type gebruikt voor de berekening. Bijvoorbeeld als één van de getallen een integer is en het andere getal een float dan zal de uitkomst een getal zijn van het type float.

Kies dus altijd een type variabele die groot genoeg is voor de gewenste berekening. Zorg dat je weet wat er gebeurt als de gebruikte variabele door een optelling ineens van positief veranderd in negatief. Weet je het niet zeker lees dan een paar pagina's terug hoe je getallen moet declareren. Let echter wel op dat float variabelen veel geheugen in beslag nemen en ook de microcontroller zwaarder belasten (lees: langzamer werken).

## Samengestelde opdrachten

Samengestelde opdrachten voor simpele wiskundige berekeningen kent de Arduino ook. Ze worden veel gebruikt in loops en worden later nog beschreven in deze manual. De meest voorkomende samengestelde opdrachten zijn:

```
x ++      // is hetzelfde als x = x + 1, of verhoog x met +1  
x --      // is hetzelfde als as x = x - 1, of verlaag x met -1  
x += y    // is hetzelfde als as x = x + y, of verhoog x met +y  
x -= y    // is hetzelfde als x = x - y, or of verlaag x met -y  
x *= y    // is hetzelfde als x = x * y, of vermenigvuldig x met y  
x /= y    // is hetzelfde als x = x / y, of deel x met y
```

## Vergelijken van getallen

Variabelen worden vaak met elkaar vergeleken. Op basis daarvan worden er dan beslissingen genomen. Op deze en de volgende pagina's staan daar veel voorbeelden van.

```
x == y      // x is gelijk aan y
x != y      // x is niet gelijk aan y
x < y       // x is kleiner dan y
x > y       // x is groter dan y
x <= y      // x is kleiner of gelijk aan y
x >= y      // x is groter of gelijk aan y
```

## Logische berekeningen

Logische berekeningen zijn vergelijkingen die als uitkomst hebben waar of niet waar (TRUE of FALSE). Er zijn drie logische operaties, AND, OR, en NOT die vaak gebruikt worden in zogenaamde if opdrachten:

Logische AND:

```
if (x > 0 && x < 5)      // waar alleen als beide
                          // vergelijkingen waar zijn
```

Logische OR:

```
if (x > 0 || y > 0)      // waar als één van de twee
                          // vergelijkingen waar zijn
```

Logische NOT:

```
if (!x > 0)              // alleen waar als
                          // vergelijking niet waar is
```

## Constantes

De Arduino taal heeft een aantal voorgedefinieerde waarden, die ook wel constantes worden genoemd. Ze worden gebruikt om een programma makkelijker te kunnen lezen of schrijven. Constantes zitten ook in verschillende groepen.

### **true/false**

Dit zijn Boolean constantes die een logisch niveau vaststellen. False wordt gedefinieerd als een 0, terwijl true wordt gedefinieerd als een 1 of iedere andere waarde anders dan een 0. In Boolean is -1, 2 en -900 gedefinieerd als true.

Voorbeeld:

```
if (b == TRUE);  
{  
    Doe iets;  
}
```

### **high/low**

Deze constantes definiëren een pin niveau van HIGH of LOW en worden gebruikt om digitale pennen te lezen. HIGH is een logische 1 en LOW is een logische 0. Een logische 1 is meestal 5 V, maar er bestaat ook al een Arduino waarbij dat 3,3 V is.

Voorbeeld:

```
digitalWrite(13, HIGH);
```

### **input/output**

Deze constantes worden gebruikt met de opdracht pinMode() om te definiëren of een digitale pin INPUT of OUTPUT moet worden. Voorbeeld:

```
pinMode(13, OUTPUT);    \\Pin 13 is een output
```

## if

De if opdracht test of bepaalde condities bereikt zijn. Denk bijvoorbeeld aan een analog signaal dat een bepaalde waarde bereikt waarbij ingegrepen moet worden. In dat geval moet er iets gebeuren. Die actie moet dan plaats vinden binnen de haakjes (zie het voorbeeld hier onder). Wordt er niet aan de voorwaarde voldaan dan wordt de actie tussen de haakjes overgeslagen.

Voorbeeld:

```
if (waardeVariabele ?? waarde)
{
    Doe iets;
}
```

In het bovenstaande voorbeeld wordt de waardeVariabele vergeleken met een andere waarde. Die waarde kan echter ook een constante zijn zoals genoemd op de vorige pagina.

**Opmerking:** Pas op met het volgende te gebruiken: `if(x=10)`. Deze is technisch gezien juist. Het geeft x de waarde 10 en heeft als resultaat altijd TRUE. Gebruik liever `'=='` zodat bij de opdracht `if(x==10)` getest wordt gelijk is aan de waarde 10 of niet.

Bedenk: bij '=' aan de term gelijk en bij '==' aan de term is gelijk aan.



## if... else

De if... else opdracht maakt het mogelijk hoe dan ook een beslissing te laten nemen. Bijvoorbeeld je meet dat een digitale input pin hoog is in dat geval wil je dat actie\_A start. Is de pin echter laag dan moet actie\_B starten Dat zou er als volgt uit kunnen zien:.

```
if (inputPin == HIGH)
{
  Voer actie_A uit;
}
else
{
  Voer actie_B uit;
}
```

Else kan ook een andere procedure zijn zodat je meerdere testen in dezelfde lus kunt verwerken. Bekijk het volgende voorbeeld eens:

```
if (inputPin < 500)
{
  Voer actie_A uit;
}
else if (inputPin >= 1000)
{
  Voer actie_ B uit;
}
else
{
  Voer actie_C uit;
}
```

**Opmerking:** Kijk goed naar de haakjes en de puntkomma's dat wil op deze wijze best wel eens ingewikkeld worden.

## for

Het for commando wordt gebruikt om een aantal commando's een bekend aantal keren te laten herhalen. Via een teller wordt bijgehouden hoe vaak de lus zich moet herhalen. Het commando ziet er als volgt uit:

```
for (variabele; conditie; expressie)
{
    doeiets;
}
```

Dat lijkt lastig, maar het is een makkelijke en vaak gebruikte opdracht. Een voorbeeld:

```
for (int i=0; i<20; i++)          // declareer i, en test of
                                  // het kleiner is
{
    digitalWrite(13, HIGH);      // dan 20, I wordt met 1 opgehoogd
    delay(250);                  // zet pin 13 aan
    digitalWrite(13, LOW);       // zet pin 13 uit
    delay(250);                  // 1/4 second pauze
}
```

In de eerste regel wordt *i* gedeclareerd en wordt meteen getest of *i* kleiner is dan 20. Daarna wordt *i* met 1 verhoogd (en krijgt de waarde 1). Aangezien *i* 0 was wordt alle code die er onder staat tussen de haakjes uitgevoerd. Nadat dat is gedaan wordt regel 1 opnieuw uitgevoerd. *i* heeft nu de waarde 1 er wordt weer getest of *i* kleiner is dan 20, hetgeen nog steeds het geval is en *i* wordt met 1 verhoogd zodat *i* de waarde 2 krijgt. Dat blijft zich herhalen tot *i* de waarde 20 bereikt. Daarna wordt de code tussen de haakjes niet meer uitgevoerd.

**Opmerking:** In C is de for loop veel meer flexibeler in te vullen dan in sommige andere computer talen zoals bijvoorbeeld BASIC. De variabele, conditie en expressie kun je naar wens aanpassen. Let op: ze worden wel gescheiden door een puntkomma.

## while

De while loop heeft wel wat weg van de for loop. Hij is gemakkelijk uit te leggen met: Zolang je aan die voorwaarde voldoet moet je dat doen. Die voorwaarde zou bijvoorbeeld het testen van een sensor kunnen zijn. De loop stopt pas als hij niet meer aan de voorwaarde voldoet. Een voorbeeld:

```
while (someVariable ?? value)
{
    doe iets;
}
```

Het volgende voorbeeld test of de someVariabele kleiner is dan 200. Als dat waar is blijft de lus zich herhalen totdat someVariabele niet langer kleiner is dan 200.

```
while (someVariabele < 200)    // test of someVariabele kleiner
                               // is dan 200
{
    Doe iets;                  // voer programmacode uit
    someVariabele++;           // verhoog variabele met 1
}
```

## do... while

De do loop loop die grotendeels hetzelfde werkt als de while loop. Het verschil zit hem in het feit dat de conditie onderaan staat in plaats van bovenaan, zoals bij de while loop. Ongeacht de condities wordt de loop altijd 1 keer doorlopen.

```
do
{
    doeiets;
} while (someVariable ?? value);
```

In het volgende voorbeeld wordt x hetzelfde als de waarde readSensors(), daarna volgt een pauze van 50 milliseconde, waarna de loop zich herhaalt totdat x is niet meer kleiner dan 100:

```
do
{
    x = readSensors();        // x krijgt de waarde readSensors()
    delay (50);               // pauze 50 milliseconde
} while (x < 100);           // herhaal totdat x is kleiner dan100
```

## **pinMode(pin, mode)**

Wordt gebruikt in de void setup() om een specifieke pin te configureren als een INPUT of een OUTPUT.

```
pinMode(pin, OUTPUT); // sets 'pin' to output
```

Arduino's digitale pinnen zijn standaard geconfigureerd als inputs. Je hoeft ze dus niet per sé te declareren als inputs met pinMode(). Pinnen die geconfigureerd zijn als INPUT bevinden zich in een hoogohmige toestand.

Er zitten ook hoogohmige weerstanden (pullup) van  $20\text{K}\Omega$  ingebouwd in de Atmega chip die bereikt kunnen worden door de Arduino software. Dat kan op de volgende manier:

```
pinMode(pin, INPUT); // maak van 'pin' een input
digitalWrite(pin, HIGH); // schakel op de 'pin' de
//pullup weerstanden in
```

Pullup weerstanden worden normaal gesproken gebruikt bij met schakelaars die op de inputs aangesloten worden.

Pinnen die geconfigureerd zijn als OUTPUT staan in een laagohmige toestand en kunnen maximaal 40 mA leveren. Dit is ruim genoeg voor een LED, maar veel te weinig voor een motor of bijvoorbeeld een relais.

Kortsluiting tussen verschillende poorten kunnen de poort of de gehele Atmega chip onherstelbaar beschadigen. In dat geval moet de chip vervangen worden (inclusief een nieuwe bootloader).

Belangrijk: Het zou niet verkeerd zijn om outputs te beveiligen met een weerstand van  $220\Omega$ . Bij kortsluiting loopt er dan een stroom van  $I = U/R = 5/220 = 22\text{ mA}$  hetgeen kleiner is dan de eerder genoemde 40 mA.

## digitalRead(pin)

Leest de waarde uit van een specifieke pin met als resultaat HIGH of LOW. De pin is gespecificeerd al een variabele of een constante (0-13).

```
Value = digitalRead(Pin);      // maak 'value' gelijk aan  
                                // de input pin
```

## digitalWrite(pin, value)

Schrijf de waarde naar een specifieke pin met als niveau HIGH of LOW. De pin is gespecificeerd als een variabele of een constante (0-13).

```
digitalWrite(pin, HIGH);      // maak 'pin' hoog
```

Het volgende voorbeeld leest een drukknop uit die is aangesloten op pin 7 en laat een LED, aangesloten op pin 13 aan gaan als de drukknop ingedrukt is:

```
int led = 13;                // LED op pin 13  
int pin = 7;                // drukknop op pin 7  
int value = 0;              // variabele value  
void setup()  
{  
    pinMode(led, OUTPUT);    // maak van pin 13 een output  
    pinMode(pin, INPUT);    // maak van pin 7 een input  
}  
void loop()  
{  
    value = digitalRead(pin); // maak van 'value' de input pin  
    digitalWrite(led, value); // zet 'value' over naar de 'led'  
}
```

## analogRead(pin)

Leest de waarde van een specifieke analoge pin in een 10 bit resolutie. Deze functie werkt alleen op pin 0 t/m 6 (Geldt natuurlijk niet voor de Arduino mega). De uitkomst is een integer waarde tussen 0 to 1023.

```
waarde = analogRead(pin); // maak van 'waarde' wat gelezen
                          // wordt op de 'pin'
```

**Opmerking:** Analoge pinnen hoeven niet te worden gedeclareerd als INPUT of OUTPUT. Het zijn automatisch al digitale inputs.

## analogWrite(pin, value)

Schrijft een pseudo-analoge waarde door gebruik te maken van hardwarematige puls-breedte (width) modulatie (PWM) naar een output pin die gemarkeerd is als PWM. Op nieuwere Arduinos met de ATmega168/368 chip, werkt deze functie op pin 3, 5, 6, 9, 10, and 11. Op oudere Arduinos met een ATmega8 werkt deze functie alleen op pin 9, 10, en 11. De waarde kan gespecificeerd worden al een variabele of constante met een bereik van 0-255.

```
analogWrite(pin, waarde); // schrijf 'waarde' naar analoge 'pin'
```

Een waarde van 0 geeft 0 V als output op de gespecificeerde pin. Een waarde van 255 geeft 5 V als output op de gespecificeerde pin. Voor waarden tussen 0 en 255 vindt er een evenredige pulsbreedte modulatie plaats, waarbij opgemerkt moet worden dat de breedte van de pulsen evenredig groter wordt naarmate de waarde stijgt.

Omdat dit een hardware functie is zal de uitgegeven pulsbreedte continue het zelfde zijn totdat er een nieuwe analogWrite wordt gedaan.

Het volgende voorbeeld leest een analoge waarde van een analoge INPUT pin. Vervolgens wordt deze waarde aangeboden aan een PWM OUTPUT pin. Let op de gelezen waarde is van 0 – 1023 maar de maximale aangeboden PWM waarde mag niet meer zijn dan 255. Daarom wordt de gelezen waarde gedeeld door 4:

```
int led = 10;           // LED met 220 weerstand op pin 10
int pin = 0;           // potentiometer op analoge pin 0
int value;             // value om te lezen
void setup(){}        // geen setup nodig
void loop()
{
    value = analogRead(pin); // lees 'value' op 'pin'
    value /= 4;             // converteer 0-1023 to 0-255
    analogWrite(led, value); // outputs PWM signaal naar led
}
```

## **delay(ms)**

Wachtlus (pauze) weergegeven in milliseconden, waarbij de waarde 1000 gelijk staat aan 1 seconde.

```
delay(1000); // wacht een seconde
```

## **millis()**

Laat zien hoeveel milliseconde het Arduino board in werking is na de start van het lopende programma. De weergave is een long variabele.

```
Looptijd = millis(); // looptijd wordt gevuld met millis()
```

**Note:** Het weer te geven getal zal na verloop van tijd een overflow veroorzaken. (Een reset naar nul), na ongeveer 9 uur.

