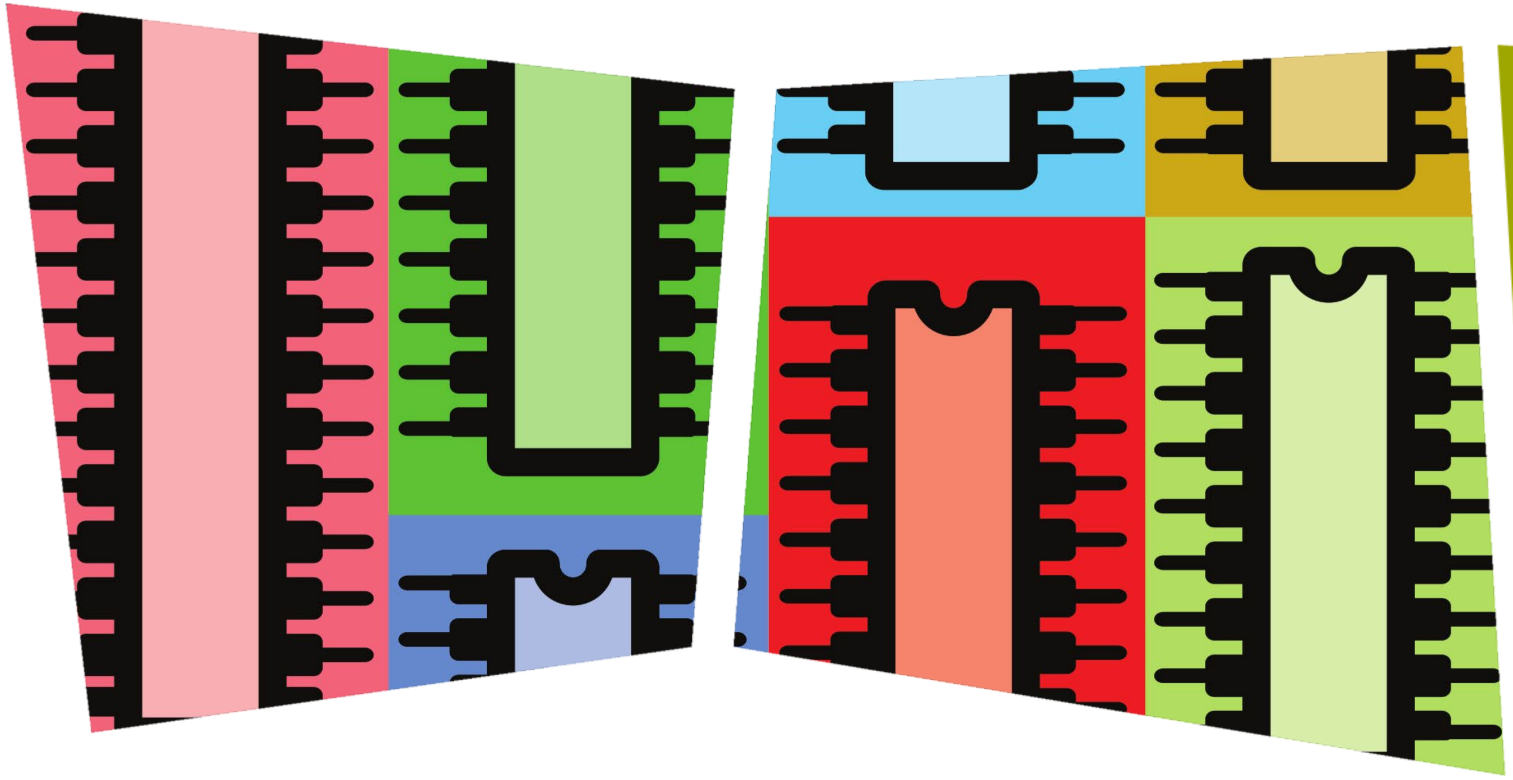


2021

G.M. Tuk

X. van Rijnsoever

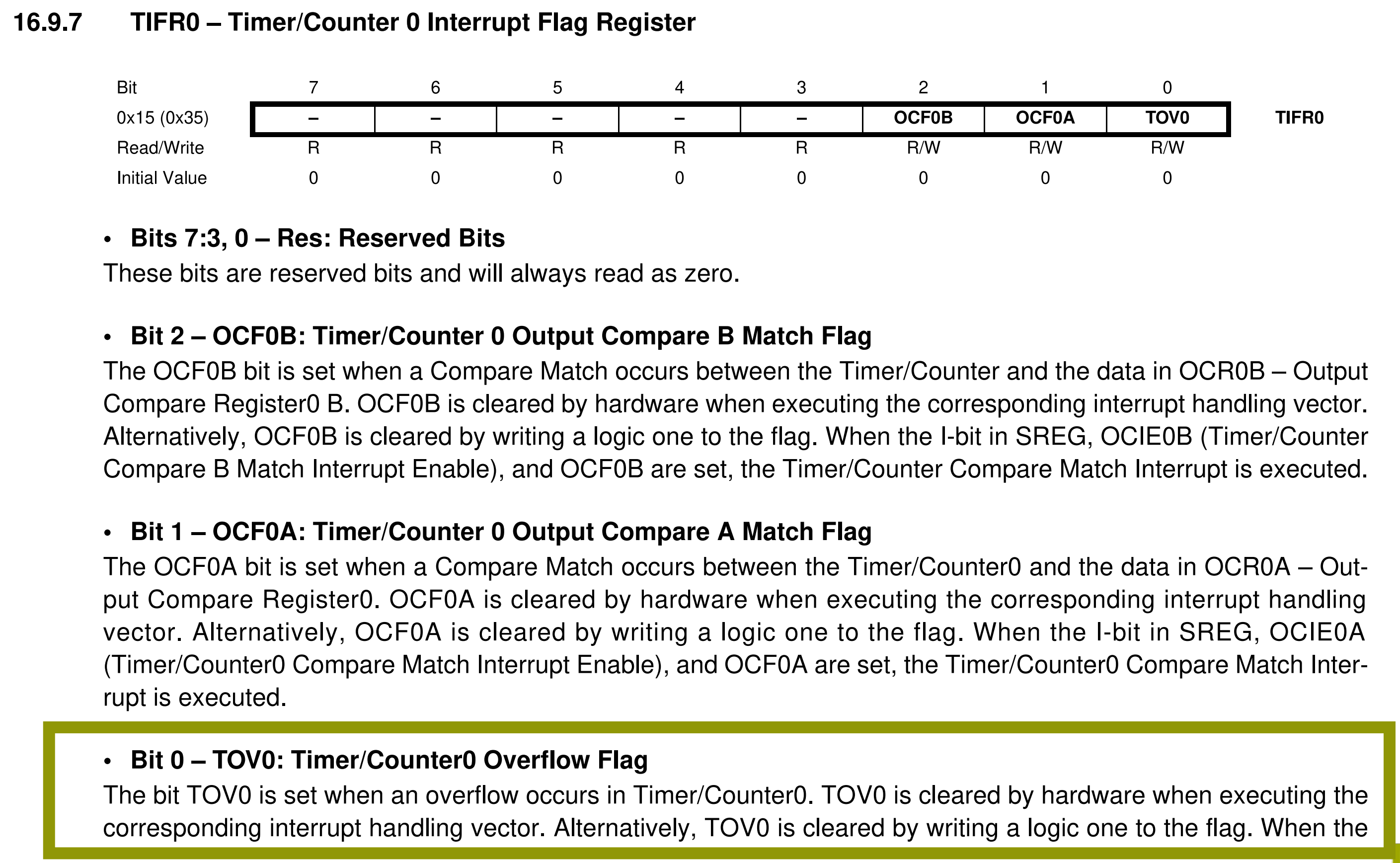
INTERRUPTS

Microcontroller Programmeren 2 – Week 3

Wat gaan we doen vandaag?

- Timer flag wissen
- Interrupts
  - Timer (internal)
  - Pinnen (external)
  - ISR()

Timer flags wissen



16.9.7 TIFR0 – Timer/Counter 0 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:3, 0 – Res: Reserved Bits**

These bits are reserved bits and will always read as zero.

- **Bit 2 – OCF0B: Timer/Counter 0 Output Compare B Match Flag**

The OCF0B bit is set when a Compare Match occurs between the Timer/Counter and the data in OCR0B – Output Compare Register0 B. OCF0B is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0B is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE0B (Timer/Counter Compare B Match Interrupt Enable), and OCF0B are set, the Timer/Counter Compare Match Interrupt is executed.

- **Bit 1 – OCF0A: Timer/Counter 0 Output Compare A Match Flag**

