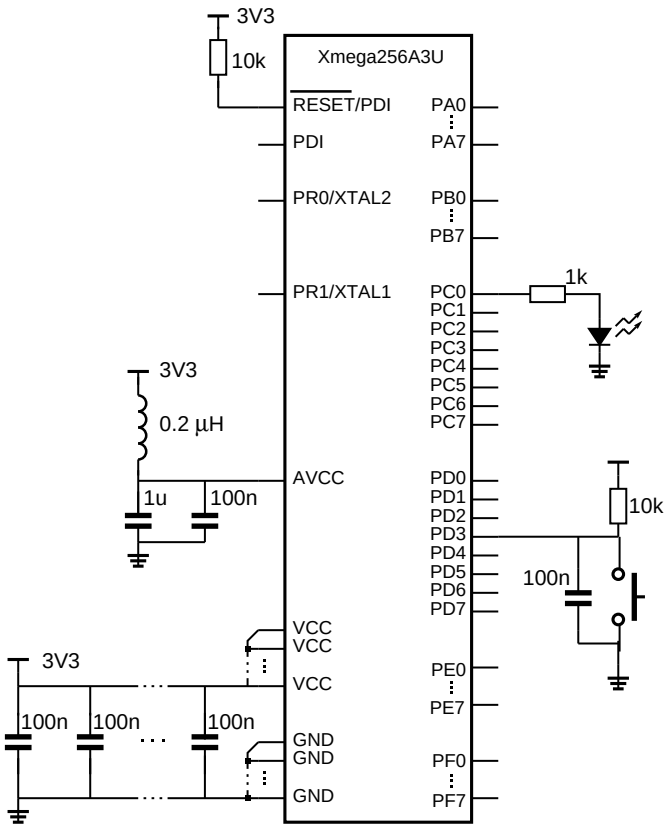
## 15.8 De generieke IO als ingang gebruiken

De generieke IO kan ook gebruikt worden als ingang om signalen te lezen. Een basaal voorbeeld is een schakeling met een drukknop en een led. Als de drukknop ingedrukt wordt is de led aan en anders is de led uit.




**Figuur 15.12 : De schakeling voor het aan- en uitzetten van een led met een drukknop.**

De waarde van de condensator hangt af van de mate van contactdender.
Iedere type drukknop heeft een ander gedrag. Er bestaan ook drukknoppen, die nauwelijks last van contactdender hebben.

De schakeling van figuur 15.12 is de schakeling uit figuur 15.1 met daaraan toegevoegd een drukknop aan pin 3 van poort D. De weerstand en de condensator zijn optioneel. De condensator is toegevoegd om contactdender te voorkomen. Dit kan hard- of softwarematig ook op andere manieren worden voorkomen. In paragraaf 15.10 wordt contactdender besproken. De weerstand zorgt er voor dat als de knop niet ingedrukt is, de ingang van de microcontroller hoog is. Als de knop ingedrukt wordt, wordt de ingang laag.

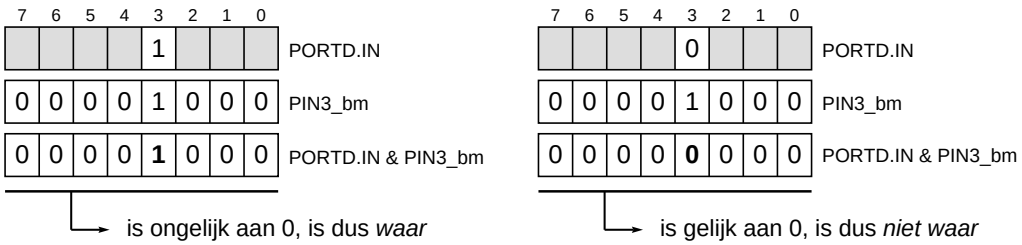
De datastructuur voor de generieke IO, de `struct PORT_t`, bevat het ingangsregister `IN`. Iedere klokslag wordt de inhoud van de ingangsregisters ververst met de waarden die er op dat moment op de aansluitpinnen staan.