## **min(x, y)**

Bereken het minimum van twee getallen en geef het kleinste getal weer.

```
waarde = min(getal, 100); // 'waarde' is gelijk aan 100
                          // of kleiner dan 100 als 'getal'
                          // kleiner dan 100 is
```

## **max(x, y)**

Bereken het maximum van twee getallen en geef het grootste getal weer.

```
waarde = min(getal, 100); // 'waarde' is gelijk aan 100
                          // of hoger dan 100 als 'getal'
                          // hoger dan 100 is
```

## **randomSeed(seed)**

Maak een willekeurige waarde aan (random).

```
randomSeed(value);
```

De Arduino kan uit zichzelf geen random nummer creëren.

Daarvoor is een commando dat wel een “willekeurige” random waarde kan aanmaken. Let wel: Er is nooit sprake van een absolute willekeurige waarde. Het is een functie om te helpen om één of meerdere “willekeurige” waardes aan te maken. De syntax is:

## **RandomSeed()**

**random(max)**

**random(min, max)**

De random functie maakt het ook mogelijk om waardes in een reeks aan te maken:

```
value = random(100, 200);           // sets 'value' to a random
                                     // number between 100-200
```

**Opmerking:** Laat bovenstaande code vooraf laten met de randomSeed() functie.

Het volgende voorbeeld maakt een willekeurige waarde aan tussen 0-255 en zet de aangemaakte waarde over naar een PWM pin:

```
int randomNumber;           // variabele om random waarde te bewaren
int led = 10;               // LED met 220Ω weerstand op pin 10
void setup() {}            // geen setup nodig
void loop()

{
  randomSeed(millis());     // gebruik millis()
  randomNumber = random(255); // random nummer van 0-255
  analogWrite(led, randomNumber); // output PWM signaal
  delay(500);               // wacht halve seconde
}
```



### **Serial.begin(rate)**

Open een seriële poort, zet de juiste baudrate om seriële data te kunnen verzenden. Een baudrate van 9600 wordt veel gebruikt maar andere snelheden zijn ook mogelijk.

```
void setup()
{
  Serial.begin(9600);    // open een seriële port
}                       // met een baudrate van 9600 bps
```

**Opmerking:** Wanneer gebruik wordt gemaakt van seriële communicatie op pin 0 (Rx) en pin 1 (Tx) let er dan op dat deze niet tegelijkertijd gebruikt kunnen worden.

### **Serial.println(data)**

Stuurt data naar de seriële poort tezamen met een carriage return en een line feed. Dit commando heeft dezelfde vorm als het commando Serial.print().

```
Serial.println(analogValue);    // zend de waarde van een
                                // analoge waarde 'analogValue'
```

**Opmerking:** Meer informatie over de verschillende van de Serial.println() en Serial.print() functies kijk op de Arduino website. [www.arduino.cc](http://www.arduino.cc) en [www.arduino.nu](http://www.arduino.nu).

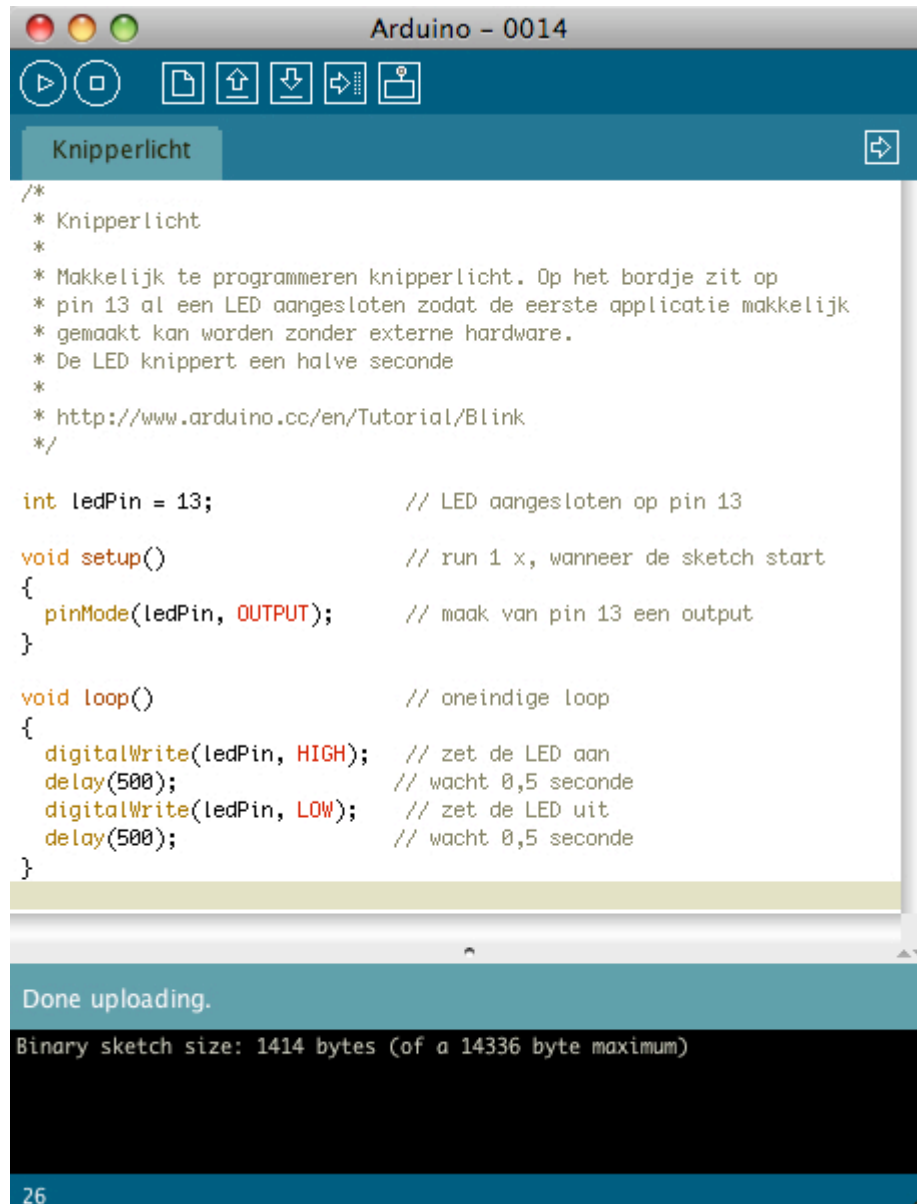
Het volgende voorbeeld leest de analoge waarde op pin 0 en zendt deze data elke seconde naar de computer.

```
void setup()
{
  Serial.begin(9600);          // open poort naar 9600bps
}

void loop()
{
  Serial.println(analogRead(0)); // zend analoge waarde
  delay(1000);                 // wacht 1 seconde
}
```

## Bijlage

Een simpel programma is het laten knipperen van een LED. Dat gaat bij de Arduino erg gemakkelijk omdat op de digitale ingangen op pin 13 al een LED is aangesloten. Simpelweg een USB-kabel aansluiten en het onderstaande programma invoeren en de LED knippert.



The screenshot shows the Arduino IDE interface. The title bar reads "Arduino - 0014". The menu bar includes "Knipperlicht" with a refresh icon. The main text area contains the following code:

```
/*
 * Knipperlicht
 *
 * Makkelijk te programmeren knipperlicht. Op het bordje zit op
 * pin 13 al een LED aangesloten zodat de eerste applicatie makkelijk
 * gemaakt kan worden zonder externe hardware.
 * De LED knippert een halve seconde
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */

int ledPin = 13;           // LED aangesloten op pin 13

void setup()               // run 1 x, wanneer de sketch start
{
  pinMode(ledPin, OUTPUT); // maak van pin 13 een output
}

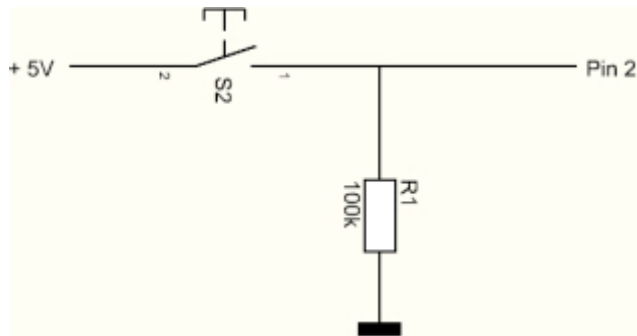
void loop()                // oneindige loop
{
  digitalWrite(ledPin, HIGH); // zet de LED aan
  delay(500);                 // wacht 0,5 seconde
  digitalWrite(ledPin, LOW);  // zet de LED uit
  delay(500);                 // wacht 0,5 seconde
}
```

Below the code editor, a status bar shows "Done uploading." and "Binary sketch size: 1414 bytes (of a 14336 byte maximum)". The page number "26" is visible in the bottom left corner of the IDE window.

## Schakelaar

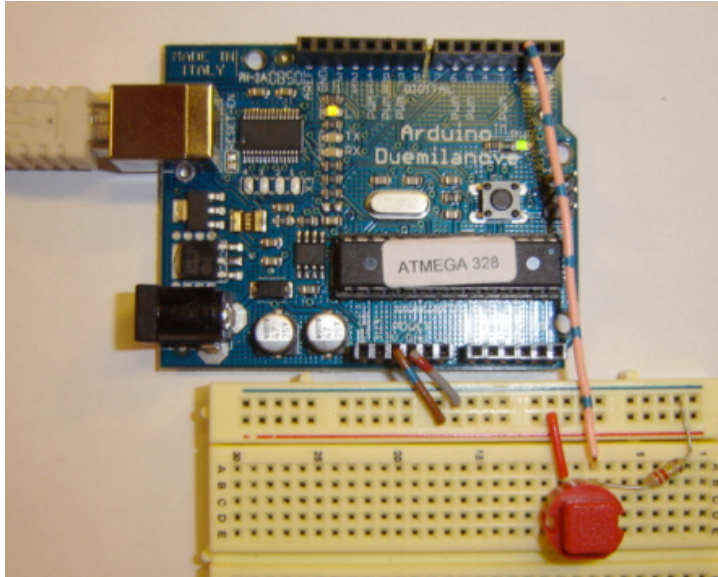
Met een schakelaar (drukknop) wordt de LED uitgezet die is aangesloten op poort 13 de Arduino.

Het schema:



Zoals je ziet is pin2 via een weerstand van 100 K verbonden aan de massa. Dat is gedaan omdat, als de schakelaar niet gesloten is, je niet weet welke spanning er op pin twee staat. Nu is dat in ieder geval 0 Volt. R1 noem je ook wel een pulldown-weerstand.

Het aansluiten:



Het bijbehorende programma:



```
Arduino - 0014
File Edit Sketch Tools Help
Button $
/*
 * Schakelaar
 *
 * Zet een LED aan die is aangesloten op pin 13 en zodra
 * de schakelaar (pin 2) wordt ingedrukt gaat de Led uit.
 *
 * http://www.arduino.cc/en/Tutorial/Button
 */

int ledPin = 13;           // LED aangesloten op pin 13
int inputPin = 2;         // schakelaar aangesloten op pin2
int val = 0;              // variable voor het lezen van de pin status

void setup() {
  pinMode(ledPin, OUTPUT); // pin 13 output
  pinMode(inputPin, INPUT); // pin 2 input
}

void loop(){
  val = digitalRead(inputPin); // lees waarde input
  if (val == HIGH) {          // check of input is HIGH
    digitalWrite(ledPin, LOW); // zet de LED uit
  } else {
    digitalWrite(ledPin, HIGH); // zet de LED aan
  }
}

Done uploading.
Binary sketch size: 1488 bytes (of a 30720 byte maximum)

12
```

Zoals je ziet zie je hier een **if - then - else** commando.

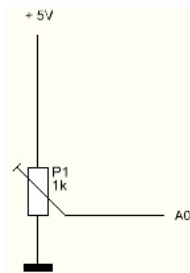
Daar staat dus:

Als (if) val hoog is zet dan (then) de LED uit en anders (else) zet de LED aan.

## Analoge ingangen

Op het Arduino bordje zitten 6 analoge ingangen. Let op geen analoge uitgangen!!  
Op één analoge ingang gaan we de spanning variëren. We zorgen er voor dat het programma deze waarde uitleest en we laten een LED knipperen al naar gelang de hoogte van de spanning.

Het schema:



Let goed op, want A0 is niet de pin 0, maar A0 van de Analoge inputs. Een LED wordt niet aangesloten, we maken gebruik van de LED die in serie met een weerstand aangesloten is op pin13.

Het programma:

```
Arduino - 0014
File Edit Sketch Tools Help
Analoog1 $
#define LED 13 // De LED is aangesloten op pin13

int analogewaarde = 0; // in analogewaarde staat de waarde van
                        // de potmeter

void setup()
{
  pinMode(LED, OUTPUT); // LED is een output
}

void loop()
{
  analogewaarde = analogRead(A0)/4; // lees de analoge waarde op pin0 en deel
                                    // deze door 4 (/4 geeft een beter resultaat)
  digitalWrite(LED, HIGH); //zet de LED aan gedurende...
  delay(analogewaarde); //de ingelezen analogewaarde

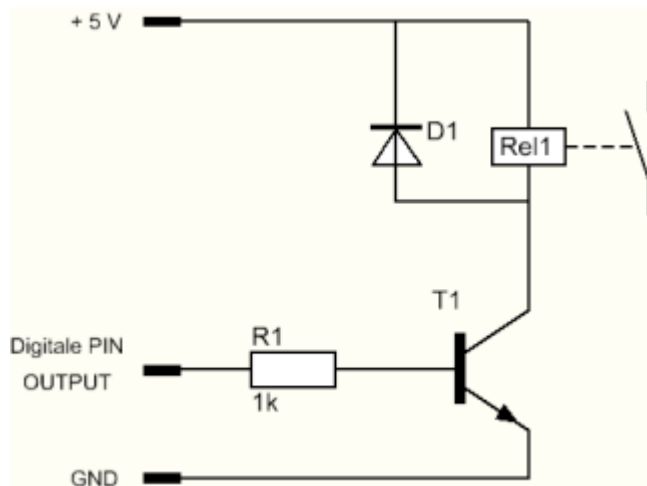
  digitalWrite(LED, LOW); //zet de LED aan gedurende...
  delay(analogewaarde); //de ingelezen analogewaarde
}

Done uploading.
Binary sketch size: 1506 bytes (of a 30720 byte maximum)

21
```

## Het gebruik van Relais met de Arduino

De Arduino kan per (digitale) poort een maximale stroom leveren van 40 mA. Dat is voor een microcontroller redelijk veel. er zijn ook types die maar 5 mA kunnen leveren per poort. Wanneer je een lamp, motor of andere elektrische grootgebruiker wilt aansturen die meer stroom nodig heeft dan 40 mA dan moet je zoeken naar een andere oplossing. Die vind je meestal in een relais. Aangezien een relais ook vaak meer dan 40 mA nodig heeft moet je een stroomversterker gebruiken in de vorm van een transistor. Zie daarvoor het schema hier onder.