The OCF0A bit is set when a Compare Match occurs between the Timer/Counter0 and the data in OCR0A – Output Compare Register0. OCF0A is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0A is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE0A (Timer/Counter0 Compare Match Interrupt Enable), and OCF0A are set, the Timer/Counter0 Compare Match Interrupt is executed.

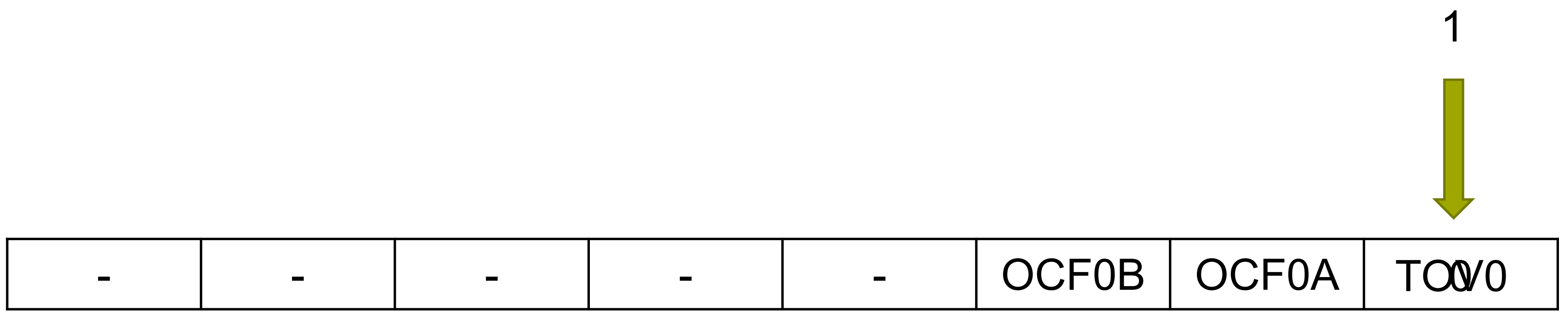
- **Bit 0 – TOV0: Timer/Counter0 Overflow Flag**

The bit TOV0 is set when an overflow occurs in Timer/Counter0. TOV0 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV0 is cleared by writing a logic one to the flag. When the

Timer flags wissen

- **Bit 0 – TOV0: Timer/Counter0 Overflow Flag**

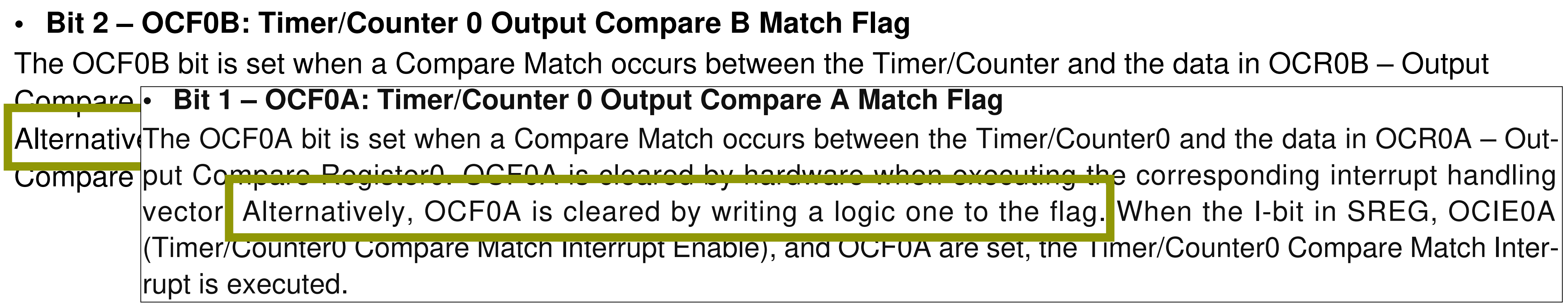
The bit TOV0 is set when an overflow occurs in Timer/Counter0. TOV0 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV0 is cleared by writing a logic one to the flag. When the SREG I-bit, TOIE0 (Timer/Counter0 Overflow Interrupt Enable), and TOV0 are set, the Timer/Counter0 Overflow interrupt is executed.