In code 15.8 wordt op regel 4 aansluiting 3 van poort D expliciet gedefinieerd als ingang door bit 3 van het `DIR`-register van poort D laag te maken. Standaard zijn bij een reset alle bits van de registers laag. Zonder de toewijzing op regel 4 is deze aansluiting ook een ingang.

**Code 15.8 : Een led die de status van een drukknop weergeeft.**

```
1  #include <avr/io.h>
2
3  int main(void) {
4    PORTD.DIRCLR = PIN3_bm;  // input pin for button
5    PORTC.DIRSET = PIN0_bm;  // output pin for led
6    PORTC.OUTSET = PIN0_bm;  // led on
7
8    while (1) {
9      if ( PORTD.IN & PIN3_bm ) {
10       PORTC.OUTCLR = PIN0_bm;   // button not pressed, led off
11     } else {
12       PORTC.OUTSET = PIN0_bm;   // button pressed, led on
13     }
14   }
15 }
```

In de oneindige **while**-lus wordt op regel 9 getest of de ingang hoog is. Voor deze test wordt de bitsgewijze EN-functie en het bitmasker voor pin 3 gebruikt.




**Figuur 15.13 : De test om te bepalen of bit 3 uit poort D hoog of laag is.** Alle bits, die in het masker `0` zijn, worden bij de bewerking `PORTD.IN & PIN3_bm` eveneens `0`. Afhankelijk van de waarde van bit 3 in `PORTD.IN` is de uitkomst gelijk of ongelijk aan `0`.

Figuur 15.13 laat zien dat de uitdrukking `PORTD.IN & PIN3_bm` waar is als de ingang hoog is en niet waar is als de ingang laag is. In het eerste geval is de drukknop *niet* ingedrukt. De uitgang is dan laag en de led is uit. De ingang is laag, zolang de drukknop ingedrukt wordt. De test is dan niet waar, zodat de toewijzing op regel 12 de uitgang hoog maakt en de led laat branden.

## 15.9 Het aan- en uitzetten van een led met een drukknop

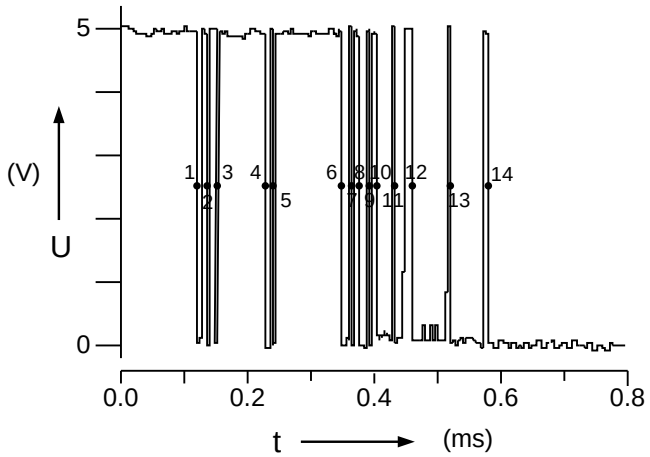
In code 15.8 gaat de led aan als de knop ingedrukt wordt en weer uit als de knop losgelaten wordt. Code 15.9 geeft een eerste aanzet van een programma dat de led laat aangaan als de knop wordt ingedrukt en weer laat uitgaat als er nogmaals op de knop wordt gedrukt.

Als de knop ingedrukt wordt, is de ingang laag. Op regel 12 staat bij de testconditie een `!`, waardoor de conditie waar is als de knop ingedrukt wordt. Op regel 13 wordt het register `OUTTGL` gebruikt om de uitgangswaarde te laten omklappen of *toggelen*. De tijdvertraging op regel 14 zorgt ervoor dat de led na één keer indrukken niet blijft toggelen. Alleen als de knop langer dan 200 ms wordt vastgehouden blijft de led aan- en uitgaan.

