1

Interrupts

In dit hoofdstuk worden interrupts en het interruptmechanisme van de ATmega32 behandeld.

1.1 Polling

De ATmega32 kan met behulp van een wachtlus reageren op signalen van externe bronnen. De externe bronnen kunnen zo aan de processor aangeven dat er bijvoorbeeld data beschikbaar is en dat afhandeling is gewenst. Dat werkt als volgt. De ATmega32 leest een byte in van een van de I/O-poorten. Vervolgens wordt de bit die getest moet worden met behulp van een bitmasker gefilterd. Is de geteste bit niet actief, dan wordt opnieuw een byte ingelezen en begint het testen opnieuw. Is de geteste bit wel actief, dan wordt de wachtlus verlaten en wordt de rest van het programma uitgevoerd. Deze methode wordt *polling* genoemd. In het vakgebied van de Operating Systems wordt dit *busy waiting* genoemd.

In listing 1.1 is een voorbeeld van zo'n wachtlus te zien. Eerst wordt de status van de pinnen van Port A ingelezen door middel van een `in`-instructie. De `andi`-instructie zorgt ervoor dat de minst significante bit wordt gefilterd; de overige bits worden op 0 gezet. De `andi`-instructie past ook de Z-vlag aan. Als blijkt dat de minst significante bit een 0 is dan wordt de Z-vlag op 1 gezet, anders wordt de Z-vlag op 0 gezet (merk op dat de overige bits op 0 gezet zijn). Als blijkt dat de minst significante bit een 0 is, dan zorgt de `breq`-instructie ervoor dat weer naar het begin van de wachtlus wordt gesprongen. Is de minst significante bit een 1, dan wordt de lus verlaten en wordt de eerstvolgende instructie uitgevoerd.

```
1 loop:   in   r16,PINA   ; read in pins Port A
2         andi r16,0x01   ; filter LSB, Z-flag = 1 if bit = 0
3         breq loop       ; again if bit is 0
```

Listing 1.1: *Polling.*

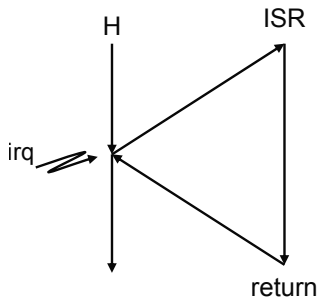
Deze manier van detectie werkt eenvoudig en snel. Het programma (de wachtlus) is eenvoudig en er zijn slechts enkele klokpulsen nodig om de status van een extern signaal te bepalen. Het nadeel is natuurlijk dat de processor niets anders kan doen.

1.2 Interrupts

Slimmer is om de hardware aan de processor door te laten geven dat de status van een signaal is veranderd, zodat de processor direct actie kan ondernemen. Het lopende programma wordt dan *geïnterrumpeerd* (onderbroken). Dit wordt een *interrupt* genoemd. Een aanvraag voor een onderbreking (het signaal) wordt een *interrupt request* (IRQ) genoemd.

Nadat een interrupt request door de processor is herkend, wordt het lopende programma onderbroken (de huidige instructie wordt afgemaakt) en wordt een *interrupt service routine* (ISR) uitgevoerd. De ISR handelt dan de interrupt verder af. Daarna gaat de processor verder waar het gebleven was.

We beelden dit uit in figuur 1.1. Hierin stelt H de uitvoer van het lopende programma voor. Op een gegeven moment wordt er een interrupt geregistreerd door de processor. Het lopende programma wordt verlaten en de processor gaat de ISR uitvoeren. Aan het einde van de ISR wordt teruggekeerd naar het lopende programma H.



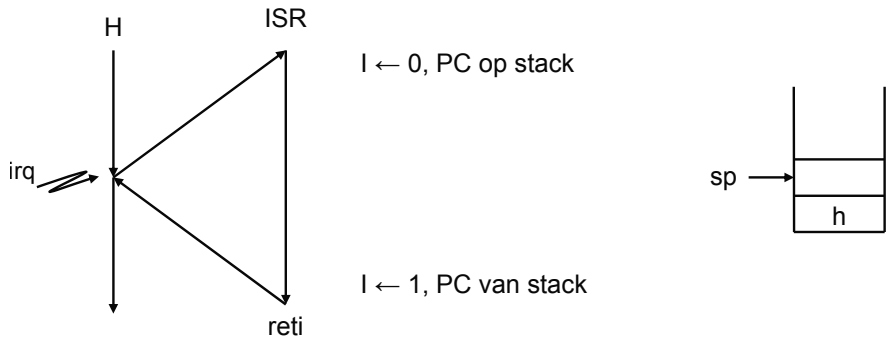
Figuur 1.1: *Eenvoudige voorstelling van een interruptafhandeling.*

