Pulse

Pulse Accumulator

- Veel signalen in de buitenwereld zijn pulsvormig.

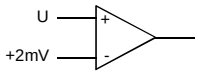
- Veel meetsensoren leveren een pulsvormige uitgangsspanning:
  - Tachometer (toerenteller)
  - Hartslagsensor
  - Snelheidsmeter (fietscomputer)
  - Bewegingssensor
  - enz...
- Om deze signalen te kunnen verwerken met een µC (microcontroller) moet vaak het aantal pulsen geteld worden.
- Veel µC's hebben een ingebouwde pulse accumulator.

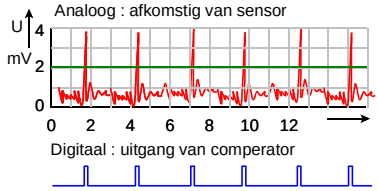
THRijswijk

Pulse

Analoog naar digitaal

- Een analoge puls (bijvoorbeeld hartslag) kan met een comperator worden omgezet naar een digitale puls.





THRijswijk

Pulse Accumulator

68HC11

- De puls accumulator staat na reset **uit**. Aanzetten met bit PAEN in het PACTL register.
- De pulse accumulator van de 68HC11 is een 8 bits teller (PACNT register) die de pulsen kan tellen op pin PA7.
- PA7 moet dan wel op input staan (DDRA7 bit in het PACTL register).
- Je kunt zelf bepalen of je de opgaande of neergaande flanken telt (PEDGE bit in PACTL register).
- Maximale frequentie is 1 MHz.

THRijswijk

Pulse Accumulator

68HC11

- PAOVF bit in TFLG2 reg: wordt 1 als PACNT van 0xFF naar 0x00 "overloopt".
- PAIF bit in TFLG2 reg: wordt 1 als active edge wordt gedetecteerd.
- Deze bits kun je weer 0 maken door er een 1 naar toe te schrijven.
- Als PAOVI bit in TMSK2 reg is geset treedt een interrupt op als PAOVF 1 wordt. Vector FFDC.
- Als PAII bit in TMSK2 reg is geset treedt een interrupt op als PAIF 1 wordt. Vector FFDA.

THRijswijk

Voorbeeld

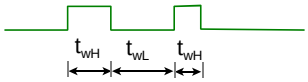
```
/* Genereer een interrupt na 10 opgaande
                              flanken op PA7. */
volatile byte* ...=(byte*)...;
void paovi_isr(void) __attribute__((interrupt));

void paovi_isr(void) {
   (*portb)++;
   *pacnt=0xFF-9;
   *tflg2=0x20;
}
int main() {
   *portb=0; *tmsk2|=0x20;
   *pactl|=0x50; *pacnt=0xFF-9;
   while (1); return 0;
}
typedef void (*isr)(void);
#define E (isr)0xffff
extern void _start(void);
isr vectors[32] __attribute__ ((
                              section (".vector"))) = {
   E, E, E, E, E, E, E, E,
   E, E, E, E, E, E, paovi_isr, E,
   E, E, E, E, E, E, E, E,
   E, E, E, E, E, E, E, _start
};
```

THRijswijk

Pulse width

Pulse Accumulator

- Soms moet de puls width worden bepaald.



- Bijvoorbeeld:
  - Snelheid meten.
  - Periodetijd meten.
  - enz...
- Om deze pulse width te kunnen meten moet geteld worden zolang het signaal hoog (of laag) is.

THRijswijk

Pulse width

Pulse Accumulator

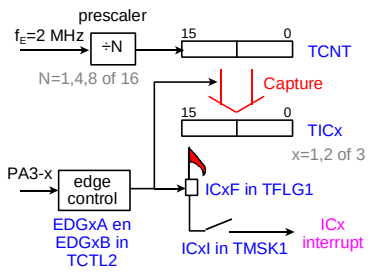
- PAMOD bit in PACTL=**0**: PA telt actieve **flanken** op PA7.
  - PEDGE=**0**: telt **neer**gaande flanken.
  - PEDGE=**1**: telt **op**gaande flanken.
- PAMOD bit in PACTL=**1**: PA telt **zolang** PA7 actief is.
  - PEDGE=**0**: telt als PA7 **hoog** is.
  - PEDGE=**1**: telt als PA7 **laag** is.
  - Telt elke **32 µs** (bij 8MHz clock).
- Nauwkeuriger meten is mogelijk met behulp van de **input capture** timer.

THRijswijk

Input Capture

Free running counter

- Met behulp van een **input capture systeem** kan het "tijdstip" waarop een signaal verandert worden gelogd.