**Code 15.9 : Een eerste aanzet om een led aan en uit te zetten met een drukknop.**

```
1  #define  F_CPU 2000000UL
2
3  #include <avr/io.h>
4  #include <util/delay.h>
5
6  int main(void) {
7    PORTD.DIRCLR = PIN3_bm;  // input pin for button
8    PORTC.DIRSET = PIN0_bm;  // output pin for led
9    PORTC.OUTCLR = PIN0_bm;  // led off
10
11   while (1) {
12     if ( ! (PORTD.IN & PIN3_bm) ) {
13       PORTC.OUTTGL = PIN0_bm;   // toggles led
14       _delay_ms(200);
15     }
16   }
17 }
```

De oplossing uit code 15.9 is niet robuust. Als de schakelaar contactdender vertoont en er geen goede hardwarematige ontdenderschakeling wordt gebruikt, kan het gedrag van de schakelaar onvoorspelbaar worden.




**Figuur 15.14 : De dender bij het indrukken van de drukknop.** Het signaal gaat veertien keer van hoog naar laag voordat het definitief laag is geworden.

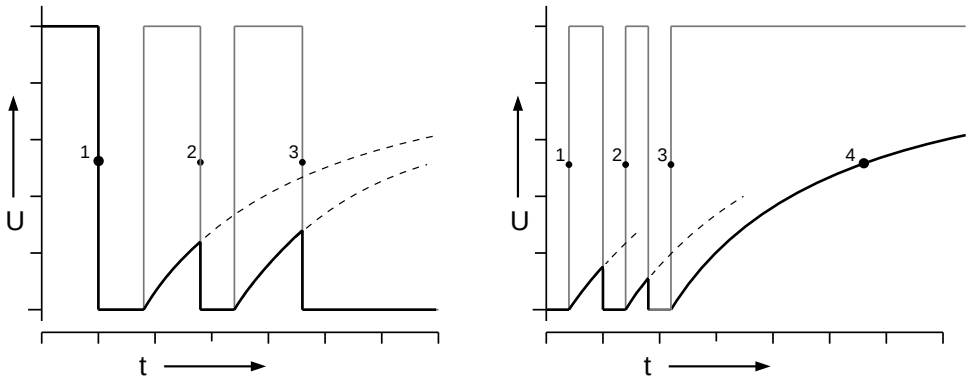
## 15.10 Contactdender

De meeste schakelaars en drukknoppen produceren contactdender. Het contact wordt niet in één keer gemaakt of in één keer verbroken. Figuur 15.14 toont de dender (*bouncing*) bij een maakcontact. In dit voorbeeld gaat het signaal niet één keer maar veertien keer van hoog naar laag. Dender heeft een mechanische oorzaak. Het is niet te voorspellen of het optreedt en in welke mate. Elke schakelaar heeft een ander gedrag. Dender hangt af van allerlei externe factoren, zoals vuil en slijtage. Soms is de duur van de dender een paar tiende milliseconde en soms duurt het enige milliseconden.

Als het signaal een oneven aantal keer van hoog naar laag gaat, is dat hetzelfde als dat het één keer van hoog naar laag gaat. Het signaal van figuur 15.14 gaat een even aantal keer, namelijk veertien keer, van hoog naar laag. Dit betekent dat de led in een paar milliseconde tijd zeven keer aan en uit gaat. Het effect is dat er niets zichtbaars gebeurt. Bij dender lijkt het systeem soms wel en soms niet goed te reageren.

## 15.11 Hardwarematige antidendermaatregelen

Dender (*bouncing*) kan met software en met hardware worden voorkomen. In de literatuur en op het internet zijn verschillende antidenderschakelingen te vinden. Deze zijn te groeperen in een paar fundamentele oplossingen. Soms wordt van het ingangssignaal een one-shot gemaakt, maar meestal wordt het signaal gefilterd. In de schakeling van figuur 15.12 is het ingangssignaal gefilterd door een condensator parallel bij de drukknop te plaatsen.