ISR's lijken veel op gewone subroutines, maar er zijn verschillen:

- Een ISR wordt gestart door een interrupt, niet door een instructie[1].
- Direct nadat de interrupt is herkend, wordt de Global Interrupt Enable vlag (de I-vlag) in het Status Register op 0 gezet. Dit zorgt ervoor dat de ISR niet kan worden onderbroken door een (nieuwe) interruptaanvraag.
- Voor terugkeer moet de instructie `RETI` (Return From Interrupt) gebruikt worden. Deze instructie zet de Global Interrupt Enable vlag weer aan.

Interrupts en subroutines hebben wel één overeenkomst: in beide gevallen wordt de program counter (PC) op de stack gezet. Dit wordt uitgebeeld in figuur 1.2. Omdat het terugkeeradres op de stack wordt gezet, moet de stack pointer (SP) worden geïnitialiseerd vóórdat de interrupts mogen worden afgehandeld. Dat betekent dat direct naar het starten van het lopende programma, bijvoorbeeld na een reset, de interrupts nog niet mogen worden afgehandeld omdat de stack pointer nog niet is geïnitialiseerd.

[1] Er zijn processoren die instructies hebben die software-interrupts genereren, zoals de Intel x86-familie.



Figuur 1.2: *Interruptafhandeling: de Program Counter wordt op de stack gezet.*

Twee instructies beïnvloeden direct de I-flag:

`sei` - set Global Interrupt Enable flag, interrupts worden afgehandeld.

`cli` - clear Global Interrupt Enable flag, interrupts worden geblokkeerd.

De I-vlag is gepositioneerd op bit 7 van het Status Register (SREG). Zie figuur 1.3.

7	6	5	4	3	2	1	0
I	T	H	S	V	N	Z	C
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figuur 1.3: *Status Register SREG.*

Pas na het initialiseren van de stack pointer mag de instructie `sei` gegeven worden.

1.3 Interruptbronnen

De ATmega32 kent veel interruptbronnen. Zo kan de ATmega32 reageren op drie externe interrupts: INT0, INT1 en INT2. Verder kan de analoog-digitaal converter (ADC) een interrupt geven als de conversie klaar is. De seriële interface (USART) kent naast een interrupt wanneer een karakter is ontvangen ook een interrupt voor wanneer een karakter is verzonden. De nog te bespreken timer/counters (zie hoofdstuk 2) kunnen een interrupt afgeven wanneer een timer/counter “over de kop” gaat, dus wanneer een timer/counter van de hoogste stand naar de laagste stand gaat. Verder zijn er nog interrupts mogelijk van de EEPROM, de TWI- en de SPI-interface.

Al deze interruptbronnen hebben een eigen interrupt-enable-bit. Om een interrupt van een bron ook daadwerkelijk te laten plaatsvinden, moet deze bit geactiveerd zijn (logisch 1) én de I-vlag moet logisch 1 zijn.

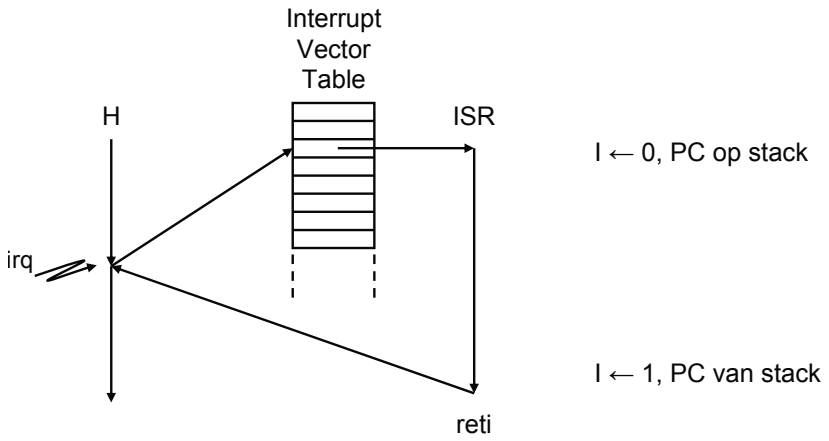
De ATmega32 heeft geen *software interrupts*. Dat zijn software-instructies die interrupts genereren. Deze zijn wel na te bootsen met INT0, INT1 en INT2.