THRijswijk

Input Capture

Free running counter

- Prescaler wordt ingesteld met PR1 en PR0 in TMSK2 register.
- TOF bit in TFLG2 reg: wordt 1 als TCNT van 0xFFFF naar 0x0000 "overloopt".
- ICxF bit in TFLG1 reg: wordt 1 als active edge wordt gedetecteerd.
- Deze bits kun je weer 0 maken door er een 1 naar toe te schrijven.
- Als TOI bit in TMSK2 reg is geset treedt een interrupt op als TOF 1 wordt. Vector FFDE.
- Als ICxI bit in TMSK1 reg is geset treedt een interrupt op als ICxF 1 wordt. Vector FFF0-2x.

x=1,2 of 3

THRijswijk

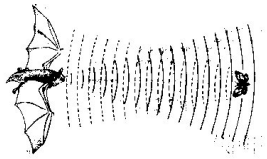
Input Capture

Edge control

x=1,2 of 3

EDGxB	EDGxA	edge
0	0	geen
0	1	op
1	0	neer
1	1	beide

! Methode om afstand te meten:
" Zend puls en ontvang reflectie verschil = maat voor afstand.

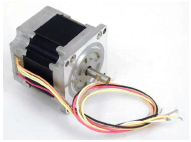


Pulse

Pulse Generator

! Veel signalen in de buitenwereld zijn pulsvormig.

! Veel actoren kunnen met een pulsvormig signaal worden aangestuurd:
" Lamp.
" Stappenmotor.
" Telefoonkiezer.
! puls en toon!
" enz...

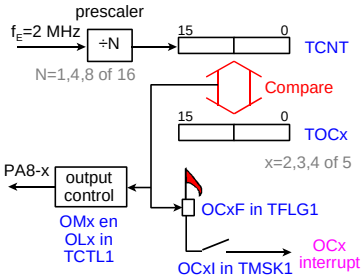


! Om deze signalen te kunnen opwekken hebben veel µC's een ingebouwde pulse generator.

Output Compare

Free running counter

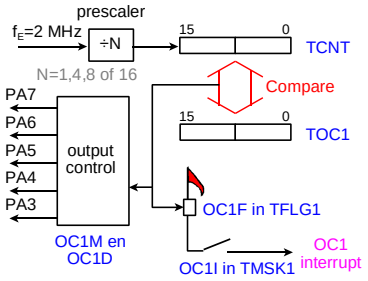
! Met behulp van een **output compare systeem** kan het "tijdstip" waarop een signaal verandert worden gekozen.



Output Compare

Free running counter

! Met behulp van **output compare 1** kunnen op een bepaald "tijdstip" meerdere signalen veranderd worden.



Output Compare

Free running counter

! Prescaler wordt ingesteld met PR1 en PR0 in TMSK2 register.
! TOF bit in TFLG2 reg: wordt 1 als TCNT van 0xFFFF naar 0x0000 "overloopt".
! OCxF bit in TFLG1 reg: wordt 1 als TCNT == TOCx
! Deze bits kun je weer 0 maken door er een 1 naar toe te schrijven.
! Als TOI bit in TMSK2 reg is geset treedt een interrupt op als TOF 1 wordt. Vector FFDE.
! Als OCxI bit in TMSK1 reg is geset treedt een interrupt op als OCxF 1 wordt. Vector FFEA-2x.

x=1,2,3,4 of 5

Output Compare

Output control

x=2,3,4 of 5

OMx	OLx	output
0	0	geen
0	1	toggle
1	0	laag
1	1	hoog

! **Output Compare 1.**
" Kan behalve pin PA7 ook pinnen PA6, PA5, PA4 en PA3 aansturen.
" Zie OC1M en OC1D registers.
" Handig voor aansturen stappenmotor.
! **Forced Output Compare.**
" Schrijf een 1 naar FOCx bit in CFORC register om output compare te forceren (COxF wordt niet geset).

x=1,2,3,4 of 5

Opgave

Stappenmotor

! **Gegevens:**
" Stappenmotor met een stapgrootte van 1.8º.
" Stappenmotoraansturing heeft 4 ingangen die als volgt aangestuurd moeten worden:

stap	PA6	PA5	PA4	PA3
0	1	0	0	1
1	1	0	1	0
2	0	1	1	0
3	0	1	0	1

" Richting wordt bepaald door volgorde: 0,1,2,3,0,1,... => rechtsom
! **Opgave:**
" Laat de motor draaien PC3 t/m PC0 snelheid N omw/sec
" Richting bepalen: PC4 0=linksom

Opgave

Stappenmotor

! Maximale snelheid
" 15 omw/s
" 1 omw = 360 graden
" stappenmotor: 1,8 graden/stap
" 1 omw = 360/1,8 = 200 stappen
" 200x15 = 3000 stappen/s
" tijd tussen stappen = 333,3 µs
! Bijna maximale snelheid
" 14 omw/s
" 200x14 = 2800 stappen/s
" tijd tussen stappen = 357,1 µs
" verschil met max = 23,8 µs
! Conclusie
" Gebruik deelfactor van 1
" **0,5 µs/timertick**
! Formule
" ticks=2x(1000000/(snelheidx200))
" **ticks=10000/snelheid**

