CPROUC/2021-2022

Ad van den Bergh

CPROUC

Bit- en Port-manipulatie

Week 14 (Les 25 en 26)

DE HAAGSE HOGESCHOOL

Operatoren (Soorten)

- Met operatoren kunnen gegevens bewerkt worden.
- 5 soorten operatoren:
  - Arithmetic Operators – rekenkundige bewerkingen
  - Comparison Operators - vergelijkende operatoren
  - Boolean Operators – Logische operatoren
  - Bitwise Operators
  - Compound Operators – samengestelde operatoren

Arithmetic Operators

- 5 soorten Operatoren:
  - **Arithmetic Operators – rekenkundige bewerkingen**
  - Comparison Operators - vergelijkende operatoren
  - Boolean Operators – Logische operatoren
  - Bitwise Operators
  - Compound Operators – samengestelde operatoren

- **Arithmetic Operators - Operatoren voor te rekenen**:
  - Toekennen: **A=B** Slaat de waarde van variabele B op in de variabele A.
  - Optellen: **A+B** Telt de waarde van A bijde waarde van B op.
  - Aftrekken: **A-B** Trekt de waarde van B af van de waarde van A.
  - Vermenigvuldigen: **A*B** Vermenigvuldigt de waarden van A met de waarde van B.
  - Delen **A/B** Deelt de teller A door de noemer B.
  - Modulo **A%B** Modulus operator en restwaarde na een deling van gehele getallen.

Comparison Operators

- 5 soorten Operatoren:
  - Arithmetic Operators – rekenkundige bewerkingen
  - **Comparison Operators - vergelijkende operatoren**
  - Boolean Operators – Logische operatoren
  - Bitwise Operators
  - Compound Operators – samengestelde operatoren
- **Comparison Operators – vergelijkende operatoren:**
  - Gelijk aan: (A==B) Als de 2 waarden gelijk zijn, dan is voorwaarde `true`.
  - Niet gelijk aan: (A!=B) Als de 2 waarden ongelijk zijn dan is voorwaarde `true`.
  - Kleiner dan: (A<B) Als A kleiner is dan B dan is voorwaarde `true`.
  - Groter dan: (A>B) Als A groter is dan B dan is voorwaarde `true`.
  - Kleiner of gelijk: (A<=B) Als A kleiner of gelijk is aan B dan is voorwaarde `true`.
  - Groter of gelijk: (A>=B) Als A groter of gelijk is aan B dan is voorwaarde `true`.

Boolean Operators

- 5 soorten Operatoren:
  - Arithmetic Operators – rekenkundige bewerkingen
  - Comparison Operators - vergelijkende operatoren
  - **Boolean Operators – Logische operatoren**
  - Bitwise Operators
  - Compound Operators – samengestelde operatoren
- **Boolean Operators – logische operatoren:**
  - Logical AND: (A && B)
    - Als beide waarden niet gelijk zijn aan 0, dan is voorwaarde `true`.
  - Logical OR: (A || B)
    - Als minimaal één van de waarden ongelijk is aan 0, dan is voorwaarde `true`.
  - Logical NOT: !(A && B)
    - Inverteren van de voorwaarde. Als voorwaarde is `false` dan wordt voorwaarde `true`.

Bitwise Operators

- 5 soorten Operatoren:
  - Arithmetic Operators – rekenkundige bewerkingen
  - Comparison Operators - vergelijkende operatoren
  - Boolean Operators – Logische operatoren
  - **Bitwise Operators**
  - Compound Operators – samengestelde operatoren
- **Bitwise Operators**
- Bitwise AND: C = A & B
- Kopieert een bit naar het resultaat als het bestaat in beide waarden.

A	B	A & B
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

	b7	b6	b5	b4	b3	b2	b1	b0	
A	0	0	1	1	1	1	0	0	**A**
B	0	0	0	0	1	1	0	1	**B**
A & B	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**A & B**

DE HAAGSE HOGESCHOOL

Bitwise Operators

- 5 soorten Operatoren:
  - Arithmetic Operators – rekenkundige bewerkingen
  - Comparison Operators - vergelijkende operatoren
  - Boolean Operators – Logische operatoren
  - **Bitwise Operators**
  - Compound Operators – samengestelde operatoren
- **Bitwise Operators**
- Bitwise OR: C = A | B
- Kopieert een bit naar het resultaat als het bestaat in beide waarden.

A	B	A | B
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

	b7	b6	b5	b4	b3	b2	b1	b0	
A	0	0	1	1	1	1	0	0	**A**
B	0	0	0	0	1	1	0	1	**B**
A | B	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**A | B**

DE HAAGSE HOGESCHOOL

Bitwise Operators

- 5 soorten Operatoren:
  - Arithmetic Operators – rekenkundige bewerkingen
  - Comparison Operators - vergelijkende operatoren
  - Boolean Operators – Logische operatoren
  - **Bitwise Operators**
  - Compound Operators – samengestelde operatoren
- **Bitwise Operators**
- Bitwise XOR: C = A ^ B
- Kopieert een bit naar het resultaat als het bestaat in beide waarden.