DE HAAGSE HOGESCHOOL

Timer flags wissen

~~TIFR |= (1 << TOV0);~~



• **Bit 2 – OCF0B: Timer/Counter 0 Output Compare B Match Flag**

The OCF0B bit is set when a Compare Match occurs between the Timer/Counter and the data in OCR0B – Output Compare

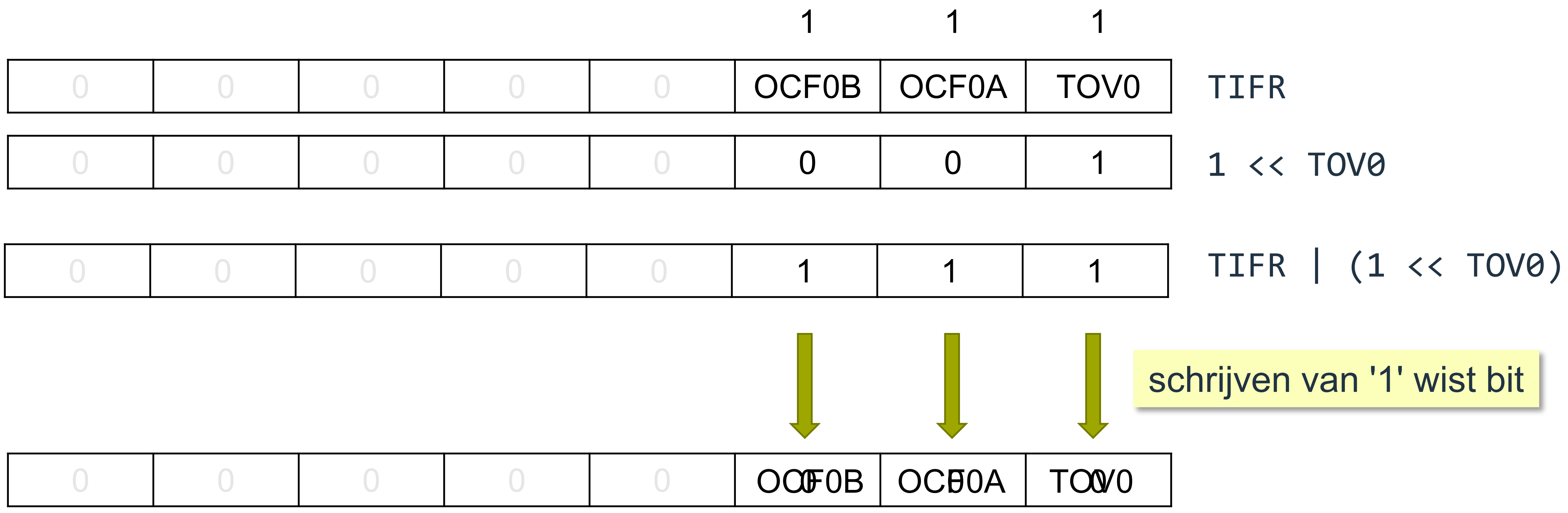
Alternativ

Compare

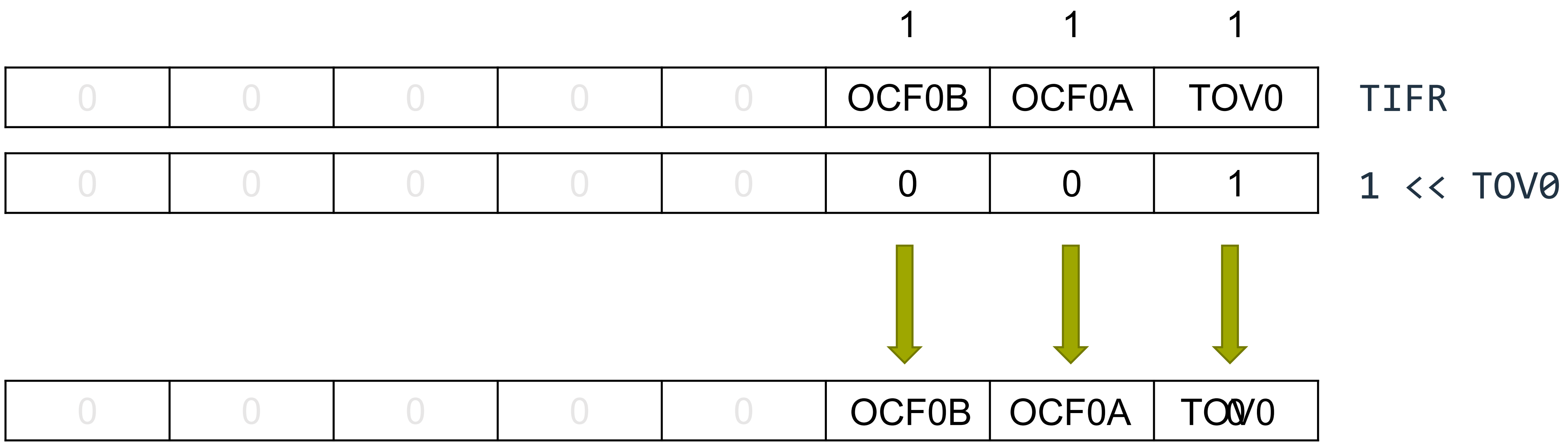
• **Bit 1 – OCF0A: Timer/Counter 0 Output Compare A Match Flag**

The OCF0A bit is set when a Compare Match occurs between the Timer/Counter0 and the data in OCR0A – Output Compare Register0. OCF0A is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0A is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE0A (Timer/Counter0 Compare Match Interrupt Enable), and OCF0A are set, the Timer/Counter0 Compare Match Interrupt is executed.

```
~~TIFR |= (1 << TOV0);~~
~~TIFR = TIFR | (1 << TOV0);~~
```



```
~~TIFR |= (1 << TOV0);~~
~~TIFR = TIFR | (1 << TOV0);~~
```



```
TIFR = (1 << TOV0);
```