Alle voedingspanningen (lees ook stroom) worden gehaald uit de voedingsbron uit de Arduino. Over het relais staat een zogenaamde antiparalleldiode die voorkomt dat de transistor stuk gaat bij het uitschakelen van het relais.

Gebruik de volgende componenten:

**R1 = 1 K**

**T1 = BC547B of BC547C**

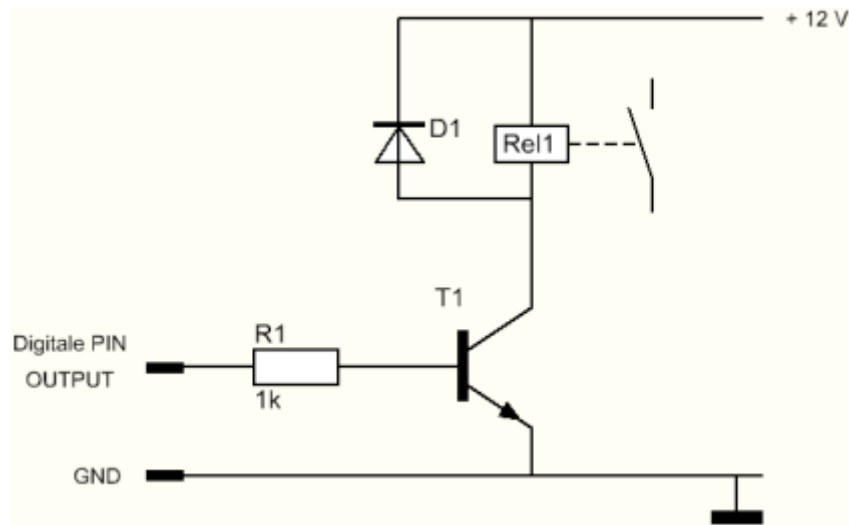
**D1 = 1N4004**

**RE1 = 5 V Relais**

De BC547 kan een maximale stroom leveren van 100 mA. Meer stroom nodig gebruik dan in plaats van een BC547 een BD137. deze laatste kan een stroom leveren van, mits gekoeld, 1 A.

### Externe voeding:

Wordt er gebruik gemaakt van een externe voeding hanteer dan het volgende schema:



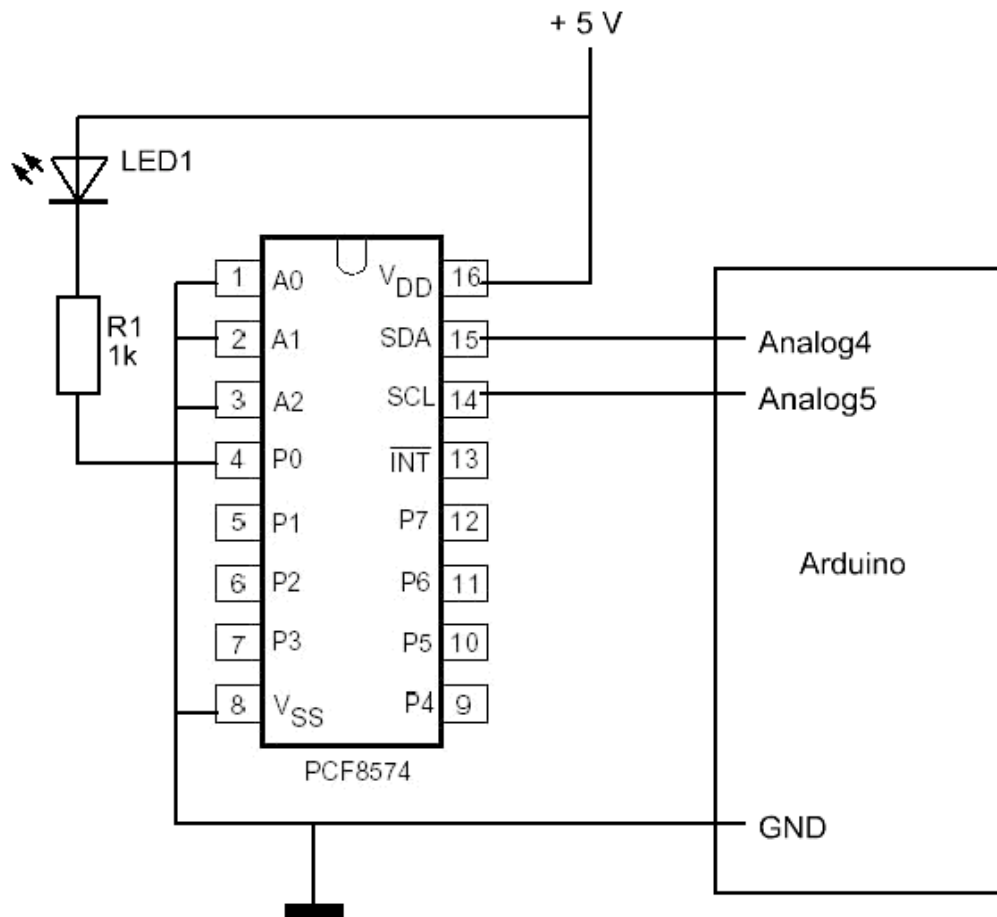
In het voorbeeld is gebruik gemaakt van een externe voeding van 12 V. Deze waarde kan anders zijn. Alle onderdelen zijn identiek aan de hier beschreven onderdelen behalve dat natuurlijk het relais een 12 V type is.

**Let op: De massa van de Arduino moet ALTIJD verbonden zijn aan de massa van de externe voeding.**

## De Arduino en I2C

I2c is een protocol dat is uitgevonden door Philips in de jaren 80. Het was oorspronkelijk bedoeld als bussysteem in bijvoorbeeld in een videorecorder. Je kunt over twee draden data verzenden en ontvangen.

Met een IC, de PCF8574, kun je via i2c totaal 8 uitgangen aansturen. We hebben daarvoor het volgende schema gebouwd:



De Arduino communiceert via de analoge poorten 4 en 5 serieel met de I/O expander PCF8574 met respectievelijk SDA en SCL op pen 15 en 14 op het IC zelf. Het hieronder gepresenteerde programma laat P0 t/m P7 continue knipperen. Voor het gemak is in het schema maar 1 LED aangesloten. Je kunt er zelf voor kiezen om op P1 tot en met P7 ook een LED in serie met een weerstand te plaatsen. Belangrijk is in ieder geval dat de massa van de PCF8574 doorverbonden is met de massa (GND) van de Arduino.



Het programma:



```
Arduino - 0014
i2c §
/*
  Test programma voor PCF8574 I2C I/O expander
  Laat alle poorten hoog en laag knipperen.
*/
#include <Wire.h>
#define expander B01000000 // Adres met alle drie de adreslijnen aan massa. B0111000
// Opmerking Het R/W bit is geen onderdeel van dit adres

void setup()
{
  Wire.begin();
  Serial.begin(9600);
}

void loop()
{
  Serial.println("Writing B00000000.");
  expanderWrite(B00000000);
  Serial.print("Read: ");
  Serial.println(expanderRead(), BIN);
  delay(1000);
  Serial.println("Writing B11111111.");
  expanderWrite(B11111111);
  Serial.print("Read: ");
  Serial.println(expanderRead(), BIN);
  delay(1000);
}

void expanderWrite(byte _data )
{
  Wire.beginTransmission(expander);
  Wire.send(_data);
  Wire.endTransmission();
}

byte expanderRead()
{
  byte _data;
  Wire.requestFrom(expander, 1);
  if(Wire.available())
  {
    _data = Wire.receive();
  }
  return _data;
}

Done Saving.
48
```

## Een servo-motor met de Arduino

Met de Arduino kun je gemakkelijk Servo's aansturen. In het hier getoonde voorbeeld krijg een servomotor zijn positie doorgegeven via een potentiometer. De Arduino leest deze potentiometer uit en geeft dat vervolgens door aan de servomotor. De servomotor zet zichzelf vervolgens in de juiste positie.

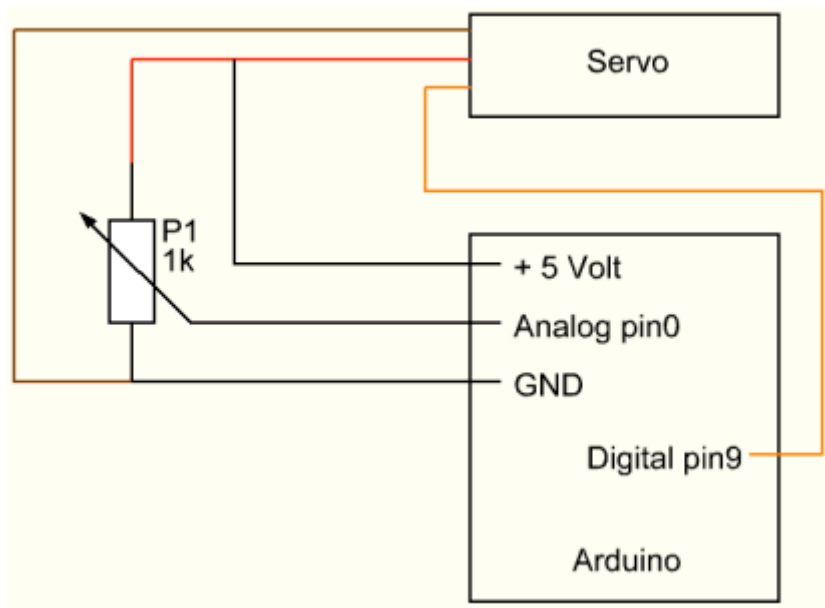
Wij maken gebruik van de RS-2 servomotor van de firma Conrad. Dit is een goedkope servo waar wij prima ervaringen mee hebben. Je kunt hem maximaal 180 graden laten draaien. Wil je hem als motor gebruiken dan moet je de servo modificeren. De servo-aansluitingen zijn als volgt:



Rood	+ 5 Volt
Bruin	Massa
Oranje	Stuurpin

Wil je de servo niet modificeren dan levert Antratek een servo gemaakt door de firma Parallax die dat al is. Overigens ook een prima product.

Het schema is erg eenvoudig. De loper van de potentiometer sluit je aan op de analoge pin 0 van de Arduino en de oranje draad van de servo sluit je aan op de digitale pin 9 van de Arduino. Daarna alleen nog even de +5V en de massa aansluiten op respectievelijk de potentiometer en de servo. Let bij de servo goed op dat je de + en - niet verkeerd aansluit. Kijk dus goed naar de kleuren!!



Nu nog de Arduino programmeren en de servo werkt.

```

Arduino - 0015
File Edit Sketch Tools Help
Servobesturing
// Een Servo besturen met een potmeter
//

#include <Servo.h>

Servo myservo; // maak een servo object voor de besturing

int potpin = 0; // op de analoge pin0 sluit je de potmeter aan
int val; // val = variabele

void setup()
{
  myservo.attach(9); // sluit pin9 aan op de servo
}

void loop()
{
  val = analogRead(potpin); // lees de analoge waarde op pin0 (0 en 1023)
  val = map(val, 0, 1023, 0, 179); // verdeel dat naar een waarde tussen 0 en 180)
  myservo.write(val); // zet deze waarde over naar de servo
  delay(15); // wacht even totdat de servo op zijn positie is
}

Done uploading.
Binary sketch size: 1988 bytes (of a 30720 byte maximum)

23

```

Het programma (de sketch) is simpel. Iedere keer wordt de stand van de potmeter uitgelezen en wordt de gelezen waarde overgezet naar de servo.

De servo krijgt na elke correctie 15 mS de tijd om zichzelf in te stellen (lees naar de juiste positie te komen).

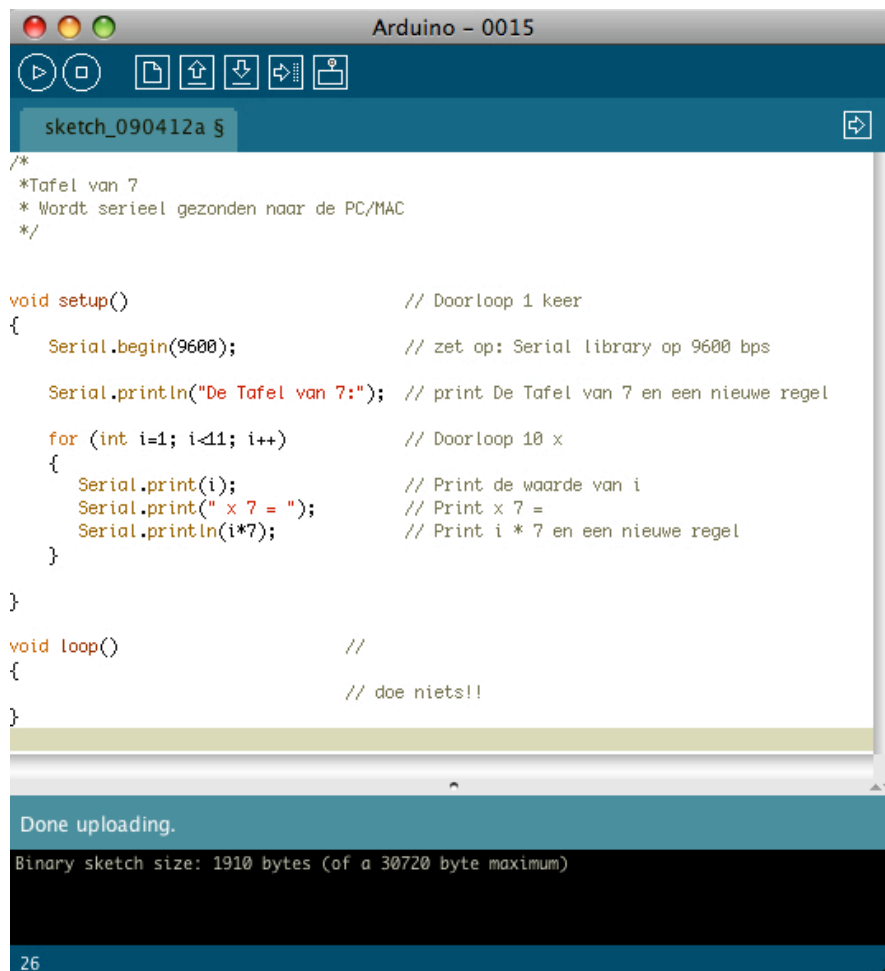
Pen9 van de digitale output is natuurlijk een PWM pen.

**Let op:** Een gemodificeerde servo zal de stand van de potmeter niet meenemen. In dat geval zal de servo voor of achteruit draaien.