**Figuur 15.15 : Het signaal van een drukknop met en zonder antidendercondensator.**
Het signaal zonder condensator is dun getekend en licht grijs. Het signaal met condensator is dik en zwart getekend. In de linker figuur wordt de drukknop ingedrukt. Zonder de condensator gaat het signaal denderen. Met de condensator wordt het signaal ook direct nul. Het stijgen van het signaal gaat — vanwege de pullupweerstand — veel langzamer, zodat alleen bij overgang 1 het signaal van hoog naar laag gaat. Bij de andere overgangen is het signaal nog niet hoog genoeg geworden. In de rechter figuur wordt de drukknop losgelaten. Het signaal met condensator zal langzaam stijgen, maar wordt door de dender telkens nul. Na de laatste dender stijgt het signaal wel door en is bij 4 de enige overgang van laag naar hoog.

Het effect van deze condensator is in figuur 15.15 te zien. Bij het indrukken van de knop wordt het signaal snel laag en bij dender gaat het langzaam omhoog. Er is nu maar een enkele overgang van hoog naar laag. Bij het loslaten van de knop wordt het signaal langzaam hoog en bij dender weer snel laag. Pas na de laatste opgaande flank wordt het signaal na enige tijd hoog.

De schmitttrigger wordt besproken in bijlage F.9.

Omdat deze laatste overgang zo traag verloopt, is een schmitttrigger gewenst. In figuur 15.12 is geen schmitttrigger getekend, omdat er in de Xmega al een aanwezig is, zoals in figuur 15.3 te zien is.

De waarde van de condensator hangt af van de waarde van de pullupweerstand en vooral van het soort schakelaar of drukknop dat gebruikt wordt. Meestal is een RC-tijd van 1 ms tot 10 ms een redelijke keuze. Voor een RC-tijd van 1 ms en een pullupweerstand van 10 kΩ is, moet de $C$ minimaal 100 nF zijn.

## 15.12 Softwarematige antidendermaatregelen

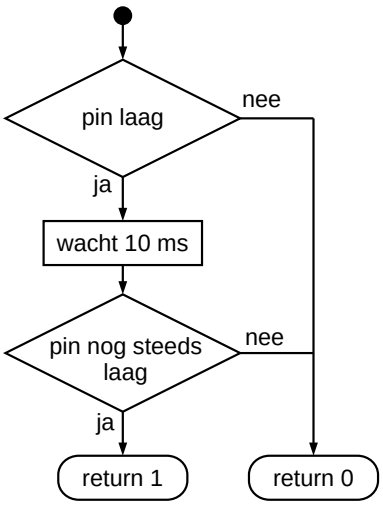
In paragraaf 16.6 staat een voorbeeld van een antidenderalgoritme met een timer en een interrupt. Het voordeel van deze methode is dat het hoofdprogramma ontlast wordt en de tijdplanning (*scheduling*) veel eenvoudiger is.

Dender (*bouncing*) is ook softwarematig te bestrijden. In de literatuur en op het internet zijn vele antidenderalgoritmen (*debouncing algorithms*) te vinden. Deze zijn te verdelen in twee soorten oplossingen: het ene type maakt gebruik van timers en interrupts en het andere gebruikt *polling* en vertragingsfuncties.

Het aantrekkelijke van een softwarematige oplossing is dat er geen extra componenten nodig zijn, de PCB eenvoudiger is en het product goedkoper zal zijn. Als de interne pullupweerstand wordt gebruikt, kan de weerstand uit figuur 15.12 ook worden weggelaten.