PART II

Intermediate AVR

Hoofdstuk 8: Interrupts

Hoofdstuk 9: Timers en Counters

Hoofdstuk 10: PWM

Hoofdstuk 11: Servomotoren

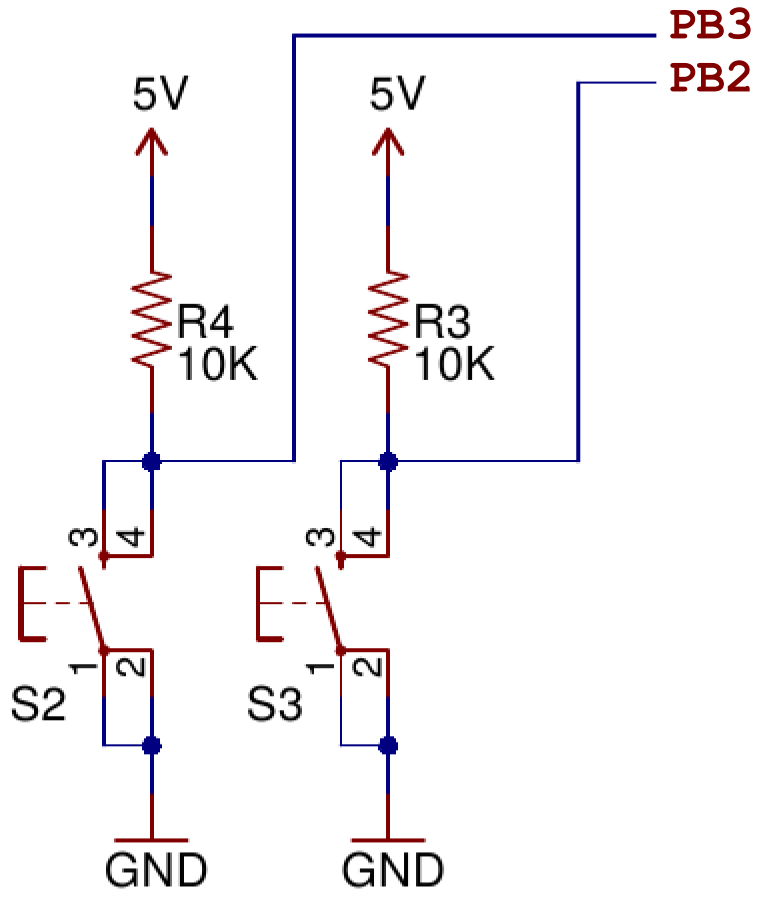
Hoofdstuk 12: AD-conversie

DE HAAGSE HOGESCHOOL

Code is sequentieel…

```
int som = 0;
for (int i = 7; i < 12; i++)
{
    if (i != 10)
    {
        som += i;
    }
}
printf("Som = %d\n", som);
```

```
// switch 2 pin: input
DDRB = DDRB & ~(1<<PB3);
// switch 3 pin: input
DDRB = DDRB & ~(1<<PB2);

while (1)
{
    // switch S2 active?
    if ((PINB & (1<<PB3)) == 0)
    {
        // do something...
    }

    // switch S3 active?
    if ((PINB & (1<<PB2)) == 0)
    {
        // do something...
    }
}
```



Polling

In blok 1 hebben we gebeurtenissen 'van buitenaf' afgevangen met **polling**.

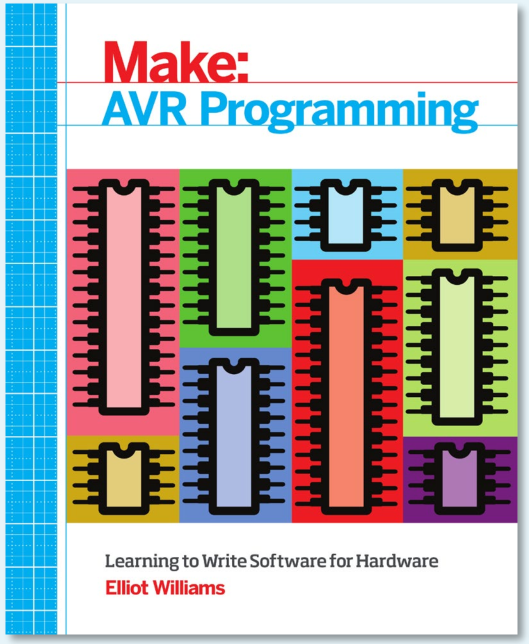
- voordeel: eenvoudig te programmeren
- nadeel: minder snelle respons
- nadeel: minder goed te timen

In 'gewone' software zorgt het Operating System ervoor dat de timing goed gaat. Bijvoorbeeld mouse clicks in Windows.

```
int main(void)
{
    while(1)
    {
        if (checkLightSensor())
        {
            set_bit(LED_PORT, LED);
        }
        switchesState = checkSwitches();
        if (switchesState == DANGER_VALUE)
        {
            turnOffKillerLaser();
        }
        doWhateverElse();
    }
}
```

Hoe lang kan dit wachten?

Hoe lang gaat dit duren?



pagina 154

Interrupts

Waarom zou je interrupts willen gebruiken?

- betere timing
- sneller respons van het systeem
- reageren op 'change' (flankdetectie)
- sluit meer aan op wat er eigenlijk gebeurt

Twee soorten interrupts

- **internally triggered interrupts**
  - AD-converter
  - USART
  - timer / counter
- **externally triggered interrupts**
  - spanningsverandering op inputpinnen

Timer interrupt

- Ook de timer kan een interrupt genereren
- Drie voorwaarden:
  - Timer moet een flag hebben gezet (TOV, OC, …)
  - De timer moet een interrupt genereren op het event: TMSK
  - Globale interrupts moeten actief zijn: `sei()`

ISR – Interrupt Service Routine

- In reactie op een interrupt, wordt een speciaal stuk code uitgevoerd: de ISR (interrupt service routine)

```
      ⋮
sub   r26, r26
sub   r27, r27
ldi   r21, 0x11
rjmp        .+14
adc   r26, r26
adc   r27, r27
cp    r26, r22
      ⋮
```

ISR - Naamgeving

- Namen van de ISR's liggen vast en worden aangeduid met de term "vector"
- Zoek lijst op op internet

```
INT0_vect
INT1_vect
PCINT0_vect
PCINT1_vect
PCINT2_vect
WDT_vect
...
```

Verschilt per type AVR!

ISR - Prioriteiten

- Als er twee interrupts tegelijkertijd arriveren, worden ze op volgorde van prioriteit
- Wordt IRQ genoemd: Interrupt Request Number
- Laagste nummer heeft hoogste prioriteit
- Zoek op op internet

ISR()

- geen echte functie: retourneert niets
- **ook geen `void`**
- heeft alleen interrupt vector als parameter
- je kunt dus niet zelf argumenten meegeven...
- ... en moet dus met globale variabelen werken