## Seriële communicatie naar de MAC/PC

Soms kan het handig zijn om directe communicatie vanuit de Arduino direct terug te koppelen naar de PC. Bij de Arduino kan dit alleen door serieel data te verzenden naar de PC.

Kijk maar eens naar het volgende voorbeeld:



```
Arduino - 0015
sketch_090412a §

/*
 *Tafel van 7
 * Wordt serieel gezonden naar de PC/MAC
 */

void setup()                // Doorloop 1 keer
{
  Serial.begin(9600);       // zet op: Serial library op 9600 bps

  Serial.println("De Tafel van 7:"); // print De Tafel van 7 en een nieuwe regel

  for (int i=1; i<=11; i++) // Doorloop 10 x
  {
    Serial.print(i);        // Print de waarde van i
    Serial.print(" x 7 = "); // Print x 7 =
    Serial.println(i*7);    // Print i * 7 en een nieuwe regel
  }
}

void loop()                 //
{
  // doe niets!!
}
```

Done uploading.  
Binary sketch size: 1910 bytes (of a 30720 byte maximum)

26

In het voorbeeld wordt de tafel van 7 geprint. Niet op papier, maar op het scherm.

In de void setup() wordt eerst een seriële poort op 9600 bps gezet met het commando Serial.begin(9600);

Vervolgens print de Arduino de tekst "De tafel van 7", en omdat er in het commando serial.println("De Tafel van 7:"); in staat wordt alles daarna op een nieuwe regel geprint. Met de for-opdracht wordt vervolgens de tafel van 7 uitgeprint. Standaard krijg je dat niet te zien. Je moet eerst de serial monitor openen. Dat doe je

door op het symbool:  te klikken. Dat symbool zie je naast de upload-knop.

De uitkomst van de tafel van 7 zie je hier onder.

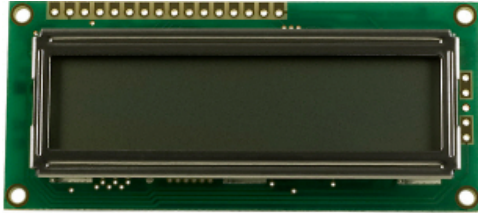


```
9600 baud
De Tafel van 7:
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
26
```

Dit is de weergave van de data die de Arduino naar de PC of MAC gestuurd heeft. Deze functie kun je ook gebruiken als een programma niet goed werkt. Het is dan mogelijk om tussentijds verschillende variabelen te testen op wat er verwacht werd en vervolgens de verschillen te analyseren.

## Het aansluiten van een LCD display

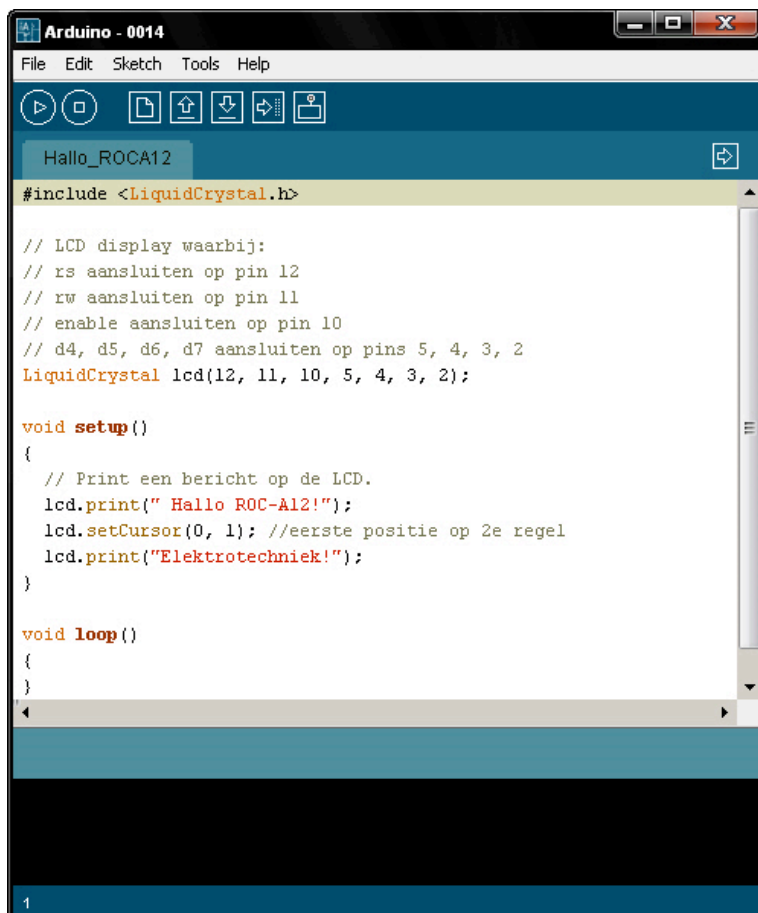
Het aansluiten van een LCD display was vroeger erg ingewikkeld, maar tegenwoordig is dat een fluitje van een cent. De Arduino software kent een flink aantal libraries (bibliotheken) die al het moeilijke werk uit handen nemen.



Kijk maar eens naar het onderstaande programma. Je ziet in het programma bovenin het commando `#include <LiquidCrystal.h>` dat betekent niets anders dan dat de Arduino software straks een beetje extra code mee upload naar de microcontroller zodat je alleen nog maar de tekst hoeft in te typen die je wilt laten zien na het commando `lcd.print("Elektrotechniek!");`

Je gebruikt een normaal 4 bits (geen seriëel) display zodat het commando `LiquidCrystal` wil weten waar de pennen `rs`, `rw`, `d4` t/m `d7` zijn aangesloten. Wil je meer weten van displays bekijk dan eens een datasheet.

Het commando `lcd.setCursor(0, 1);` zorgt er voor dat de nieuw te plaatsen tekst op de tweede regel komt. Zou je een display hebben met vier regels dan zou de start van de derde regel dus plaats vinden op regel 0,2. Dat klinkt vreemd, maar de positie 0,0 is namelijk de eerste positie op de eerste regel.



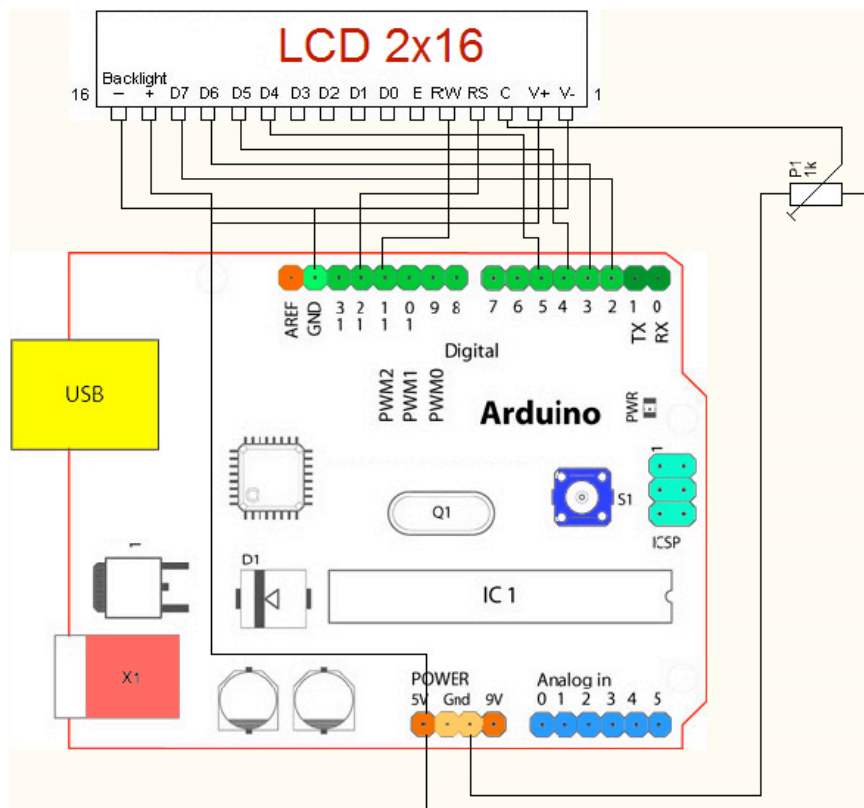
```
Arduino - 0014
File Edit Sketch Tools Help
Hallo_ROCA12
#include <LiquidCrystal.h>

// LCD display waarbij:
// rs aansluiten op pin 12
// rw aansluiten op pin 11
// enable aansluiten op pin 10
// d4, d5, d6, d7 aansluiten op pins 5, 4, 3, 2
LiquidCrystal lcd(12, 11, 10, 5, 4, 3, 2);

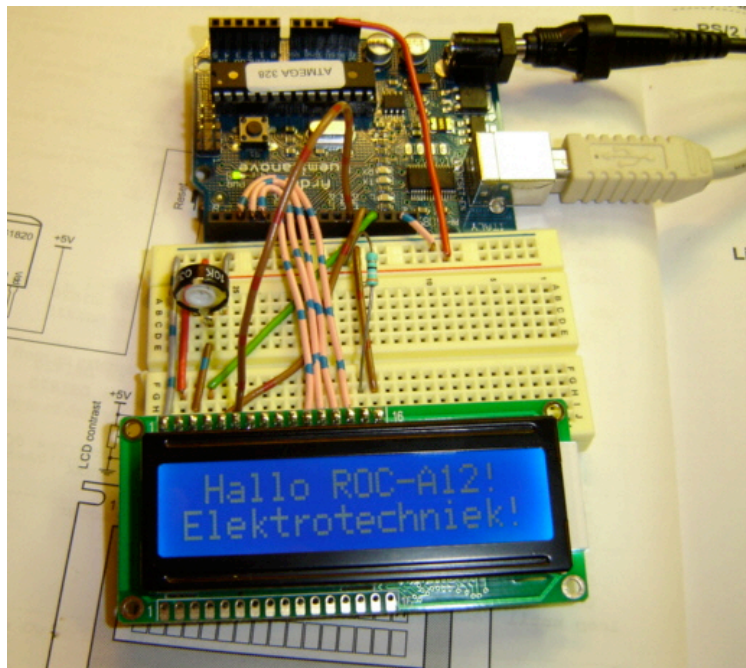
void setup()
{
  // Print een bericht op de LCD.
  lcd.print(" Hallo ROC-A12!");
  lcd.setCursor(0, 1); //eerste positie op 2e regel
  lcd.print("Elektrotechniek!");
}

void loop()
{
}
1
```

Bekijk ook het schema hier onder hoe je het moet aansluiten:

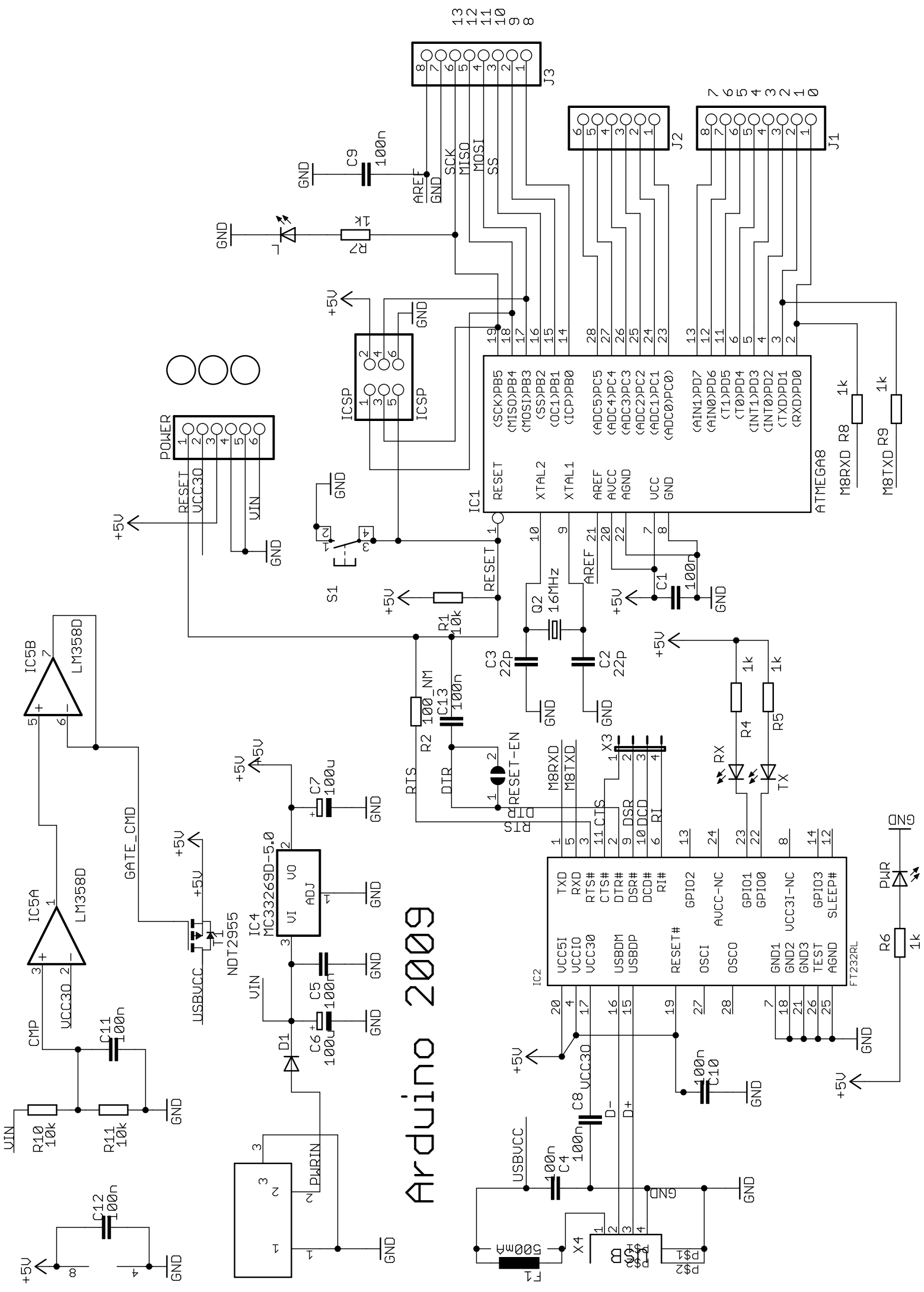


Als je alles goed hebt aangesloten dan krijg je het volgende te zien:



Kijk eens goed naar het display, dan zie je dat deze 16 aansluitpennen heeft. Er zijn ook displays met 14 aansluitpennen. Dat zijn displays waarin geen licht (backlight) in zit, zodat pin 15 en pin 16 niet nodig zijn en dus ook niet op het display zitten. Displays met een backlight zijn doorgaans iets dikker dan een display zonder een backlight.