1.4 De Interrupt Vector Table

We hebben nu besproken hoe het verwerken van een interrupt wordt uitgevoerd. Maar nu rijst de vraag: waar moet de ISR beginnen? We zouden hiervoor een vast adres in de

Flash-ROM kunnen kiezen, bijvoorbeeld adres 0x1000. Vervolgens reserveren we 512 bytes voor de ISR. De volgende ISR begint dan op adres 0x1200. Hoewel dit natuurlijk te realiseren is, kleven hier wel wat bezwaren aan. Ten eerste moet de ISR altijd op adres 0x1000 beginnen. Ten tweede kan de ISR niet groter zijn dan 512 bytes. Als de ISR korter is dan 512 bytes, dan gebruiken we een gedeelte van de Flash-ROM niet (bedenk dat de volgende ISR op adres 0x1200 begint). Ten derde moet de Flash-ROM minimaal 4608 (= 0x1200) bytes groot zijn. We zien nu dat een vast adres en een vaste lengte voor een ISR niet flexibel is.

We kunnen deze drie problemen oplossen door gebruik te maken van een *Interrupt Vector Table*.



Figuur 1.4: *Interruptafhandeling: gebruik van de Interrupt Vector Table.*

1.5 Stappen in de interruptverwerking

Als een interruptsignaal bij de processor binnenkomt, worden de volgende stappen genomen:

1. De processor maakt de instructie af die op dat moment wordt uitgevoerd. Dit is een instructie in het programma.

2. De processor slaat het adres van de volgende instructie op de stack op. Dit adres staat in de program counter. De Global Interrupt Enable vlag wordt op 0 gezet.

3. De processor springt naar de *interrupt vector*. Dit is een vaste geheugenlocatie in de Flash-ROM.

4. Vanuit de interrupt vector wordt gesprongen naar naar het adres van de daadwerkelijke interrupt service routine (ISR).

5. Bij terugkeer uit de ISR wordt de program counter geladen met de adres dat op de stack gezet was. De Global Interrupt Enable vlag wordt op 1 gezet.

6. De processor vervolgt het uitvoeren van het programma. Er wordt altijd één instructie uitgevoerd voordat een eventuele nieuwe interrupt wordt verwerkt.

Tabel 1.1: *Interrupt vectortabel voor de ATmeag32(A).*

vector #	ROM-adres	Bron	Omschrijving
1	0x000	RESET	Reset vector
2	0x002	INT0	External Interrupt request 0
3	0x004	INT1	External Interrupt request 1
4	0x006	INT2	External Interrupt request 2
5	0x008	TIMER2_COMP	Timer/Counter2 Compare Match
6	0x00A	TIMER2_OVF	Timer/Counter2 Overflow
7	0x00C	TIMER1_CAPT	Timer/Counter1 Capture Event
8	0x00E	TIMER1_COMPA	Timer/Counter1 Compare Match A
9	0x010	TIMER1_COMPB	Timer/Counter1 Compare Match B
10	0x012	TIMER1_OVF	Timer/Counter1 Overflow
11	0x014	TIMER0_COMP	Timer/Counter0 Compare Match
12	0x016	TIMER0_OVF	Timer/Counter0 Overflow
13	0x018	SPI, STC	Serial Transfer Complete
14	0x01A	USART, RXC	USART, Rx Complete
15	0x01C	USART, UDRE	USART Data Register Empty
16	0x01E	USART, TXC	USART, Tx Complete
17	0x020	ADC	ADC Conversion Complete
18	0x022	EE_RDY	EEPROM Ready
19	0x024	ANA_COMP	Analog Comparator
20	0x026	TWI	Two-wire Serial Interface
21	0x028	SPM_RDY	Store Program Memory Ready

Noot: het ROM-adres is in words.

1.6 Interruptresponstijd

De responstijd voor alle ingeschakelde interrupts van de ATmega32 is minimaal vier klokcycli. Tijdens deze vier klokcyclusperiode wordt de program counter op de stack geplaatst (twee bytes) en wordt de program counter geladen met het vectoradres van de interrupt. Na vier klokcycli wordt gesprongen naar het vectoradres van de interrupt die wordt gehonoreerd. De instructie op de vector is normaal gesproken een sprong naar de interruptroutine en deze sprong duurt drie klokcycli.