Later dit blok krijg je nog uitleg over hoe je dit beter kan aanpakken

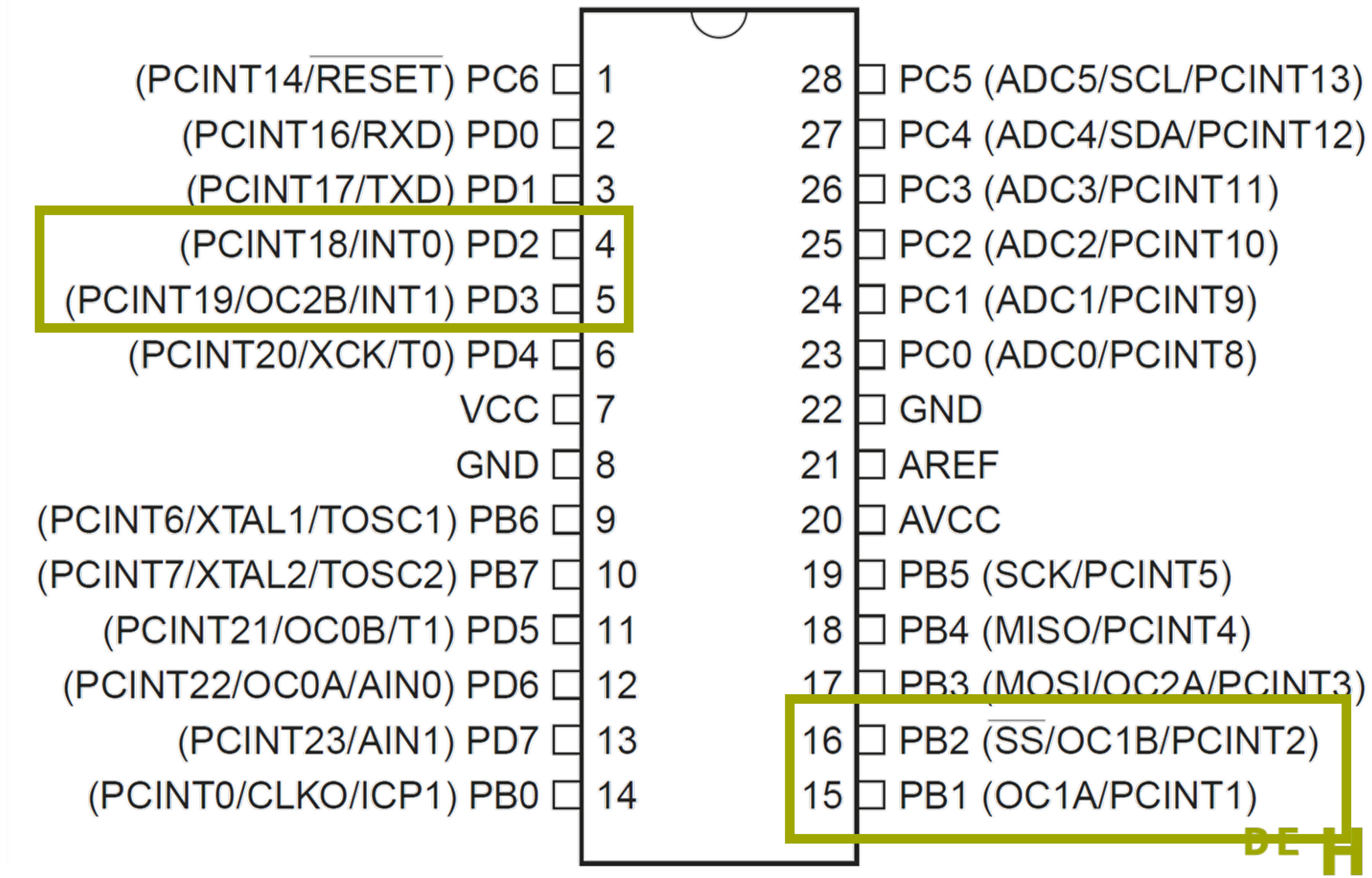
DE HAAGSE HOGESCHOOL

Twee 'varianten' externe interrupt

- **INT0 en INT1 (en nog twee?)**
  - meer mogelijkheden: opgaande flank, neergaande flank, verandering, continu voor laag
  - complexere hardware
  - hogere prioriteit
- **PCINT**
  - mogelijk op alle inputpinnen
  - alleen verandering (geen richting)

DE HAAGSE HOGESCHOOL

Arduino UNO

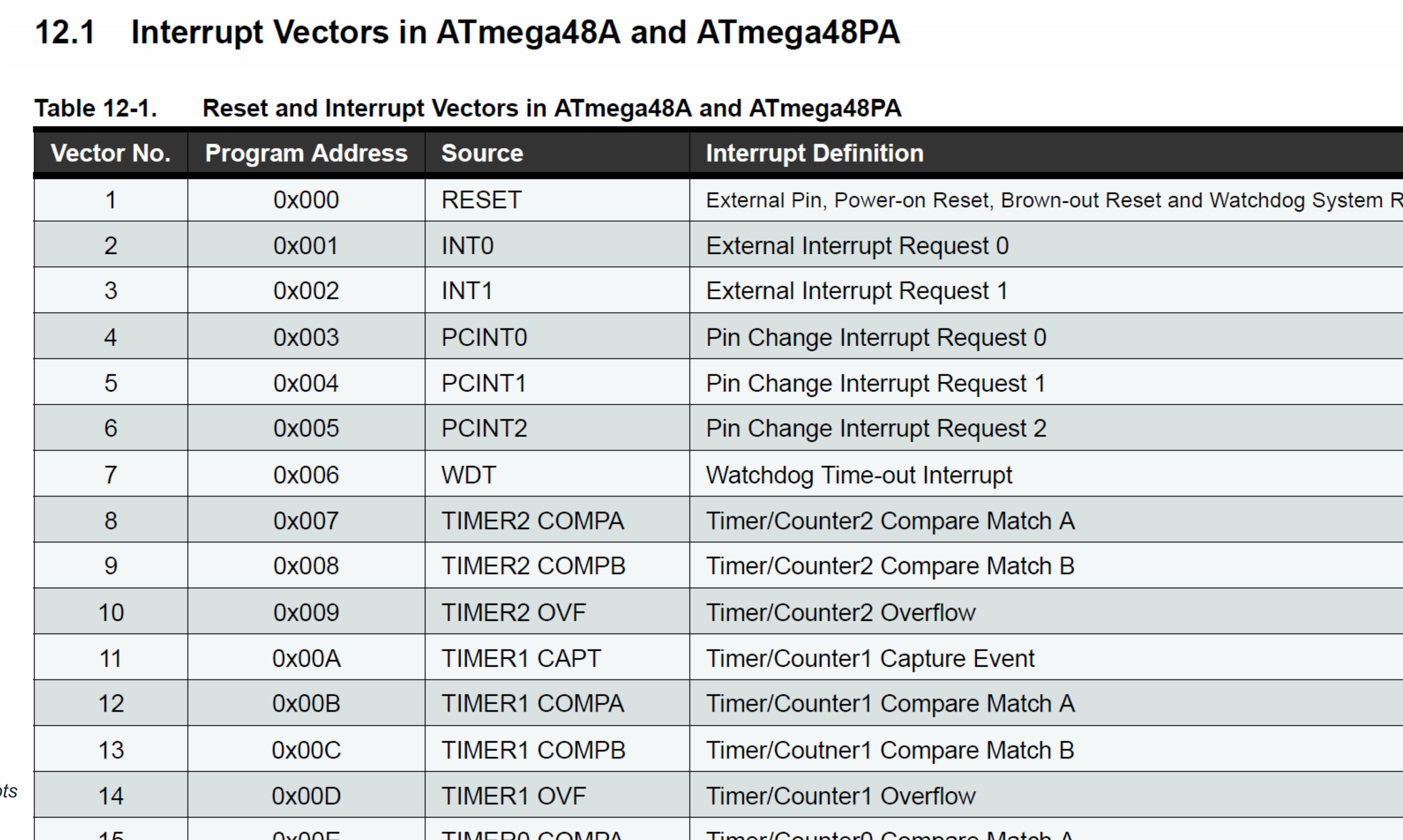


DE HAAGSE HOGESCHOOL

Pin Change Interrupt (PCINT)