# Arduino 2009

## Features

- High Performance, Low Power AVR<sup>®</sup> 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32K Bytes of In-System Self-Programmable Flash program memory (ATmega48P/88P/168P/328P)
  - 256/512/512/1K Bytes EEPROM (ATmega48P/88P/168P/328P)
  - 512/1K/1K/2K Bytes Internal SRAM (ATmega48P/88P/168P/328P)
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C<sup>(1)</sup>
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I<sup>2</sup>C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V for ATmega48P/88P/168PV
  - 2.7 - 5.5V for ATmega48P/88P/168P
  - 1.8 - 5.5V for ATmega328P
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - ATmega48P/88P/168PV: 0 - 4 MHz @ 1.8 - 5.5V, 0 - 10 MHz @ 2.7 - 5.5V
  - ATmega48P/88P/168P: 0 - 10 MHz @ 2.7 - 5.5V, 0 - 20 MHz @ 4.5 - 5.5V
  - ATmega328P: 0 - 4 MHz @ 1.8 - 5.5V, 0 - 10 MHz @ 2.7 - 5.5V, 0 - 20 MHz @ 4.5 - 5.5V
- Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48P/88P/168P:
  - Active Mode: 0.3 mA
  - Power-down Mode: 0.1  $\mu$ A
  - Power-save Mode: 0.8  $\mu$ A (Including 32 kHz RTC)

Note: 1. See "Data Retention" on page 7 for details.



**8-bit AVR<sup>®</sup>  
Microcontroller  
with 4/8/16/32K  
Bytes In-System  
Programmable  
Flash**

**ATmega48P/V\*  
ATmega88P/V\*  
ATmega168P/V  
ATmega328P\*\***

**\*\*Preliminary**

**Summary**

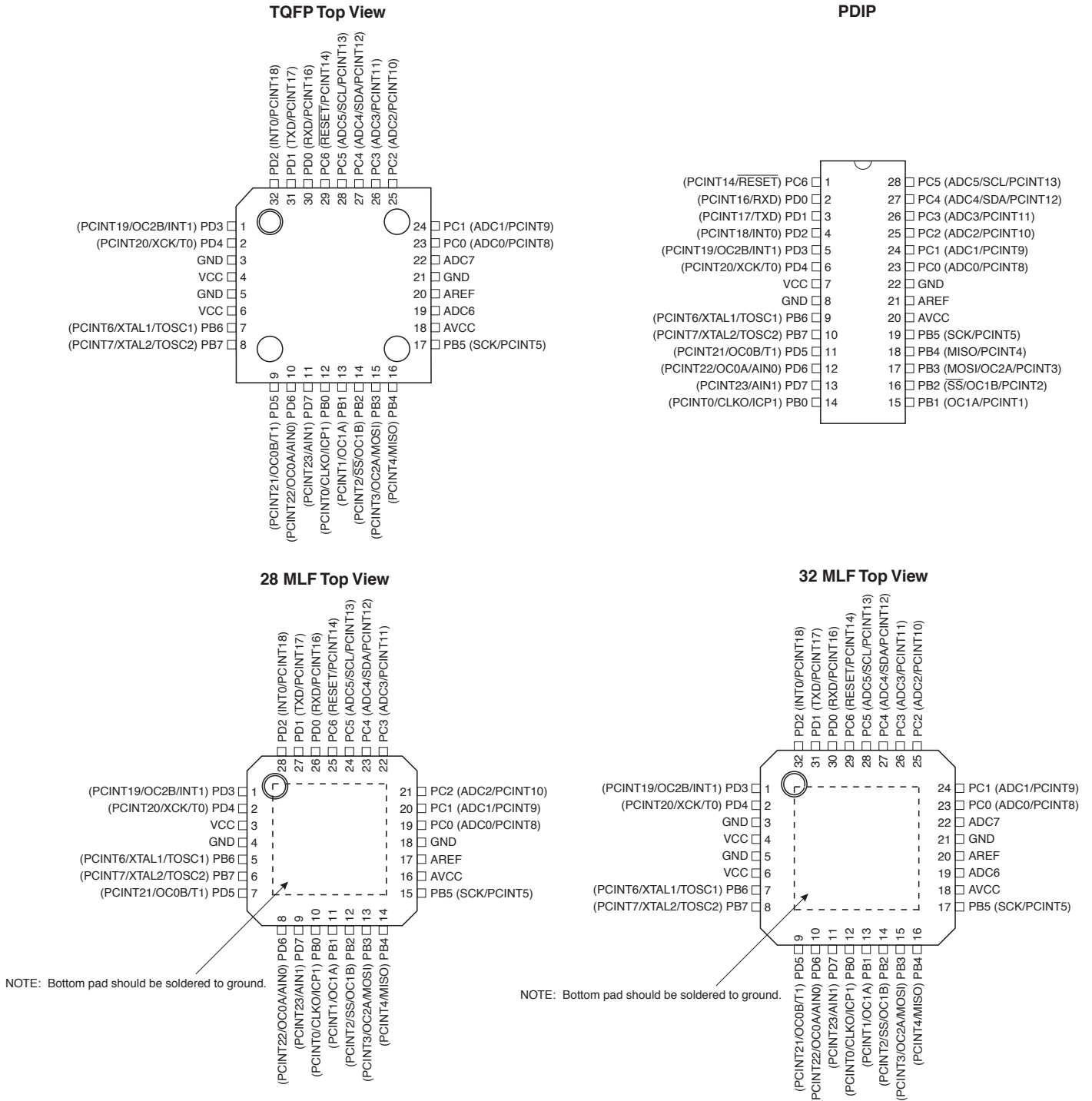
\* Not recommended for new designs.

Rev. 8025I-AVR-02/09



## 1. Pin Configurations

Figure 1-1. Pinout ATmega48P/88P/168P/328P



## 1.1 Pin Descriptions

### 1.1.1 VCC

Digital supply voltage.

### 1.1.2 GND

Ground.

### 1.1.3 Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB7..6 is used as TOSC2..1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

The various special features of Port B are elaborated in ["Alternate Functions of Port B" on page 82](#) and ["System Clock and Clock Options" on page 26](#).

### 1.1.4 Port C (PC5:0)

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5..0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

### 1.1.5 PC6/ $\overline{\text{RESET}}$

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. The minimum pulse length is given in [Table 26-3 on page 320](#). Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated in ["Alternate Functions of Port C" on page 85](#).

### 1.1.6 Port D (PD7:0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

The various special features of Port D are elaborated in ["Alternate Functions of Port D"](#) on page 88.

## 1.1.7 $AV_{CC}$

$AV_{CC}$  is the supply voltage pin for the A/D Converter, PC3:0, and ADC7:6. It should be externally connected to  $V_{CC}$ , even if the ADC is not used. If the ADC is used, it should be connected to  $V_{CC}$  through a low-pass filter. Note that PC6..4 use digital supply voltage,  $V_{CC}$ .

## 1.1.8 AREF

AREF is the analog reference pin for the A/D Converter.

## 1.1.9 ADC7:6 (TQFP and QFN/MLF Package Only)

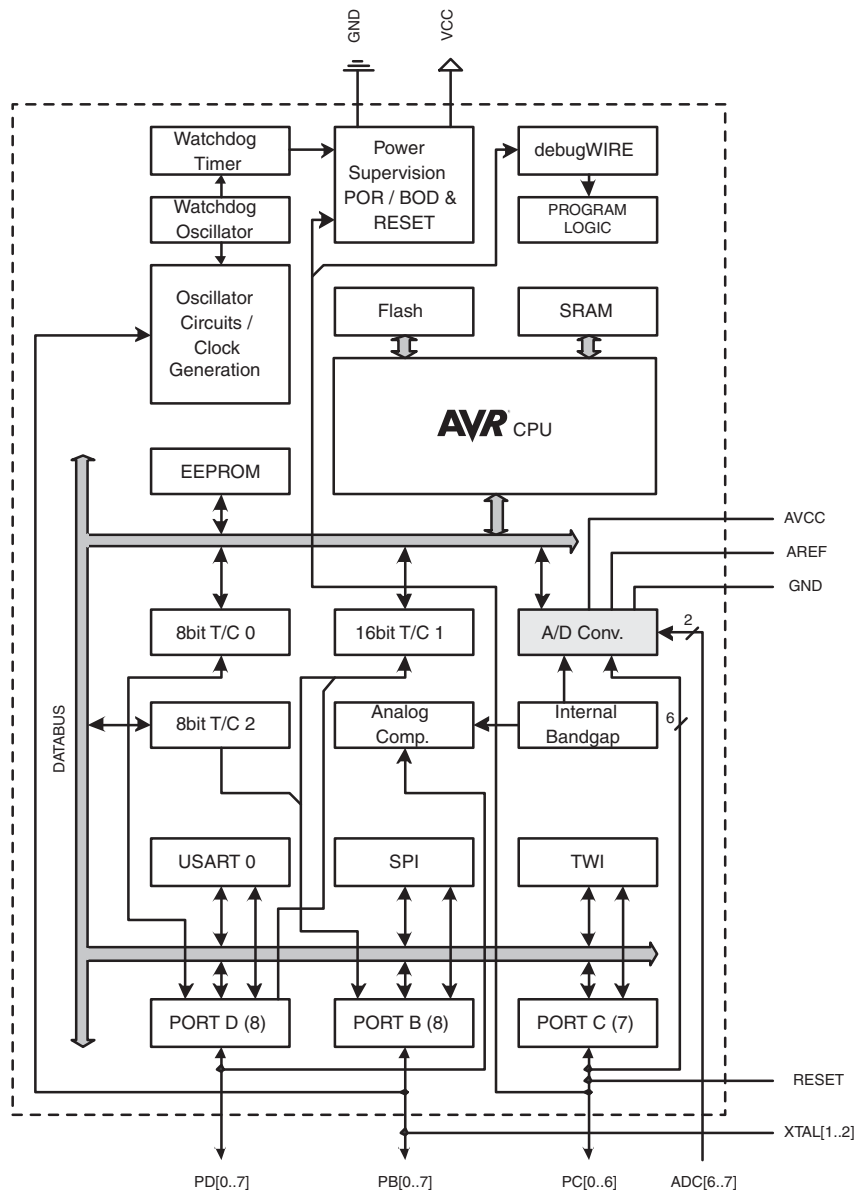
In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

## 2. Overview

The ATmega48P/88P/168P/328P is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega48P/88P/168P/328P achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

## 2.1 Block Diagram

Figure 2-1. Block Diagram



The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega48P/88P/168P/328P provides the following features: 4K/8K/16K/32K bytes of In-System Programmable Flash with Read-While-Write capabilities, 256/512/512/1K bytes EEPROM, 512/1K/1K/2K bytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte-oriented 2-wire Serial Interface, an SPI serial port, a 6-channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages), a programmable

Watchdog Timer with internal Oscillator, and five software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, USART, 2-wire Serial Interface, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption.

The device is manufactured using Atmel's high density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional non-volatile memory programmer, or by an On-chip Boot program running on the AVR core. The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega48P/88P/168P/328P is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega48P/88P/168P/328P AVR is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/Simulators, In-Circuit Emulators, and Evaluation kits.

## 2.2 Comparison Between ATmega48P, ATmega88P, ATmega168P, and ATmega328P

The ATmega48P, ATmega88P, ATmega168P, and ATmega328P differ only in memory sizes, boot loader support, and interrupt vector sizes. [Table 2-1](#) summarizes the different memory and interrupt vector sizes for the three devices.

**Table 2-1.** Memory Size Summary

Device	Flash	EEPROM	RAM	Interrupt Vector Size
ATmega48P	4K Bytes	256 Bytes	512 Bytes	1 instruction word/vector
ATmega88P	8K Bytes	512 Bytes	1K Bytes	1 instruction word/vector
ATmega168P	16K Bytes	512 Bytes	1K Bytes	2 instruction words/vector
ATmega328P	32K Bytes	1K Bytes	2K Bytes	2 instructions words/vector

ATmega88P, ATmega168P, and ATmega328P support a real Read-While-Write Self-Programming mechanism. There is a separate Boot Loader Section, and the SPM instruction can only execute from there. In ATmega48P, there is no Read-While-Write support and no separate Boot Loader Section. The SPM instruction can execute from the entire Flash.

## 3. About

### 3.1 Disclaimer

Typical values contained in this datasheet are based on simulations and characterization of other AVR microcontrollers manufactured on the same process technology. Min and Max values will be available after the device is characterized.

### 3.2 Resources

A comprehensive set of development tools, application notes and datasheets are available for download on <http://www.atmel.com/avr>.

### 3.3 Data Retention

Reliability Qualification results show that the projected data retention failure rate is much less than 1 PPM over 20 years at 85°C or 100 years at 25°C.

### 3.4 Code Examples

This documentation contains simple code examples that briefly show how to use various parts of the device. These code examples assume that the part specific header file is included before compilation. Be aware that not all C compiler vendors include bit definitions in the header files and interrupt handling in C is compiler dependent. Please confirm with the C compiler documentation for more details.

For I/O Registers located in extended I/O map, "IN", "OUT", "SBIS", "SBIC", "CBI", and "SBI" instructions must be replaced with instructions that allow access to extended I/O. Typically "LDS" and "STS" combined with "SBRS", "SBRC", "SBR", and "CBR".