Als een interrupt optreedt tijdens de uitvoering van een instructie met meerdere cycli dan wordt deze instructie voltooid voordat de interrupt wordt gehonoreerd. Staat de ATmega32 in slaapstand dan wordt de responstijd met vier klokcycli verhoogd. Deze verhoging komt bovenop de opstarttijd van de geselecteerde slaapmodus.

Een terugkeer (return) uit een interruptroutine duurt vier klokcycli. Tijdens deze vier klokcycli wordt de program counter van de stack gehaald (twee bytes) en wordt de stack pointer met twee verhoogd. Tevens word de I-bit in het statusregister op 1 gezet.

1.7 Context switch

```
1 ISR:    push r16         ; push R16
2         in   r16,SREG    ; SREG in I/O memory ...
```

```
3         push r16          ; ... so push flags via R16
4
5         pop  r16          ; pop flags via R16
6         out  SREG,r16
7         pop  r16          ; pop R16
8         reti              ; return
```

Listing 1.2: *Opslaan van de vlaggen.*

1.8 Externe interrupts

De General Interrupt Control Register (GICR) heeft drie bits waarmee de INT's kunnen worden geactiveerd.

Een logische 1 in het betreffende bit activeert de INT.

De I-flag moet 1 zijn om interrupt ook daadwerkelijk te herkennen.

7	6	5	4	3	2	1	0
INT0	INT1	INT2	–	–	–	IVSEL	IVCE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figuur 1.5: *GICR register*

2

Timer/Counter 0

In veel applicaties moet gewacht worden, bijvoorbeeld elke seconde een meting doen, waterklep 1 seconde open. Het meten van deze tijd (wachten) is op te lossen door middel van de bekende wachtlus. De processor kan dan echter geen andere taken uitvoeren en dat is in veel situaties wél gewenst. Denk hierbij aan Real Time systemen en besturingssystemen. Beter is deze tijd hardwarematig te meten. Een *timer* kan dan uitkomst bieden. Een timer is niets anders dan een teller die op iedere klokpuls, bijvoorbeeld van de oscillator, wordt verhoogd. Na een bepaald aantal klokpulsen heeft de timer de hoogste stand bereikt en begint weer opnieuw. Er wordt dan een overflow-signaal afgegeven. Dit signaal genereert, indien geactiveerd, een interrupt.

Dezelfde schakeling kan ook gebruikt worden om extern aangeboden pulsen te tellen. Dit wordt dan een *counter* genoemd. Met de combinatie van een counter en een timer kan een *frequentiemeter* gemaakt worden. Eén timer meet de tijd van 1 seconde en een (andere) counter telt in die tijd het aantal aangeboden pulsen.

Timer/Counter 0 is de eerste van de drie timer/counters van de ATmega32 die besproken worden. Het is een 8-bits timer/counter wat inhoudt dat het kan tellen tussen 0 en 255. Als de timer/counter intern opgewekte klokpulsen telt spreken we van een timer. Bij het tellen van extern opgewekte klokpulsen spreken we van een counter. Technisch gezien is er geen verschil tussen de hardware van een timer en een counter. Het verschil is dus alleen wat de klokbron is.

Timer/Counter 0 heeft vier werkmodi:

- Normal mode: de Timer/Counter 0 telt van 0 tot en met 255 en begint dan weer bij 0. Zowel interne ale externe klokpulsen kunnen geteld worden.
- CTC-mode: de Timer/Counter telt van 0 tot een van te voren opgegeven maximale waarde en begint dan weer op 0. Zowel interne als externe klokpulsen kunnen geteld worden.
- Fast PWM-mode: de Timer/Counter telt van 0 tot 255 en begint dan weer op 0.

Deze modus wordt voornamelijk gebruikt om een signaalvorm op één van de externe pinnen van de ATmega32 te genereren.

- Phase correct PWM-mode: de Timer/Counter telt omhoog van 0 tot en met 255 en telt dan weer omlaag tot en met 0. Deze modus wordt voornamelijk gebruikt om een signaalvorm op één van de externe pinnen van de ATmega32 te genereren.

In alle werkmodi is het mogelijk om een interrupts te genereren. Elke interrupt is gekoppeld aan een eigen interruptvector.

```
1     ldi r16,16     ; load R16 with value 16
2     LDI R16,16     ; the same
```

Listing 2.1: *Test.*

```
1 int main(void) {
2     return 0;
3 }
```

Listing 2.2: *Test.*

2.1 De registers van Timer/Counter 0

De besturing van Timer/Counter 0 is vastgelegd in het TCCR0-register, zie figuur 2.1. Dit register kan zowel gelezen als geschreven worden.