- De interrupt geeft aan dat er iets veranderd is, maar je weet niet wat...
- ... en ook niet op welke pin!
- De interrupt signaleert een verandering op een bank, bijvoorbeeld B

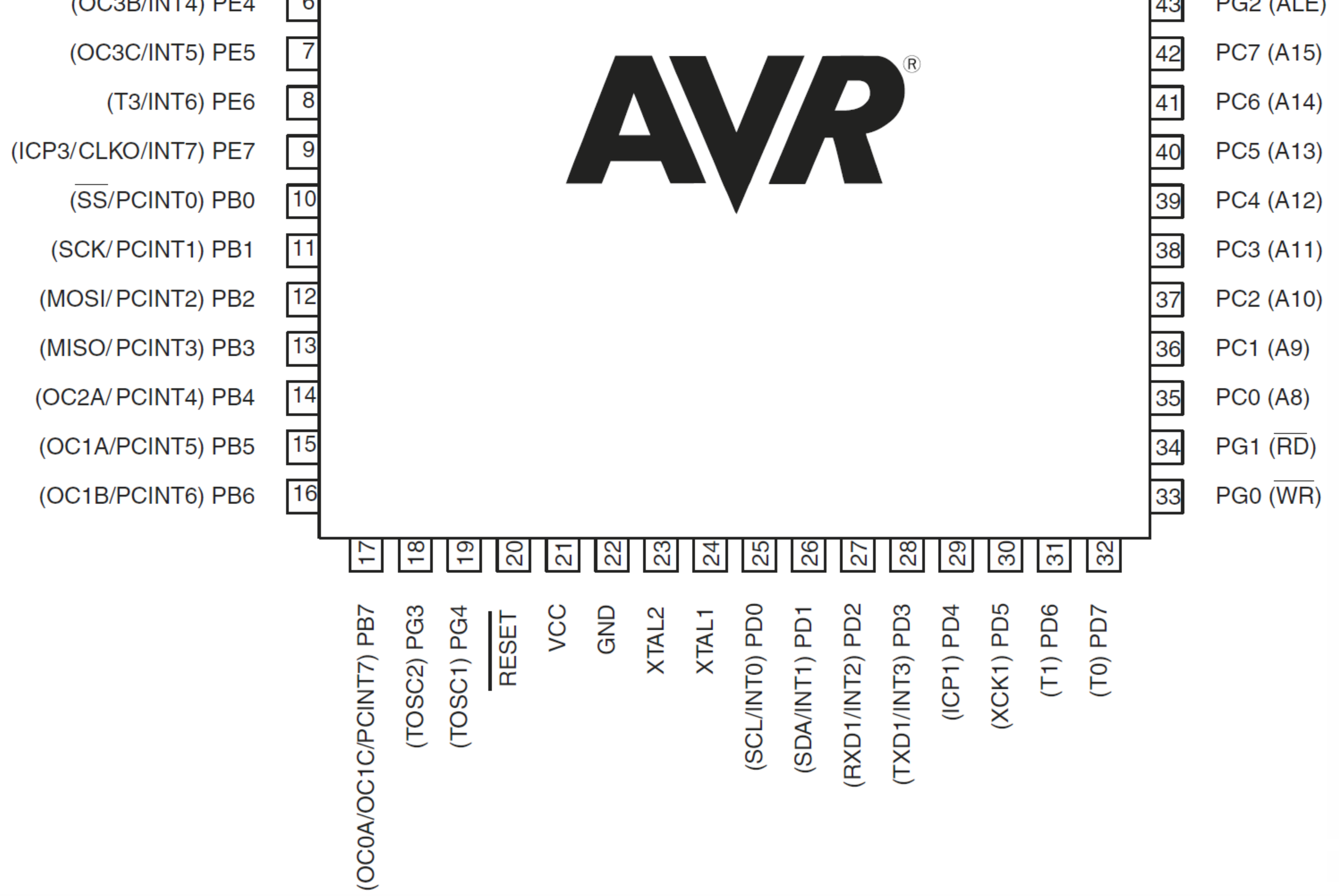
DE HAAGSE HOGESCHOOL



12.1 Interrupt Vectors in ATmega48A and ATmega48PA

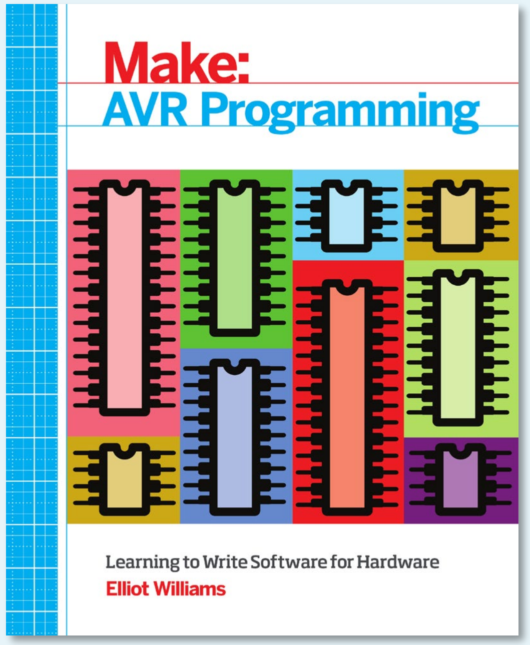
Table 12-1. Reset and Interrupt Vectors in ATmega48A and ATmega48PA

Vector No.	Program Address	Source	Interrupt Definition
1	0x000	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System R
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	PCINT0	Pin Change Interrupt Request 0
5	0x004	PCINT1	Pin Change Interrupt Request 1
6	0x005	PCINT2	Pin Change Interrupt Request 2
7	0x006	WDT	Watchdog Time-out Interrupt
8	0x007	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x008	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x009	TIMER2 OVF	Timer/Counter2 Overflow
11	0x00A	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x00B	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x00C	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x00D	TIMER1 OVF	Timer/Counter1 Overflow
15	0x00E	TIMER0 COMPA	Timer/Counter0 Compare Match A