## 4. Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0xFF	Reserved	–	–	–	–	–	–	–	–	
0xFE	Reserved	–	–	–	–	–	–	–	–	
0xFD	Reserved	–	–	–	–	–	–	–	–	
0xFC	Reserved	–	–	–	–	–	–	–	–	
0xFB	Reserved	–	–	–	–	–	–	–	–	
0xFA	Reserved	–	–	–	–	–	–	–	–	
0xF9	Reserved	–	–	–	–	–	–	–	–	
0xF8	Reserved	–	–	–	–	–	–	–	–	
0xF7	Reserved	–	–	–	–	–	–	–	–	
0xF6	Reserved	–	–	–	–	–	–	–	–	
0xF5	Reserved	–	–	–	–	–	–	–	–	
0xF4	Reserved	–	–	–	–	–	–	–	–	
0xF3	Reserved	–	–	–	–	–	–	–	–	
0xF2	Reserved	–	–	–	–	–	–	–	–	
0xF1	Reserved	–	–	–	–	–	–	–	–	
0xF0	Reserved	–	–	–	–	–	–	–	–	
0xEF	Reserved	–	–	–	–	–	–	–	–	
0xEE	Reserved	–	–	–	–	–	–	–	–	
0xED	Reserved	–	–	–	–	–	–	–	–	
0xEC	Reserved	–	–	–	–	–	–	–	–	
0xEB	Reserved	–	–	–	–	–	–	–	–	
0xEA	Reserved	–	–	–	–	–	–	–	–	
0xE9	Reserved	–	–	–	–	–	–	–	–	
0xE8	Reserved	–	–	–	–	–	–	–	–	
0xE7	Reserved	–	–	–	–	–	–	–	–	
0xE6	Reserved	–	–	–	–	–	–	–	–	
0xE5	Reserved	–	–	–	–	–	–	–	–	
0xE4	Reserved	–	–	–	–	–	–	–	–	
0xE3	Reserved	–	–	–	–	–	–	–	–	
0xE2	Reserved	–	–	–	–	–	–	–	–	
0xE1	Reserved	–	–	–	–	–	–	–	–	
0xE0	Reserved	–	–	–	–	–	–	–	–	
0xDF	Reserved	–	–	–	–	–	–	–	–	
0xDE	Reserved	–	–	–	–	–	–	–	–	
0xDD	Reserved	–	–	–	–	–	–	–	–	
0xDC	Reserved	–	–	–	–	–	–	–	–	
0xDB	Reserved	–	–	–	–	–	–	–	–	
0xDA	Reserved	–	–	–	–	–	–	–	–	
0xD9	Reserved	–	–	–	–	–	–	–	–	
0xD8	Reserved	–	–	–	–	–	–	–	–	
0xD7	Reserved	–	–	–	–	–	–	–	–	
0xD6	Reserved	–	–	–	–	–	–	–	–	
0xD5	Reserved	–	–	–	–	–	–	–	–	
0xD4	Reserved	–	–	–	–	–	–	–	–	
0xD3	Reserved	–	–	–	–	–	–	–	–	
0xD2	Reserved	–	–	–	–	–	–	–	–	
0xD1	Reserved	–	–	–	–	–	–	–	–	
0xD0	Reserved	–	–	–	–	–	–	–	–	
0xCF	Reserved	–	–	–	–	–	–	–	–	
0xCE	Reserved	–	–	–	–	–	–	–	–	
0xCD	Reserved	–	–	–	–	–	–	–	–	
0xCC	Reserved	–	–	–	–	–	–	–	–	
0xCB	Reserved	–	–	–	–	–	–	–	–	
0xCA	Reserved	–	–	–	–	–	–	–	–	
0xC9	Reserved	–	–	–	–	–	–	–	–	
0xC8	Reserved	–	–	–	–	–	–	–	–	
0xC7	Reserved	–	–	–	–	–	–	–	–	
0xC6	UDR0	USART I/O Data Register								195
0xC5	UBRR0H					USART Baud Rate Register High				199
0xC4	UBRR0L	USART Baud Rate Register Low								199
0xC3	Reserved	–	–	–	–	–	–	–	–	
0xC2	UCSROC	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01 /UDORD0	UCSZ00 /UCPHA0	UCPOL0	197/212



Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xC1)	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80	196
(0xC0)	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0	195
(0xBF)	Reserved	–	–	–	–	–	–	–	–	
(0xBE)	Reserved	–	–	–	–	–	–	–	–	
(0xBD)	TWAMR	TWAM6	TWAM5	TWAM4	TWAM3	TWAM2	TWAM1	TWAM0	–	244
(0xBC)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	241
(0xBB)	TWDR	2-wire Serial Interface Data Register								243
(0xBA)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	244
(0xB9)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	243
(0xB8)	TWBR	2-wire Serial Interface Bit Rate Register								241
(0xB7)	Reserved	–	–	–	–	–	–	–	–	
(0xB6)	ASSR	–	EXCLK	AS2	TCN2UB	OCR2AUB	OCR2BUB	TCR2AUB	TCR2BUB	164
(0xB5)	Reserved	–	–	–	–	–	–	–	–	
(0xB4)	OCR2B	Timer/Counter2 Output Compare Register B								162
(0xB3)	OCR2A	Timer/Counter2 Output Compare Register A								162
(0xB2)	TCNT2	Timer/Counter2 (8-bit)								162
(0xB1)	TCCR2B	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	161
(0xB0)	TCCR2A	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	158
(0xAF)	Reserved	–	–	–	–	–	–	–	–	
(0xAE)	Reserved	–	–	–	–	–	–	–	–	
(0xAD)	Reserved	–	–	–	–	–	–	–	–	
(0xAC)	Reserved	–	–	–	–	–	–	–	–	
(0xAB)	Reserved	–	–	–	–	–	–	–	–	
(0xAA)	Reserved	–	–	–	–	–	–	–	–	
(0xA9)	Reserved	–	–	–	–	–	–	–	–	
(0xA8)	Reserved	–	–	–	–	–	–	–	–	
(0xA7)	Reserved	–	–	–	–	–	–	–	–	
(0xA6)	Reserved	–	–	–	–	–	–	–	–	
(0xA5)	Reserved	–	–	–	–	–	–	–	–	
(0xA4)	Reserved	–	–	–	–	–	–	–	–	
(0xA3)	Reserved	–	–	–	–	–	–	–	–	
(0xA2)	Reserved	–	–	–	–	–	–	–	–	
(0xA1)	Reserved	–	–	–	–	–	–	–	–	
(0xA0)	Reserved	–	–	–	–	–	–	–	–	
(0x9F)	Reserved	–	–	–	–	–	–	–	–	
(0x9E)	Reserved	–	–	–	–	–	–	–	–	
(0x9D)	Reserved	–	–	–	–	–	–	–	–	
(0x9C)	Reserved	–	–	–	–	–	–	–	–	
(0x9B)	Reserved	–	–	–	–	–	–	–	–	
(0x9A)	Reserved	–	–	–	–	–	–	–	–	
(0x99)	Reserved	–	–	–	–	–	–	–	–	
(0x98)	Reserved	–	–	–	–	–	–	–	–	
(0x97)	Reserved	–	–	–	–	–	–	–	–	
(0x96)	Reserved	–	–	–	–	–	–	–	–	
(0x95)	Reserved	–	–	–	–	–	–	–	–	
(0x94)	Reserved	–	–	–	–	–	–	–	–	
(0x93)	Reserved	–	–	–	–	–	–	–	–	
(0x92)	Reserved	–	–	–	–	–	–	–	–	
(0x91)	Reserved	–	–	–	–	–	–	–	–	
(0x90)	Reserved	–	–	–	–	–	–	–	–	
(0x8F)	Reserved	–	–	–	–	–	–	–	–	
(0x8E)	Reserved	–	–	–	–	–	–	–	–	
(0x8D)	Reserved	–	–	–	–	–	–	–	–	
(0x8C)	Reserved	–	–	–	–	–	–	–	–	
(0x8B)	OCR1BH	Timer/Counter1 - Output Compare Register B High Byte								138
(0x8A)	OCR1BL	Timer/Counter1 - Output Compare Register B Low Byte								138
(0x89)	OCR1AH	Timer/Counter1 - Output Compare Register A High Byte								138
(0x88)	OCR1AL	Timer/Counter1 - Output Compare Register A Low Byte								138
(0x87)	ICR1H	Timer/Counter1 - Input Capture Register High Byte								139
(0x86)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte								139
(0x85)	TCNT1H	Timer/Counter1 - Counter Register High Byte								138
(0x84)	TCNT1L	Timer/Counter1 - Counter Register Low Byte								138
(0x83)	Reserved	–	–	–	–	–	–	–	–	
(0x82)	TCCR1C	FOC1A	FOC1B	–	–	–	–	–	–	137
(0x81)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	136
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	134

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0x7F)	DIDR1	–	–	–	–	–	–	AIN1D	AIN0D	249
(0x7E)	DIDR0	–	–	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	266
(0x7D)	Reserved	–	–	–	–	–	–	–	–	
(0x7C)	ADMUX	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	262
(0x7B)	ADCSRB	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	265
(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	263
(0x79)	ADCH	ADC Data Register High byte								265
(0x78)	ADCL	ADC Data Register Low byte								265
(0x77)	Reserved	–	–	–	–	–	–	–	–	
(0x76)	Reserved	–	–	–	–	–	–	–	–	
(0x75)	Reserved	–	–	–	–	–	–	–	–	
(0x74)	Reserved	–	–	–	–	–	–	–	–	
(0x73)	Reserved	–	–	–	–	–	–	–	–	
(0x72)	Reserved	–	–	–	–	–	–	–	–	
(0x71)	Reserved	–	–	–	–	–	–	–	–	
(0x70)	TIMSK2	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2	163
(0x6F)	TIMSK1	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	139
(0x6E)	TIMSK0	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	111
(0x6D)	PCMSK2	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	74
(0x6C)	PCMSK1	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	74
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	74
(0x6A)	Reserved	–	–	–	–	–	–	–	–	
(0x69)	EICRA	–	–	–	–	ISC11	ISC10	ISC01	ISC00	71
(0x68)	PCICR	–	–	–	–	–	PCIE2	PCIE1	PCIE0	
(0x67)	Reserved	–	–	–	–	–	–	–	–	
(0x66)	OSCCAL	Oscillator Calibration Register								37
(0x65)	Reserved	–	–	–	–	–	–	–	–	
(0x64)	PRR	PRTW1	PRTIM2	PRTIM0	–	PRTIM1	PRSPI	PRUSART0	PRADC	42
(0x63)	Reserved	–	–	–	–	–	–	–	–	
(0x62)	Reserved	–	–	–	–	–	–	–	–	
(0x61)	CLKPR	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	37
(0x60)	WDTCR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	54
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	9
0x3E (0x5E)	SPH	–	–	–	–	–	(SP10) <sup>5</sup>	SP9	SP8	12
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	12
0x3C (0x5C)	Reserved	–	–	–	–	–	–	–	–	
0x3B (0x5B)	Reserved	–	–	–	–	–	–	–	–	
0x3A (0x5A)	Reserved	–	–	–	–	–	–	–	–	
0x39 (0x59)	Reserved	–	–	–	–	–	–	–	–	
0x38 (0x58)	Reserved	–	–	–	–	–	–	–	–	
0x37 (0x57)	SPMCSR	SPMIE	(RWWSB) <sup>5</sup>	–	(RWWRE) <sup>5</sup>	BLBSET	PGWRT	PGERS	SELFPRGEN	292
0x36 (0x56)	Reserved	–	–	–	–	–	–	–	–	
0x35 (0x55)	MCUCR	–	BODS	BODSE	PUD	–	–	IVSEL	IVCE	44/68/92
0x34 (0x54)	MCUSR	–	–	–	–	WDRF	BORF	EXTRF	PORF	54
0x33 (0x53)	SMCR	–	–	–	–	SM2	SM1	SM0	SE	40
0x32 (0x52)	Reserved	–	–	–	–	–	–	–	–	
0x31 (0x51)	Reserved	–	–	–	–	–	–	–	–	
0x30 (0x50)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	247
0x2F (0x4F)	Reserved	–	–	–	–	–	–	–	–	
0x2E (0x4E)	SPDR	SPI Data Register								175
0x2D (0x4D)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	174
0x2C (0x4C)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	173
0x2B (0x4B)	GPIOR2	General Purpose I/O Register 2								25
0x2A (0x4A)	GPIOR1	General Purpose I/O Register 1								25
0x29 (0x49)	Reserved	–	–	–	–	–	–	–	–	
0x28 (0x48)	OCR0B	Timer/Counter0 Output Compare Register B								
0x27 (0x47)	OCR0A	Timer/Counter0 Output Compare Register A								
0x26 (0x46)	TCNT0	Timer/Counter0 (8-bit)								
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	
0x23 (0x43)	GTCCR	TSM	–	–	–	–	–	PSRASY	PSRSYNC	143/165
0x22 (0x42)	EEARH	(EEPROM Address Register High Byte) <sup>5</sup>								21
0x21 (0x41)	EEARL	EEPROM Address Register Low Byte								21
0x20 (0x40)	EEDR	EEPROM Data Register								21
0x1F (0x3F)	EEDR	–	–	EEP1	EEP0	EERIE	EEMPE	EEPE	EERE	21
0x1E (0x3E)	GPIOR0	General Purpose I/O Register 0								25

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x1D (0x3D)	EIMSK	–	–	–	–	–	–	INT1	INT0	72
0x1C (0x3C)	EIFR	–	–	–	–	–	–	INTF1	INTF0	72
0x1B (0x3B)	PCIFR	–	–	–	–	–	PCIF2	PCIF1	PCIF0	
0x1A (0x3A)	Reserved	–	–	–	–	–	–	–	–	
0x19 (0x39)	Reserved	–	–	–	–	–	–	–	–	
0x18 (0x38)	Reserved	–	–	–	–	–	–	–	–	
0x17 (0x37)	TIFR2	–	–	–	–	–	OCF2B	OCF2A	TOV2	163
0x16 (0x36)	TIFR1	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	140
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0	
0x14 (0x34)	Reserved	–	–	–	–	–	–	–	–	
0x13 (0x33)	Reserved	–	–	–	–	–	–	–	–	
0x12 (0x32)	Reserved	–	–	–	–	–	–	–	–	
0x11 (0x31)	Reserved	–	–	–	–	–	–	–	–	
0x10 (0x30)	Reserved	–	–	–	–	–	–	–	–	
0x0F (0x2F)	Reserved	–	–	–	–	–	–	–	–	
0x0E (0x2E)	Reserved	–	–	–	–	–	–	–	–	
0x0D (0x2D)	Reserved	–	–	–	–	–	–	–	–	
0x0C (0x2C)	Reserved	–	–	–	–	–	–	–	–	
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	93
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	93
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	93
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	92
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	92
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	92
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	92
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	92
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	92
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–	
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–	
0x0 (0x20)	Reserved	–	–	–	–	–	–	–	–	

- Note:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
  2. I/O Registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
  3. Some of the Status Flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such Status Flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
  4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATmega48P/88P/168P/328P is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
  5. Only valid for ATmega88P/168P.