**Code 15.10 : De functie `button_pressed` voor het uitlezen van een drukknop.**

```
1  #define  F_CPU 2000000UL
2
3  #include <avr/io.h>
4  #include <util/delay.h>
5
6  #define DEBOUNCE_PERIOD_MS   10
7  #define LOCK_PERIOD_MS       200
8
9  int button_pressed(void)
10 {
11   if ( bit_is_clear(PORTD.IN,PIN3_bp) ) {
12     _delay_ms(DEBOUNCE_PERIOD_MS);
13     if ( bit_is_clear(PORTD.IN,PIN3_bp) ) return 1;
14   }
15
16   return 0;
17 }
18
19 int main(void)
20 {
21   PORTD.DIRCLR   = PIN3_bm;            // input pin button
22   PORTD.PIN3CTRL = PORT_OPC_PULLUP_gc; // enable pull up
23   PORTC.DIRSET   = PIN0_bm;            // output pin led
24
25   while (1) {
26     if ( button_pressed() ) {
27       PORTC.OUTTGL = PIN0_bm;          // toggles output
28       _delay_ms(LOCK_PERIOD_MS);       // lock input
29     }
30   }
31 }
```




**Figuur 15.16 : Het antidenderalgoritme om de drukknop uit te lezen.** Als de ingang na 10 ms nog steeds laag is, is de knop ingedrukt.

In figuur 15.16 staat een antidenderalgoritme voor het uitlezen van een drukknop. Als de drukknop ingedrukt is, wordt er 10 ms gewacht. Mits deze tijd bij de knop past, zal het ingangssignaal stabiel zijn en geen dender meer vertonen. Als de pin dan nog steeds laag is, is de knop ingedrukt.

De functie `button_pressed` uit het programma van code 15.10 is een implementatie van antidenderalgoritme uit figuur 15.16. Het programma hoort bij figuur 15.12 waarbij de pullupweerstand en de condensator niet nodig zijn.

Het hoofdprogramma checkt voortdurend met de functie `button_pressed` of de knop ingedrukt is. Als dat het geval is, verandert de status van de led en wacht het programma 0,2 seconde. Als deze vertragingstijd wordt weggelaten, knippert de led een aantal keer. Een gebruiker houdt een drukknop altijd een fractie van een seconde ingedrukt. Na 0,2 seconde zal de knop losgelaten zijn. Als de gebruiker de knop langer ingedrukt houdt, zal de led blijven knipperen.

**Uitleg code 15.10 regel 11**
`bit_is_clear()`
`bit_is_set()`

Op regel 11 en op regel 13 is de macrodefinitie `bit_is_clear` gebruikt. Deze macro samen met de macro `bit_is_set` is gedefinieerd in `avr/sfr_defs.h`. Dit headerbestand wordt automatisch ingesloten als `avr/io.h` wordt gebruikt. De macrodefinitie `bit_is_set` test of een bit hoog is en `bit_is_clear` test of een bit laag is. Deze macrodefinities hebben twee ingangsparameters `sfr` en `bit`:

```
#define bit_is_clear(sfr, bit) (!(_SFR_BYTE(sfr) & _BV(bit)))
#define bit_is_set(sfr, bit)   ( (_SFR_BYTE(sfr) & _BV(bit)))
```

Parameter `sfr` is de naam van het register en `bit` het nummer van de bit. De uitdrukking:

```
bit_is_clear(PORTD.IN,PIN3_bp)
```

is identiek met:

```
! (PORTD.IN & PIN3_bm)
```

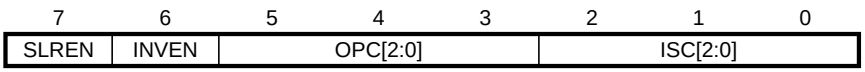
Bij `bit_is_clear` en `bit_is_set` wordt het bitnummer (`PIN0_bp`) gebruikt en niet het bitmasker (`PIN0_bm`).