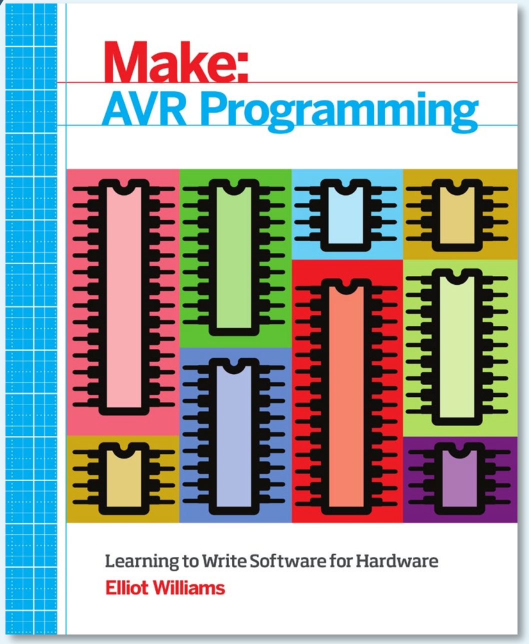
DE HAAGSE HOGESCHOOL

```
/*
Demo of using interrupts for doing what
they do best -- two things at once.

Flashes LED0 at a fixed rate, interrupting
whenever button is pressed.
*/

// ------- Preamble -------- //
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "pinDefines.h"
```



pagina 157

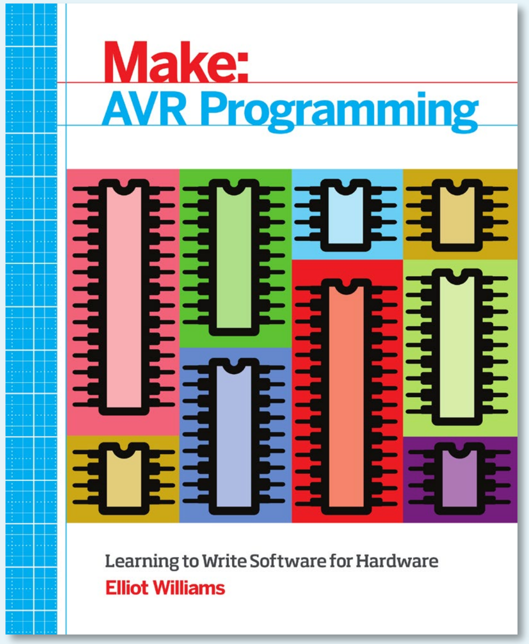
```
/* Run every time there is a change on button */
ISR(INT0_vect)
{
    if (bit_is_clear(BUTTON_PIN, BUTTON))
    {
        LED_PORT |= (1 << LED1);
    }
    else
    {
        LED_PORT &= ~(1 << LED1);
    }
}
```



pagina 157

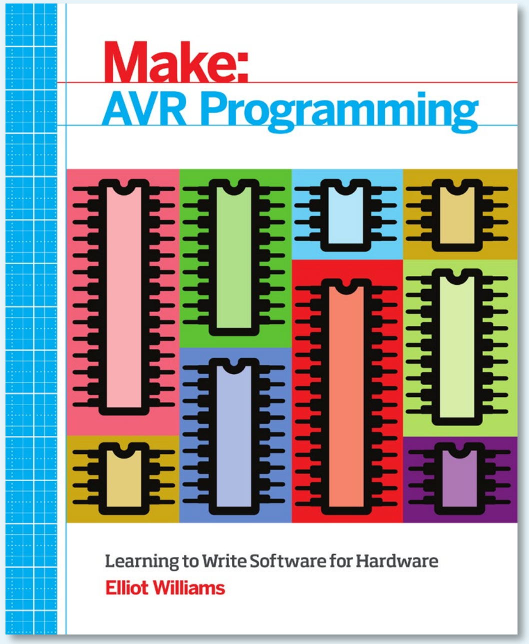
DE HAAGSE HOGESCHOOL

```
void initInterrupt0(void)
{
    EIMSK |= (1 << INT0);
    EICRA |= (1 << ISC00);
    /* set (global) interrupt enable bit */
    sei();
}
```



pagina 157

```
int main(void)
{
    // -------- Inits --------- //
    LED_DDR = 0xff; /* all LEDs active */
    BUTTON_PORT |= (1 << BUTTON); /* pullup */
    initInterrupt0();
    // ------ Event loop ------ //
    while (1)
    {
        _delay_ms(200);
        LED_PORT ^= (1 << LED0);
    } /* End event loop */
}
```



pagina 157

DE HAAGSE HOGESCHOOL

15.2.3 EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	EIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – INT7:0: External Interrupt Request 7 - 0 Enable**

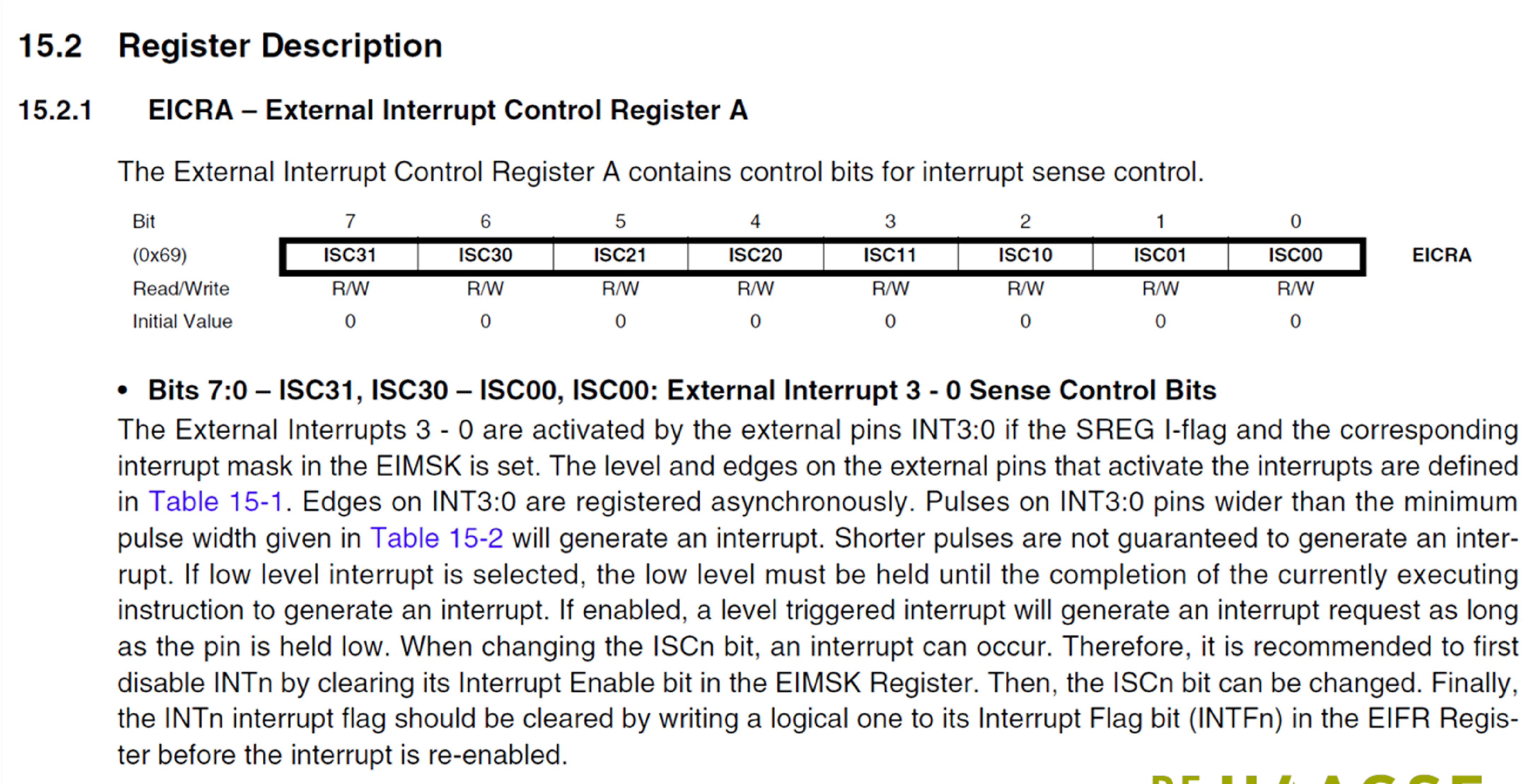
When an INT7:0 bit is written to one and the I-bit in the Status Register (SREG) is set (one), the corresponding external pin interrupt is enabled. The Interrupt Sense Control bits in the External Interrupt Control Registers – EICRA and EICRB – defines whether the external interrupt is activated on rising or falling edge or level sensed. Activity on any of these pins will trigger an interrupt request even if the pin is enabled as an output. This provides a way of generating a software interrupt.