## 5. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
<b>ARITHMETIC AND LOGIC INSTRUCTIONS</b>					
ADD	Rd, Rr	Add two Registers	$Rd \leftarrow Rd + Rr$	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	$Rd \leftarrow Rd + Rr + C$	Z,C,N,V,H	1
ADIW	Rd,K	Add Immediate to Word	$Rdh:Rdl \leftarrow Rdh:Rdl + K$	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	$Rd \leftarrow Rd - Rr$	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	$Rd \leftarrow Rd - K$	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	$Rd \leftarrow Rd - Rr - C$	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	$Rd \leftarrow Rd - K - C$	Z,C,N,V,H	1
SBIW	Rd,K	Subtract Immediate from Word	$Rdh:Rdl \leftarrow Rdh:Rdl - K$	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	$Rd \leftarrow Rd \bullet Rr$	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	$Rd \leftarrow Rd \bullet K$	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	$Rd \leftarrow Rd \vee Rr$	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	$Rd \leftarrow Rd \vee K$	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	$Rd \leftarrow Rd \oplus Rr$	Z,N,V	1
COM	Rd	One's Complement	$Rd \leftarrow 0xFF - Rd$	Z,C,N,V	1
NEG	Rd	Two's Complement	$Rd \leftarrow 0x00 - Rd$	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	$Rd \leftarrow Rd \vee K$	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	$Rd \leftarrow Rd \bullet (0xFF - K)$	Z,N,V	1
INC	Rd	Increment	$Rd \leftarrow Rd + 1$	Z,N,V	1
DEC	Rd	Decrement	$Rd \leftarrow Rd - 1$	Z,N,V	1
TST	Rd	Test for Zero or Minus	$Rd \leftarrow Rd \bullet Rd$	Z,N,V	1
CLR	Rd	Clear Register	$Rd \leftarrow Rd \oplus Rd$	Z,N,V	1
SER	Rd	Set Register	$Rd \leftarrow 0xFF$	None	1
MUL	Rd, Rr	Multiply Unsigned	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
MULS	Rd, Rr	Multiply Signed	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	$R1:R0 \leftarrow (Rd \times Rr) \lll 1$	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	$R1:R0 \leftarrow (Rd \times Rr) \lll 1$	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	$R1:R0 \leftarrow (Rd \times Rr) \lll 1$	Z,C	2
<b>BRANCH INSTRUCTIONS</b>					
RJMP	k	Relative Jump	$PC \leftarrow PC + k + 1$	None	2
IJMP		Indirect Jump to (Z)	$PC \leftarrow Z$	None	2
JMP <sup>(1)</sup>	k	Direct Jump	$PC \leftarrow k$	None	3
RCALL	k	Relative Subroutine Call	$PC \leftarrow PC + k + 1$	None	3
ICALL		Indirect Call to (Z)	$PC \leftarrow Z$	None	3
CALL <sup>(1)</sup>	k	Direct Subroutine Call	$PC \leftarrow k$	None	4
RET		Subroutine Return	$PC \leftarrow STACK$	None	4
RETI		Interrupt Return	$PC \leftarrow STACK$	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if $(Rd = Rr)$ $PC \leftarrow PC + 2$ or 3	None	1/2/3
CP	Rd,Rr	Compare	$Rd - Rr$	Z, N, V, C, H	1
CPC	Rd,Rr	Compare with Carry	$Rd - Rr - C$	Z, N, V, C, H	1
CPI	Rd,K	Compare Register with Immediate	$Rd - K$	Z, N, V, C, H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if $(Rr(b)=0)$ $PC \leftarrow PC + 2$ or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set	if $(Rr(b)=1)$ $PC \leftarrow PC + 2$ or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if $(P(b)=0)$ $PC \leftarrow PC + 2$ or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if $(P(b)=1)$ $PC \leftarrow PC + 2$ or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if $(SREG(s) = 1)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if $(SREG(s) = 0)$ then $PC \leftarrow PC + k + 1$	None	1/2
BREQ	k	Branch if Equal	if $(Z = 1)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRNE	k	Branch if Not Equal	if $(Z = 0)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRCS	k	Branch if Carry Set	if $(C = 1)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRCC	k	Branch if Carry Cleared	if $(C = 0)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRSH	k	Branch if Same or Higher	if $(C = 0)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRLO	k	Branch if Lower	if $(C = 1)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRMI	k	Branch if Minus	if $(N = 1)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRPL	k	Branch if Plus	if $(N = 0)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if $(N \oplus V = 0)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if $(N \oplus V = 1)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if $(H = 1)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if $(H = 0)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRTS	k	Branch if T Flag Set	if $(T = 1)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRTC	k	Branch if T Flag Cleared	if $(T = 0)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if $(V = 1)$ then $PC \leftarrow PC + k + 1$	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if $(V = 0)$ then $PC \leftarrow PC + k + 1$	None	1/2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2
<b>BIT AND BIT-TEST INSTRUCTIONS</b>					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0) ← C, Rd(n+1) ← Rd(n), C ← Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7) ← C, Rd(n) ← Rd(n+1), C ← Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0) ← Rd(7..4), Rd(7..4) ← Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
<b>DATA TRANSFER INSTRUCTIONS</b>					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, -X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, -Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd, Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	-X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	-Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q, Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q, Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
<b>MCU CONTROL INSTRUCTIONS</b>					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A

Note: 1. These instructions are only available in ATmega168P and ATmega328P.

## 6. Ordering Information

### 6.1 ATmega48P

Speed (MHz)	Power Supply	Ordering Code <sup>(2)</sup>	Package <sup>(1)</sup>	Operational Range
10 <sup>(3)</sup>	1.8 - 5.5	ATmega48PV-10AU ATmega48PV-10MMU ATmega48PV-10MU ATmega48PV-10PU	32A 28M1 32M1-A 28P3	Industrial (-40°C to 85°C)
20 <sup>(3)</sup>	2.7 - 5.5	ATmega48P-20AU ATmega48P-20MMU ATmega48P-20MU ATmega48P-20PU	32A 28M1 32M1-A 28P3	Industrial (-40°C to 85°C)

- Note:
1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
  2. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
  3. See [Figure 26-1 on page 317](#) and [Figure 26-2 on page 317](#).

Package Type	
<b>32A</b>	32-lead, Thin (1.0 mm) Plastic Quad Flat Package (TQFP)
<b>28M1</b>	28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45 mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
<b>32M1-A</b>	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50 mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
<b>28P3</b>	28-lead, 0.300" Wide, Plastic Dual Inline Package (PDIP)



## 6.2 ATmega88P

Speed (MHz)	Power Supply	Ordering Code <sup>(2)</sup>	Package <sup>(1)</sup>	Operational Range
10 <sup>(3)</sup>	1.8 - 5.5	ATmega88PV-10AU ATmega88PV-10MU ATmega88PV-10PU	32A 32M1-A 28P3	Industrial (-40°C to 85°C)
20 <sup>(3)</sup>	2.7 - 5.5	ATmega88P-20AU ATmega88P-20MU ATmega88P-20PU	32A 32M1-A 28P3	Industrial (-40°C to 85°C)

- Note:
1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
  2. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
  3. See [Figure 26-1 on page 317](#) and [Figure 26-2 on page 317](#).

Package Type	
<b>32A</b>	32-lead, Thin (1.0 mm) Plastic Quad Flat Package (TQFP)
<b>28P3</b>	28-lead, 0.300" Wide, Plastic Dual Inline Package (PDIP)
<b>32M1-A</b>	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50 mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)

## 6.3 ATmega168P

Speed (MHz) <sup>(3)</sup>	Power Supply	Ordering Code <sup>(2)</sup>	Package <sup>(1)</sup>	Operational Range
10	1.8 - 5.5	ATmega168PV-10AU ATmega168PV-10MU ATmega168PV-10PU	32A 32M1-A 28P3	Industrial (-40°C to 85°C)
20	2.7 - 5.5	ATmega168P-20AU ATmega168P-20MU ATmega168P-20PU	32A 32M1-A 28P3	Industrial (-40°C to 85°C)

- Note:
1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
  2. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
  3. See [Figure 26-1 on page 317](#) and [Figure 26-2 on page 317](#).

Package Type	
<b>32A</b>	32-lead, Thin (1.0 mm) Plastic Quad Flat Package (TQFP)
<b>28P3</b>	28-lead, 0.300" Wide, Plastic Dual Inline Package (PDIP)
<b>32M1-A</b>	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50 mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)

## 6.4 ATmega328P

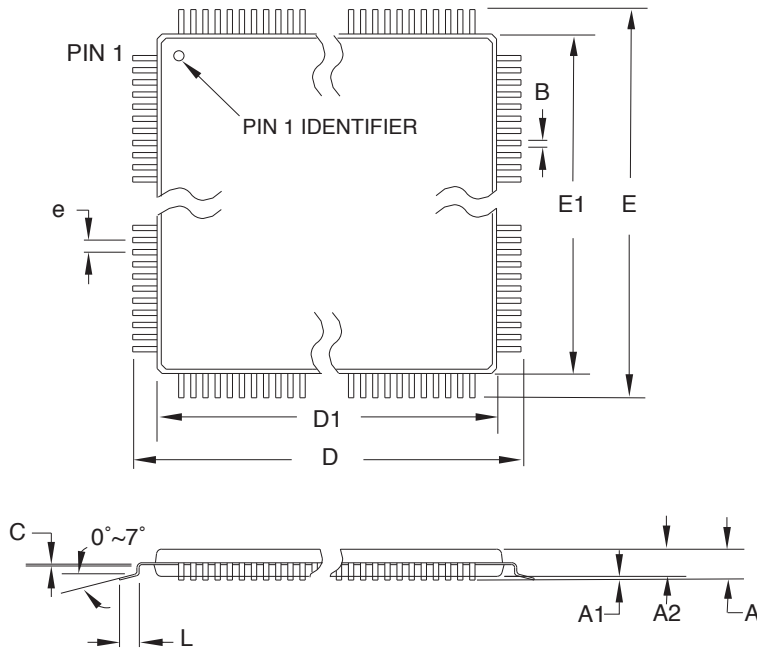
Speed (MHz)	Power Supply	Ordering Code <sup>(2)</sup>	Package <sup>(1)</sup>	Operational Range
20 <sup>(3)</sup>	1.8 - 5.5	ATmega328P- AU ATmega328P- MU ATmega328P- PU	32A 32M1-A 28P3	Industrial (-40°C to 85°C)

- Note:
1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
  2. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
  3. See [Figure 26-3 on page 318](#).

Package Type	
<b>32A</b>	32-lead, Thin (1.0 mm) Plastic Quad Flat Package (TQFP)
<b>28P3</b>	28-lead, 0.300" Wide, Plastic Dual Inline Package (PDIP)
<b>32M1-A</b>	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50 mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)

## 7. Packaging Information

### 7.1 32A



**COMMON DIMENSIONS**  
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	1.20	
A1	0.05	–	0.15	
A2	0.95	1.00	1.05	
D	8.75	9.00	9.25	
D1	6.90	7.00	7.10	Note 2
E	8.75	9.00	9.25	
E1	6.90	7.00	7.10	Note 2
B	0.30	–	0.45	
C	0.09	–	0.20	
L	0.45	–	0.75	
e	0.80 TYP			

- Notes:
1. This package conforms to JEDEC reference MS-026, Variation ABA.
  2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25 mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
  3. Lead coplanarity is 0.10 mm maximum.

10/5/2001



2325 Orchard Parkway  
San Jose, CA 95131

**TITLE**

**32A**, 32-lead, 7 x 7 mm Body Size, 1.0 mm Body Thickness,  
0.8 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP)