**Regel 11**
`PIN3_bp`

De macro's `PINn_bp` zijn gedefinieerd in `avr/io.h` en representeren bitnummers of bitposities. De macro `PIN3_bp` komt overeen met de waarde 3. Verwar de bitpositie `PIN3_bp` niet met het bitmasker `PIN3_bm`, zie ook code 15.5.

**Regel 22**
`PORTD.PIN3CTRL`

Met het pincontrolregister kan een individuele pinaansluiting worden aangepast. `PORTD.PIN3CTRL` is het pincontrolregister van pin 3 van poort D. Figuur 15.17 toont de inhoud van dit register. `SRLEN` staat voor *slew rate limit enable*, hiermee wordt




**Figuur 15.17 : Het pincontrolregister `PINnCTRL`.**

de slew rate gelimiteerd. `INVEN` staat voor *inverted i/o enable*, hiermee wordt de polariteit van de in- of uitgang geïnverteerd. `OPC` staat voor *output and pull configuration*. Deze bits stellen onder andere de pullup- en pulldownschakeling van de generieke IO in. `ISC` staat voor *input/sense configuration*. Deze bits regelen de triggergevoeligheid van een ingang bij interrupts en events.

**Regel 22**
`PORT_OPC_PULLUP_gc`

De drie `OPC`-bits uit het `PINnCTRL`-register worden gebruikt om de aansluiting als wired-and of wired-or te configureren en om de pullup- en pulldownschakeling in te stellen. Tabel 15.2 geeft de verschillende instelmogelijkheden.
In code 15.10 hebben de `OPC`-bits de waarde `PORT_OPC_PULLUP_gc` en heeft ingang 3 van poort D een interne pullupweerstand.

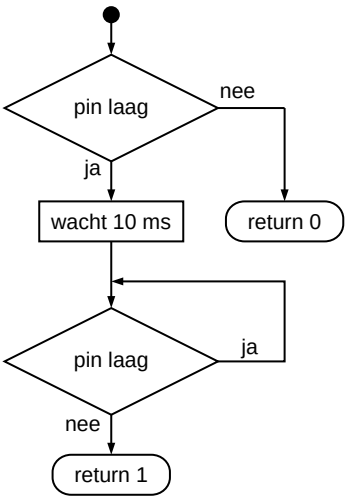
### Alternatief antidenderalgoritme dat altijd maar één keer reageert

In code 15.10 is de ontdendertijd 10 ms en wacht het hoofdprogramma 0,2 s. De tijdvertragingen voor het ontdenderen en het wachten hangen sterk af van het type knop en van wat de applicatie moet doen. Als de knop langer dan 0,2 s wordt ingedrukt, wordt er opnieuw een druk op de knop geregistreerd. Dat

**Tabel 15.2 : De uitgangs- en pullup/pulldown-configuratie.**

| OPC[2:0] | groepsconfiguratie | betekenis |
|---|---|---|
| 000 | PORT_OPC_TOTEM_gc | normaal |
| 001 | PORT_OPC_BUSKEEPER_gc | buskeeper |
| 010 | PORT_OPC_PULLDOWN_gc | pulldown |
| 011 | PORT_OPC_PULLUP_gc | pullup |
| 100 | PORT_OPC_WIREDOR_gc | wired-OR |
| 101 | PORT_OPC_WIREDAND_gc | wired-AND |
| 110 | PORT_OPC_WIREDORPULL_gc | wired-OR met pulldown |
| 111 | PORT_OPC_WIREDANDPULL_gc | wired-AND met pullup |

kan ongewenst zijn. Natuurlijk kan de wachttijd langer gemaakt worden, maar dan bestaat juist de kans dat een korte indruk van de knop gemist wordt. Bij een toetsenbord is het juist plezierig dat de aanslag voortdurend herhaald wordt zolang de toets ingedrukt blijft.