15.2.4 EIFR – External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	IINTF0	EIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – INTF7:0: External Interrupt Flags 7 - 0**

When an edge or logic change on the INT7:0 pin triggers an interrupt request, INTF7:0 becomes



15.2 Register Description

15.2.1 EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – ISC31, ISC30 – ISC00, ISC00: External Interrupt 3 - 0 Sense Control Bits**

The External Interrupts 3 - 0 are activated by the external pins INT3:0 if the SREG I-flag and the corresponding interrupt mask in the EIMSK is set. The level and edges on the external pins that activate the interrupts are defined in Table 15-1. Edges on INT3:0 are registered asynchronously. Pulses on INT3:0 pins wider than the minimum pulse width given in Table 15-2 will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt. If enabled, a level triggered interrupt will generate an interrupt request as long as the pin is held low. When changing the ISCn bit, an interrupt can occur. Therefore, it is recommended to first disable INTn by clearing its Interrupt Enable bit in the EIMSK Register. Then, the ISCn bit can be changed. Finally, the INTn interrupt flag should be cleared by writing a logical one to its Interrupt Flag bit (INTFn) in the EIFR Register before the interrupt is re-enabled.

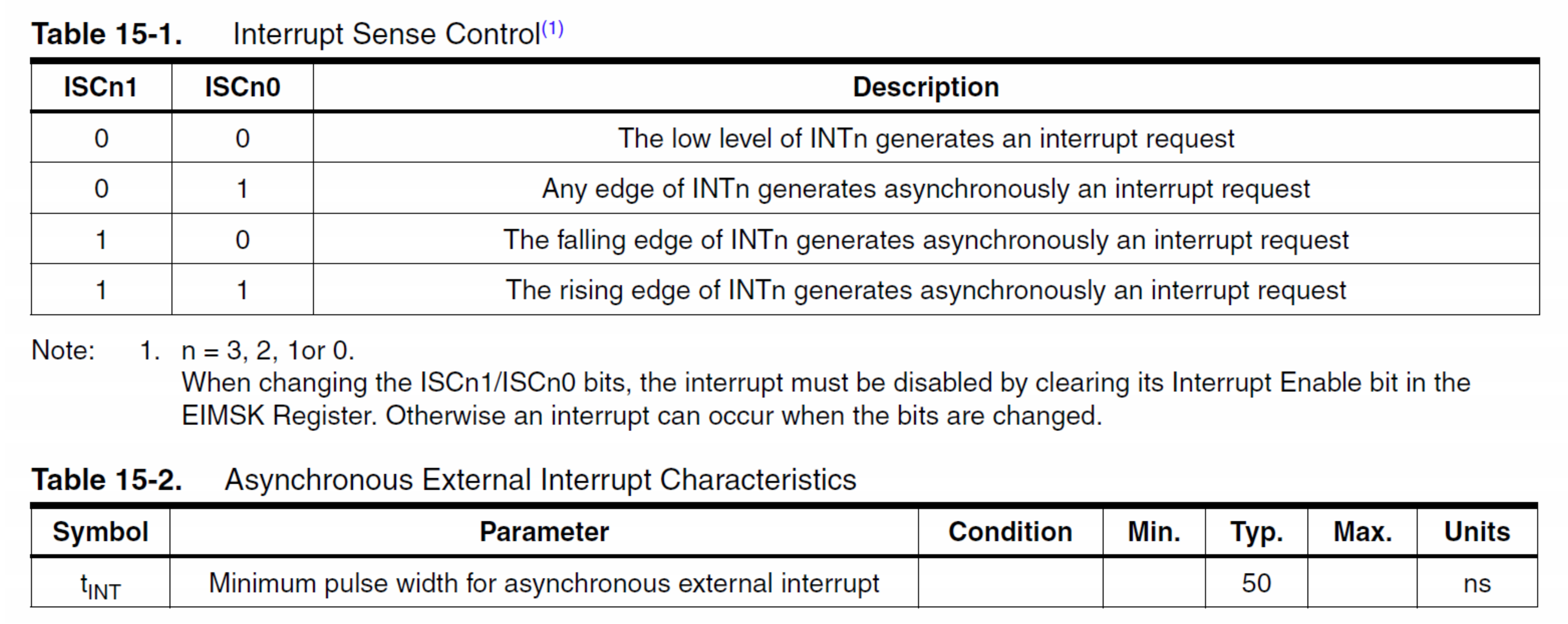


Table 15-1. Interrupt Sense Control[1]

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any edge of INTn generates asynchronously an interrupt request
1	0	The falling edge of INTn generates asynchronously an interrupt request
1	1	The rising edge of INTn generates asynchronously an interrupt request

Note: 1. n = 3, 2, 1or 0.
When changing the ISCn1/ISCn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.

Table 15-2. Asynchronous External Interrupt Characteristics

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
t_{INT}	Minimum pulse width for asynchronous external interrupt			50		ns

Volatile

In **computer programming**, particularly in the C, C++, C#, and Java programming languages, the **volatile** keyword indicates that a value may change between different accesses, even if it does not appear to be modified. This keyword prevents an optimizing compiler from optimizing away subsequent reads or writes and thus incorrectly reusing a stale value or omitting writes.

DE HAAGSE HOGESCHOOL

WE KUNNEN NU
EENVOUDIGE INTERRUPTS
AFHANDELEN