Opgave

Stappenmotor

! Gebruik OC1 (kan meerdere uitgangen gelijk aansturen)

! OC1M = **masker**
" 01111000 = 0x78

! OC1D = **patroon** (rechtsom)
" 01001000 = 0x48
" 01010000 = 0x50
" 00110000 = 0x30
" 00101000 = 0x28

Stappenmotor

```
typedef unsigned char byte;
typedef unsigned short word;

struct Ports {
   byte porta;
   byte reserved1;
   byte pioc;
   byte portc;
   byte portb;
   byte portcl;
   byte reserved2;
   byte ddrc;
   byte portd;
   byte ddrd;
   byte porte;
};
```

```
struct Timer {
   byte cforc;
   byte oc1m;
   byte oc1d;
   word tcnt;
   word tic1;
   word tic2;
   word tic3;
   word toc1;
   word toc2;
   word toc3;
   word toc4;
   word toc5;
   byte tctl1;
   byte tctl2;
   byte tmsk1;
   byte tflg1;
   byte tmsk2;
   byte tflg2;
};
```

THRijswijk

Stappenmotor

```
typedef signed char sbyte;
int main() {
   volatile struct Ports* ports=
      (volatile struct Ports*)0x1000;
   volatile struct Timer* timer=
      (volatile struct Timer*)0x100b;
   byte patroon[]={0x48,0x50,0x30,0x28};
   sbyte i=0; byte snelheid=0;
   timer->oc1m=0x78;
   timer->oc1d=patroon[0];
   timer->toc1=timer->tcnt+50;
   while (1) {
      while ((timer->tflg1&0x80)==0x00) /* wacht */;
      timer->tflg1=0x80;
      snelheid = ports->portc&0x0f;
      if (snelheid==0) {
         timer->toc1+=0x1000;
      }
      else {
         timer->toc1+=10000/snelheid;
         timer->oc1d=patroon[i];
         if ((ports->portc&0x10)==0x10) {
            if (++i==4) i=0;
         }
         else {
            if (--i==-1) i=3;
   } } } return 0;
}
```

THRijswijk

Opgave

Stappenmotor

! Gegevens:

" Zelfde als de vorige keer.

! Opgave:

" Laat de motor draaien met behulp van interrupts zodat in hoofdprogramma andere taken kunnen worden uitgevoerd.

THRijswijk

Stappenmotor

```
void oc1_isr(void) __attribute__((interrupt));

typedef void (*isr)(void);
#define E (isr)0xffff
extern void _start(void);

isr vectors[32] __attribute__
                  ((section (".vector"))) = {
   E, E, E, E, E, E, E, E,
   E, E, E, E, E, E, E, E,
   E, E, E, E, oc1_isr, E, E, E,
   E, E, E, E, E, E, E, _start
};

volatile byte snelheid=0;
volatile byte rechts=0;
```

THRijswijk

Stappenmotor

```
void oc1_isr(void) {
   static byte patroon[]={
                        0x48, 0x50, 0x30, 0x28};
   volatile struct Timer* timer=
                  (volatile struct Timer*)0x100b;
   static sbyte i=0;
   timer->tflg1=0x80;
   if (snelheid==0) {
      timer->toc1+=0x1000;
   }
   else {
      timer->toc1+=10000/snelheid;
      timer->oc1d=patroon[i];
      if (rechts) {
         if (++i==4)
            i=0;
      }
      else {
         if (--i==-1)
            i=3;
      }
   }
}
```

THRijswijk

Stappenmotor

```
#include "lcd.h"

void display(void) {
   lcd_setpos(0,0); lcd_puts("Snelheid = ");
   lcd_putc('0'+snelheid/10);
   lcd_putc('0'+snelheid%10);
   lcd_setpos(2,0); lcd_puts("Richting = ");
   if (rechts) lcd_puts("rechts");
   else lcd_puts("links "); /* let op spatie! */
}

int main() {
   volatile struct Ports* ports=
                        (volatile struct Ports*)0x1000;
   volatile struct Timer* timer=
                        (volatile struct Timer*)0x100b;
   lcd_init(); display();
   timer->oc1m=0x78; timer->oc1d=0;
   timer->toc1=timer->tcnt+50; timer->tmsk1=0x80;
   while (1) {
      byte s=snelheid, r=rechts;
      snelheid = ports->portc&0x0f;
      if (snelheid<3) /*alleen omkeren bij snelheid<3*/
         rechts = (ports->portc&0x10)==0x10;
      if (rechts!=r||snelheid!=s) display();
   }
   return 0;
}
```

THRijswijk