7	6	5	4	3	2	1	0
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figuur 2.1: *TCCR0 register*

Bit 7 – FOC0: Force Output Compare
De FOC0-bit is alleen actief als we de WGM0*x*-bits non-PWM-modus specificeren. Bij het specificeren van één van de twee PWM-modi moet deze bit altijd als 0 geschreven worden. Als deze bit met een 1 geladen wordt, zal de signaalvormgenerator een directe compare match krijgen en zal de OC0-uitgang veranderen zoals is vastgelegd in de COM0*x*-bits.

Bits 3,6 – WGM0*x* signaalvormgeneratorbits
Deze bits bepalen de telvolgorde van de Timer/Counter, de maximale telwaarde en het type van de signaalvormgenerator. Zie tabel 2.1.

Bits 5,4 – COM0*x* Compare Match Output Mode
Deze bits bepalen het gedrag van de Output Compare pin (OC0). Merk op dat de Data Direction (DDR) bit van de OC0-pin op 1 gezet moet worden om de uitgang te activeren. Tabel 2.2 geeft de functionaliteit van de COM0*x*-bits weer in non-PWM-modus (Normal en CTC).

Tabel 2.1: *Signaalvormgeneratie*

WGM01	WGM00	Modus	Top	Update OCR0	TOV0-vlag
0	0	Normaal	255	direct	255
0	1	Phase Correct PWM	255	255	0
1	0	CTC	OCR0	direct	255
1	1	Fast PWM	255	0	255

Tabel 2.2: *Compare Match uitgang, non-PWM-modus.*

COM01	COM00	Beschrijving
0	0	Normale poortwerking, OC0 is gedeactiveerd
0	1	Verander OC0 op een compare match (toggle)
1	0	Zet OC0 op 0 op een compare match
1	1	Zet OC0 op 1 op een compare match

Tabel 2.3: *Compare Match uitgang, Fast PWM-modus.*

COM01	COM00	Beschrijving
0	0	Normale poortwerking, OC0 is gedeactiveerd
0	1	Niet gebruikt
1	0	Zet OC0 op 0 op een compare match, zet op 1 op TCNT = 255
1	1	Zet OC0 op 1 op een compare match, zet op 0 op TCNT = 255

Tabel 2.3 geeft de functionaliteit van de COM0*x*-bits weer in Fast PWM-modus.

Tabel 2.4 geeft de functionaliteit van de COM0*x*-bits weer in Phase Correct PWM-modus.

Tabel 2.4: *Compare Match uitgang, Fast PWM-modus.*

COM01	COM00	Beschrijving
0	0	Normale poortwerking, OC0 is gedeactiveerd
0	1	Niet gebruikt
1	0	Zet OC0 op 0 bij Compare Match bij omhoog tellen, zet OC0 op 1 bij omlaag tellen
1	1	Zet OC0 op 1 bij Compare Match bij omhoog tellen, zet OC0 op 0 bij omlaag tellen

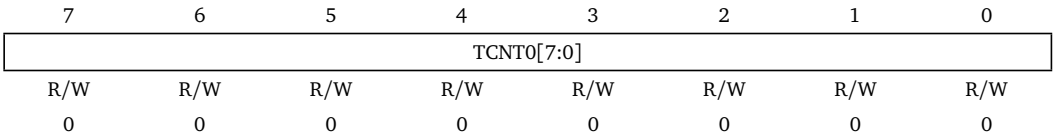
Bits 2,1,0 – CS0*x* klokselectiebits
In tabel 2.5 is te zien hoe een klok geselecteerd kan worden.

Het TCNT0-register is een 8-bits register en bevat de tellerwaarde van Timer/Counter 0, zie figuur 2.2. Dit register kan zowel gelezen als geschreven worden. Bij een schrijfactie telt Timer/Counter 0 verder vanaf de geschreven waarde.