**DRAWING NO.**

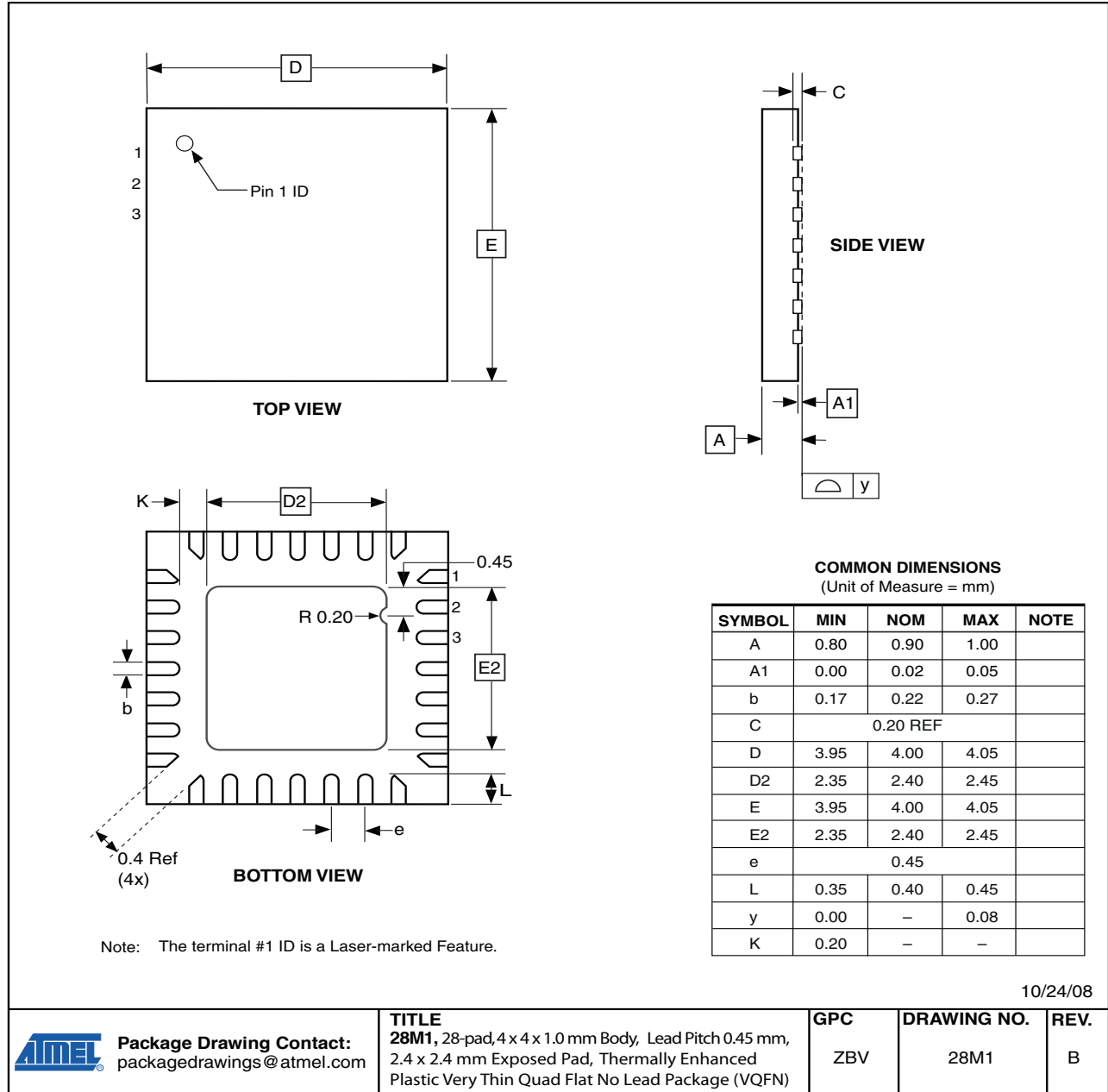
32A

**REV.**

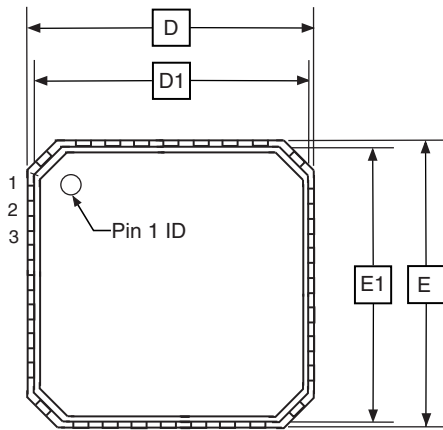
B



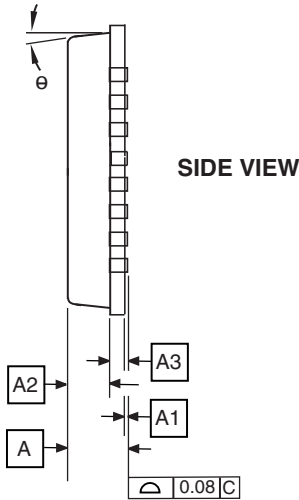
## 7.2 28M1



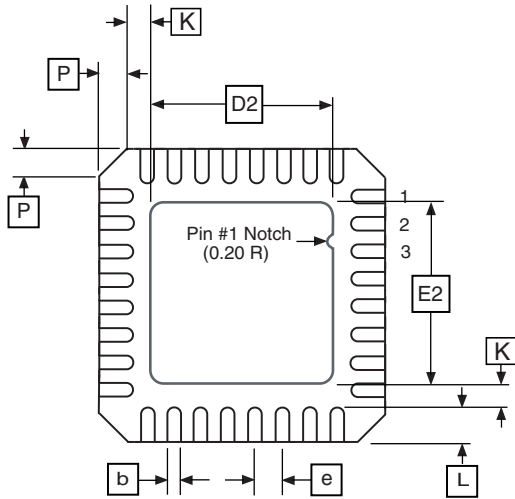
## 7.3 32M1-A



**TOP VIEW**



**SIDE VIEW**



**BOTTOM VIEW**

**COMMON DIMENSIONS**  
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	0.80	0.90	1.00	
A1	–	0.02	0.05	
A2	–	0.65	1.00	
A3	0.20 REF			
b	0.18	0.23	0.30	
D	4.90	5.00	5.10	
D1	4.70	4.75	4.80	
D2	2.95	3.10	3.25	
E	4.90	5.00	5.10	
E1	4.70	4.75	4.80	
E2	2.95	3.10	3.25	
e	0.50 BSC			
L	0.30	0.40	0.50	
P	–	–	0.60	
θ	–	–	12°	
K	0.20	–	–	

Note: JEDEC Standard MO-220, Fig. 2 (Anvil Singulation), VHHD-2.

5/25/06



2325 Orchard Parkway  
San Jose, CA 95131

**TITLE**

**32M1-A**, 32-pad, 5 x 5 x 1.0 mm Body, Lead Pitch 0.50 mm,  
3.10 mm Exposed Pad, Micro Lead Frame Package (MLF)

**DRAWING NO.**

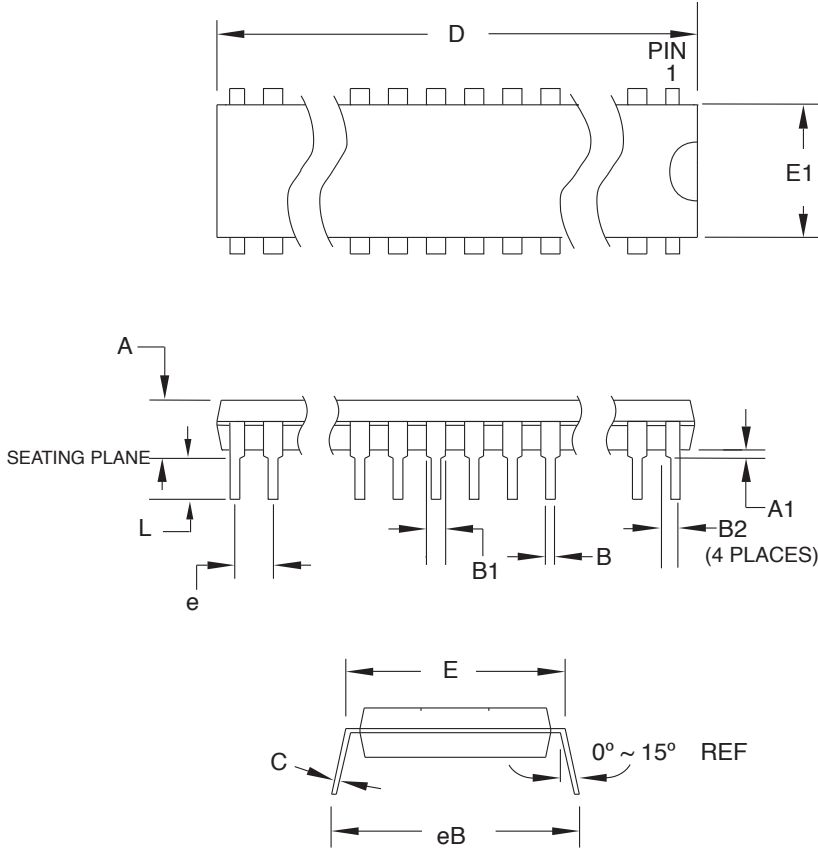
32M1-A

**REV.**

E



## 7.4 28P3



**COMMON DIMENSIONS**  
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	-	-	4.5724	
A1	0.508	-	-	
D	34.544	-	34.798	Note 1
E	7.620	-	8.255	
E1	7.112	-	7.493	Note 1
B	0.381	-	0.533	
B1	1.143	-	1.397	
B2	0.762	-	1.143	
L	3.175	-	3.429	
C	0.203	-	0.356	
eB	-	-	10.160	
e	2.540 TYP			

Note: 1. Dimensions D and E1 do not include mold Flash or Protrusion.  
Mold Flash or Protrusion shall not exceed 0.25 mm (0.010").

09/28/01



2325 Orchard Parkway  
San Jose, CA 95131

**TITLE**

**28P3**, 28-lead (0.300"/7.62 mm Wide) Plastic Dual  
Inline Package (PDIP)

**DRAWING NO.**

28P3

**REV.**

B



## 8. Errata

### 8.1 Errata ATmega48P

The revision letter in this section refers to the revision of the ATmega48P device.

#### 8.1.1 Rev. C

No known errata.

#### 8.1.2 Rev. B

No known errata.

#### 8.1.3 Rev. A

Not Sampled.

### 8.2 Errata ATmega88P

The revision letter in this section refers to the revision of the ATmega88P device.

#### 8.2.1 Rev. C

Not sampled.

#### 8.2.2 Rev. B

No known errata.

#### 8.2.3 Rev. A

No known errata.

### 8.3 Errata ATmega168P

The revision letter in this section refers to the revision of the ATmega168P device.

#### 8.3.1 Rev B

No known errata.

#### 8.3.2 Rev A

No known errata.

### 8.4 Errata ATmega328P

The revision letter in this section refers to the revision of the ATmega328P device.

#### 8.4.1 Rev C

No known errata.

#### 8.4.2 Rev B

- **Unstable 32 kHz Oscillator**

1. **Unstable 32 kHz Oscillator**

The 32 kHz oscillator does not work as system clock.

The 32 kHz oscillator used as asynchronous timer is inaccurate.



## Problem Fix/ Workaround

None

### 8.4.3 Rev A

- **Unstable 32 kHz Oscillator**

1. **Unstable 32 kHz Oscillator**

The 32 kHz oscillator does not work as system clock.

The 32 kHz oscillator used as asynchronous timer is inaccurate.

## Problem Fix/ Workaround

None

## 9. Datasheet Revision History

Please note that the referring page numbers in this section are referred to this document. The referring revision in this section are referring to the document revision.

### 9.1 Rev. 2545I-02/09

1. Removed “preliminary” from ATmega48P/88P/168P.

### 9.2 Rev. 2545H-02/09

1. Added Power-save Maximum values and footnote to ["ATmega48P DC Characteristics" on page 314](#).
2. Added Power-save Maximum values and footnote to ["ATmega88P DC Characteristics" on page 315](#).
3. Added Power-save Maximum values and footnote to ["ATmega168P DC Characteristics" on page 315](#).
4. Added Power-save Maximum values and footnote to ["ATmega328P DC Characteristics" on page 316](#).
5. Added errata for revision A, ["Errata ATmega328P" on page 440](#).

### 9.3 Rev. 2545G-01/09

1. ATmega48P/88P not recommended for new designs.
2. Updated the footnote Note1 of the [Table 6-3 on page 29](#).
3. Updated the [Table 6-5 on page 30](#) by removing a footnote Note1.
4. Updated the [Table 6-10 on page 33](#) by removing a footnote Note1.
5. Updated the footnote Note1 of the [Table 6-12 on page 34](#).
6. Updated the footnote Note2 of the ["ATmega48P DC Characteristics" on page 314](#) and removed TBD from the table.
7. Updated the footnote Note2 of the ["ATmega88P DC Characteristics" on page 315](#) and removed TBD from the table.
8. Updated the footnote Note2 of the ["ATmega168P DC Characteristics" on page 315](#) and removed TBD from the table.
9. Updated the footnote Note2 of the ["ATmega328P DC Characteristics" on page 316](#) and removed TBD from the table.
10. Updated the footnote Note1 of the [Table 26-4 on page 320](#).
11. Replaced the [Figure 27-69 on page 365](#) by a correct one.
12. Replaced the [Figure 27-173 on page 419](#) by a correct one.
13. Updated ["Errata" on page 440](#).
14. Updated ["MCUCR – MCU Control Register" on page 44](#).
15. Updated ["TCCR2B – Timer/Counter Control Register B" on page 161](#).

## 9.4 Rev. 2545F-08/08

1. Updated "ATmega328P Typical Characteristics" on page 401 with Power-save numbers.
2. Added ATmega328P "Standby Supply Current" on page 408.

## 9.5 Rev. 2545E-08/08

1. Updated description of "Stack Pointer" on page 12.
2. Updated description of use of external capacitors in "Low Frequency Crystal Oscillator" on page 32.
3. Updated Table 6-9 in "Low Frequency Crystal Oscillator" on page 32.
4. Added note to "Address Match Unit" on page 222.
5. Added section "Reading the Signature Row from Software" on page 285.
6. Updated "Program And Data Memory Lock Bits" on page 294 to include ATmega328P in the description.
7. Added "ATmega328P DC Characteristics" on page 316.
8. Updated "Speed Grades" on page 316 for ATmega328P.
9. Removed note 6 and 7 from the table "2-wire Serial Interface Characteristics" on page 323.
10. Added figure "Minimum Reset Pulse width vs.  $V_{CC}$ ." on page 352 for ATmega48P.
11. Added figure "Minimum Reset Pulse width vs.  $V_{CC}$ ." on page 376 for ATmega88P.
12. Added figure "Minimum Reset Pulse width vs.  $V_{CC}$ ." on page 400 for ATmega168P.
13. Added "ATmega328P Typical Characteristics" on page 401.
14. Updated Ordering Information for "ATmega328P" on page 435.

## 9.6 Rev. 2545D-03/08

1. Updated figures in "Speed Grades" on page 316.
2. Updated note in Table 26-4 in "System and Reset Characteristics" on page 320.
3. Ordering codes for "ATmega328P" on page 435 updated.
  - ATmega328P is offered in 20 MHz option only.
4. Added Errata for ATmega328P rev. B, "Errata ATmega328P" on page 440.

## 9.7 Rev. 2545C-01/08

1. Power-save Maximum values removed from "ATmega48P DC Characteristics" on page 314, "ATmega88P DC Characteristics" on page 315, and "ATmega168P DC Characteristics" on page 315.

**9.8 Rev. 2545B-01/08**

1. Updated "Features" on page 1.
2. Added "Data Retention" on page 7.
3. Updated Table 6-2 on page 28.
4. Removed "Low-frequency Crystal Oscillator Internal Load Capacitance" table from "Low Frequency Crystal Oscillator" on page 32.
5. Removed JTD bit from "MCUCR – MCU Control Register" on page 44.
6. Updated typical and general program setup for Reset and Interrupt Vector Addresses in "Interrupt Vectors in ATmega168P" on page 62 and "Interrupt Vectors in ATmega328P" on page 65.
7. Updated Interrupt Vectors Start Address in Table 9-5 on page 63 and Table 9-7 on page 66.
8. Updated "Temperature Measurement" on page 261.
9. Updated ATmega328P "Fuse Bits" on page 295.
10. Removed  $V_{OL3}/V_{OH3}$  rows from "DC Characteristics" on page 313.
11. Updated condition for  $V_{OL}$  in "DC Characteristics" on page 313.  
Updated max value for  $V_{IL2}$  in "DC Characteristics" on page 313.
12. Added "ATmega48P DC Characteristics" on page 314, "ATmega88P DC Characteristics" on page 315, and "ATmega168P DC Characteristics" on page 315.
13. Updated "System and Reset Characteristics" on page 320.
14. Added "ATmega48P Typical Characteristics" on page 329, "ATmega88P Typical Characteristics" on page 353, and "ATmega168P Typical Characteristics" on page 377.
15. Updated note in "Instruction Set Summary" on page 429.

**9.9 Rev. 2545A-07/07**

1. Initial revision.



## Headquarters

---

**Atmel Corporation**  
2325 Orchard Parkway  
San Jose, CA 95131  
USA  
Tel: 1(408) 441-0311  
Fax: 1(408) 487-2600

## International

---

**Atmel Asia**  
Unit 1-5 & 16, 19/F  
BEA Tower, Millennium City 5  
418 Kwun Tong Road  
Kwun Tong, Kowloon  
Hong Kong  
Tel: (852) 2245-6100  
Fax: (852) 2722-1369

**Atmel Europe**  
Le Krebs  
8, Rue Jean-Pierre Timbaud  
BP 309  
78054 Saint-Quentin-en-  
Yvelines Cedex  
France  
Tel: (33) 1-30-60-70-00  
Fax: (33) 1-30-60-71-11

**Atmel Japan**  
9F, Tonetsu Shinkawa Bldg.  
1-24-8 Shinkawa  
Chuo-ku, Tokyo 104-0033  
Japan  
Tel: (81) 3-3523-3551  
Fax: (81) 3-3523-7581

## Product Contact

---

**Web Site**  
[www.atmel.com](http://www.atmel.com)

**Technical Support**  
[avr@atmel.com](mailto:avr@atmel.com)

**Sales Contact**  
[www.atmel.com/contacts](http://www.atmel.com/contacts)

**Literature Requests**  
[www.atmel.com/literature](http://www.atmel.com/literature)

---

**Disclaimer:** The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© 2009 Atmel Corporation. All rights reserved. Atmel®, logo and combinations thereof, AVR® and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.