**Figuur 15.18 : Het antidenderalgoritme dat wacht op het loslaten van de drukknop.**

**Code 15.11 : Alternatief voor uitlezen drukknop.** Deze functie `button_pressed` detecteert dat de drukknop losgelaten wordt.

```
 9  int button_pressed(void)
10  {
11    if ( bit_is_clear(PORTD.IN,PIN3_bp) ) {
12      _delay_ms(DEBOUNCE_PERIOD_MS);
13      while ( bit_is_clear(PORTD.IN,PIN3_bp) ) ;
14      return 1;
15    }
16    return 0;
17  }
```

Een alternatief is om de functie `button_pressed` te laten wachten totdat de drukknop losgelaten wordt. Het algoritme voor deze methode staat in figuur 15.18 en de alternatieve functie `button_pressed` staat in code 15.11. Het enige verschil met de functie uit code 15.10 is dat de **`if`** op regel 26 vervangen is door een **`while`**.

Het nadeel van de functie `button_pressed` uit code 15.11 is dat het programma in de functie blijft hangen zolang de gebruiker de knop ingedrukt houdt. Zeker bij programma's met een ingewikkelde tijdplanning (*scheduling*) kan dat er toe leiden dat het programma niet goed zal functioneren.

### Alternatieve functie `button_pressed` met parameters

De functies `button_pressed` uit code 15.10 en code 15.11 werken alleen bij de aansluiting 3 van poort D. Voor een andere aansluiting moeten de functies worden aangepast. In code 15.12 staat een functie `button_pressed` met twee parameters: een pointer `p` die naar de poort wijst en een integer `bit` voor het bitnummer.

**Code 15.12 : De functie `button_pressed` met parameters.**

```
1  #define  F_CPU 2000000UL
2
3  #include <avr/io.h>
4  #include <util/delay.h>
5
6  #define LOCK_PERIOD_MS       200
7
8  int button_pressed(PORT_t *p, uint8_t bit)
9  {
10   if ( bit_is_clear(p->IN, bit) ) {
11     _delay_ms(10);
12     loop_until_bit_is_set(p->IN, bit);
13     return 1;
14   }
15   return 0;
16 }
17
18 int main(void) {
19   PORTD.DIRCLR   = PIN3_bm;            // input pin button
20   PORTD.PIN3CTRL = PORT_OPC_PULLUP_gc; // enable pull up
21   PORTC.DIRSET   = PIN0_bm;            // output pin led
22
23   while (1) {
24     if ( button_pressed(&PORTD, PIN3_bp) ) {
25       PORTC.OUTTGL = PIN0_bm;          // toggles output
26       _delay_ms(LOCK_PERIOD_MS);       // lock input
27     }
28   }
29 }
```

Om de betreffende poort te benaderen moet aan de functie het adres van de poort worden meegegeven. Pointer `p` wijst daarom naar het adres van een datastructuur `PORT_t`. Bij de aanroep van `button_pressed` op regel 24 staat voor de variabele `PORTD` de adresoperator `&`, zodat het adres van `PORTD` aan de functie wordt meegegeven.

In de functie is `p` een pointer naar de registers van de poort. Het scheidingsteken tussen het veld en een pointer naar een datastructuur is een pijltje `->`. Het ingangsregister `IN` van de poort wordt op regel 10 en regel 12 daarom gevonden met `p->IN`.

Op regel 12 is in plaats van een **`while`** de macrodefinitie `loop_until_bit_is_set` gebruikt. Het headerbestand `sfr_defs.h` bevat naast de definities `bit_is_clear` en `bit_is_set` nog twee macro's `loop_until_bit_is_clear` en `loop_until_bit_is_set`. Deze macro's wachten totdat de bit in het register respectievelijk laag is en hoog is. De uitdrukking:

```
loop_until_bit_is_set(p->IN, bit);
```

is identiek met:

```
while ( ( bit_is_clear(p->IN, bit) ) ;
```

Bij de Xmega-stijl op basis van datastructuren is het maken van algemene functies met parameters, zoals de functie `button_pressed` in code 15.12, relatief eenvoudig.