Het OCR0-register bevat een 8-bits waarde en wordt continue vergeleken met de waarde

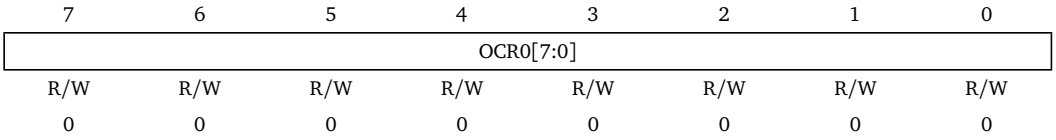
Tabel 2.5: *Klokselectie voor Timer/Counter 0.*

CS02	CS01	CS00	operatie
0	0	0	geen werking, klok is gestopt
0	0	1	$CLK_{IO}/1$, geen prescaler
0	1	0	$CLK_{IO}/8$, van prescaler
0	1	1	$CLK_{IO}/64$, van prescaler
1	0	0	$CLK_{IO}/256$, van prescaler
1	0	1	$CLK_{IO}/1024$, van prescaler
1	1	0	externe klokbron op de T0-pin, neergaande flank
1	1	1	externe klokbron op de T0-pin, opgaande flank



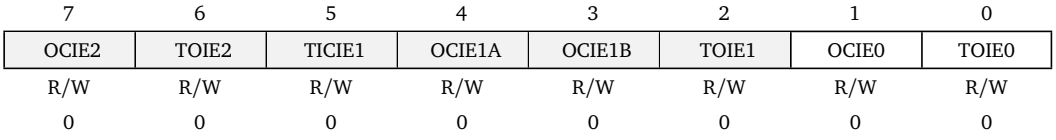
Figuur 2.2: *TCNT0 register*

in het TCNT0-register, zie figuur 2.3. Als de twee register gelijk zijn aan elkaar kan een interrupt gegenereerd worden of kan een golfvorm gegenereerd worden op de OC0-uitgang.



Figuur 2.3: *OCR0 register*

De indeling van het TIMSK-register is te zien in figuur 2.4. Voor Timer/Counter 0 zijn alleen de bits OCIE0 (bit 1) en TOIE0 (bit 0) van belang.



Figuur 2.4: *TIMSK register*

Bit 1 – OCIE0: Timer/Counter 0 Output Compare Match Interrupt Enable
Een logische 1 in deze bit zorgt ervoor dat de compare match interrupt geactiveerd wordt. Een interrupt wordt pas daadwerkelijk uitgevoerd als de I-bit in het statusregister ook een logische 1 is.

Bit 0 – TOIE0: Timer/Counter 0 Overflow Interrupt Enable
Een logische 1 in deze bit zorgt ervoot dat de overflow interrupt geactiveerd wordt. Een

interrupt wordt pas daadwerkelijk uitgevoerd als de I-bit in het statusregister ook een logische 1 is.

De indeling van het TIFR-register is te zien in figuur 2.5. Voor Timer/Counter 0 zijn alleen de bits OCF0 (bit 1) en TOV0 (bit 0) van belang.

7	6	5	4	3	2	1	0
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figuur 2.5: *TIFR register*

Bit 1 – OCF0: Output Compare Flag 0
Deze bit wordt op een logische 1 gezet als de waarde van het OCR0-register gelijk is aan de waarde in het TCNT0-register (compare match). Dit gebeurt op de volgende Timer/Counter 0 klokpuls. Deze bit wordt op een logische 0 gezet als de bijbehorende interrupt wordt uitgevoerd. Als alternatief voor het op 0 zetten kan een 1 worden geschreven naar deze bit.

Bit 0 – TOV0: Timer/Counter 0 Overflow Flag
Deze bit wordt op een logische 1 gezet als Timer/Counter 0 een overflow veroorzaakt. Dit gebeurt op de volgende Timer/Counter 0 klokpuls. Deze bit wordt op 0 gezet als de bijbehorende interrupt wordt uitgevoerd. Als alternatief voor het op 0 zetten kan een 1 worden geschreven naar deze bit. In fasecorrect PWM-modus wordt deze bit op 1 gezet als Timer/Counter 0 van telrichting verandert op 0.

In figuur 2.6 is het SFIOR-register te zien.

7	6	5	4	3	2	1	0
ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PRS10
R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figuur 2.6: *SFIOR register*

Bit 0 – PRS10: Prescaler reset Timer/Counter 1 en Timer/Counter 0
Wanneer deze bit met een 1 geschreven wordt, zal de prescaler van Timer/Counter 1 en Timer/Counter 0 worden gereset. Nadat de operatie is voltooid, wordt deze bit door de hardware op 0 gezet. Merk op dat de prescaler door zowel Timer/Counter 1 als Timer/Counter 0 gebruikt wordt. Het beïnvloedt de werking van beide Timer/Counters. Deze bit leest altijd als een 0.