A	B	A ^ B
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

	b7	b6	b5	b4	b3	b2	b1	b0	
A	0	0	1	1	1	1	0	0	**A**
B	0	0	0	0	1	1	0	1	**B**
A ^ B	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**A ^ B**

DE HAAGSE HOGESCHOOL

Bitwise Operators

- 5 soorten Operatoren:
  - Arithmetic Operators – rekenkundige bewerkingen
  - Comparison Operators - vergelijkende operatoren
  - Boolean Operators – Logische operatoren
  - **Bitwise Operators**
  - Compound Operators – samengestelde operatoren
- **Bitwise Unary Operators**
- Bitwise NOT: ~ B
- Inverteert alle bits (flipping)

	b7	b6	b5	b4	b3	b2	b1	b0	
B	0	0	0	0	1	1	0	1	**B**
~B	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**~B**

DE HAAGSE HOGESCHOOL

Bitwise Operators

- 5 soorten Operatoren:
  - Arithmetic Operators – rekenkundige bewerkingen
  - Comparison Operators - vergelijkende operatoren
  - Boolean Operators – Logische operatoren
  - **Bitwise Operators**
  - Compound Operators – samengestelde operatoren
- **Bitwise Unary Operators**
- Bitwise SHIFT LEFT: B << 1
- Waarde van B wordt 1 bit naar links verschoven

	b7	b6	b5	b4	b3	b2	b1	b0	
B	0	0	0	0	1	1	0	1	**B**
B<<1	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**B<<1**

DE HAAGSE HOGESCHOOL

Bitwise Operators

- 5 soorten Operatoren:
  - Arithmetic Operators – rekenkundige bewerkingen
  - Comparison Operators - vergelijkende operatoren
  - Boolean Operators – Logische operatoren
  - **Bitwise Operators**
  - Compound Operators – samengestelde operatoren
- **Bitwise Unary Operators**
- Bitwise SHIFT LEFT: B << 2
- Waarde van B wordt 2 bits naar links verschoven

	b7	b6	b5	b4	b3	b2	b1	b0	
B	0	0	0	0	1	1	0	1	**B**
B<<2	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**B<<2**

DE HAAGSE HOGESCHOOL

Bitwise Operators

- 5 soorten Operatoren:
  - Arithmetic Operators – rekenkundige bewerkingen
  - Comparison Operators - vergelijkende operatoren
  - Boolean Operators – Logische operatoren
  - **Bitwise Operators**
  - Compound Operators – samengestelde operatoren
- **Bitwise Unary Operators**
- Bitwise SHIFT RIGHT: B >> 1
- Waarde van B wordt 1 bit naar rechts verschoven

	b7	b6	b5	b4	b3	b2	b1	b0	
B	0	0	0	0	1	1	0	1	**B**
B>>1	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**B>>1**

DE HAAGSE HOGESCHOOL

Bitwise Operators

- 5 soorten Operatoren:
  - Arithmetic Operators – rekenkundige bewerkingen
  - Comparison Operators - vergelijkende operatoren
  - Boolean Operators – Logische operatoren
  - **Bitwise Operators**
  - Compound Operators – samengestelde operatoren
- **Bitwise Unary Operators**
- Bitwise SHIFT RIGHT: B >> 2
- Waarde van B wordt 2 bits naar rechts verschoven

	b7	b6	b5	b4	b3	b2	b1	b0	
B	0	0	0	0	1	1	0	1	**B**
B>>2	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**B>>2**

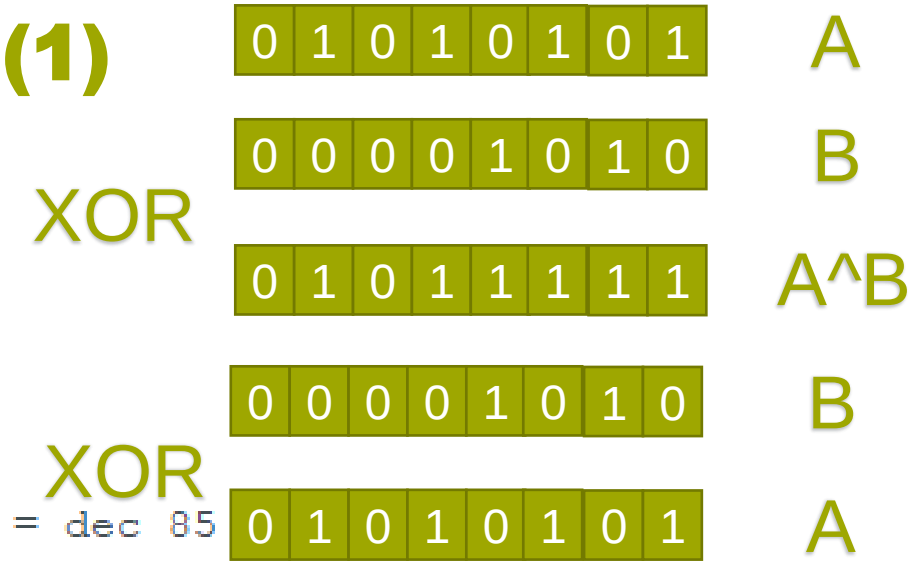
DE HAAGSE HOGESCHOOL

Compound Operators

- 5 soorten Operatoren:
  - Arithmetic Operators – rekenkundige bewerkingen
  - Comparison Operators - vergelijkende operatoren
  - Boolean Operators – Logische operatoren
  - Bitwise Operators
  - **Compound Operators – samengestelde operatoren**
- **Compound Operators – samengestelde operatoren:**
  - Verkorte schrijfwijze:
  - A++ ; A = A + 1;
  - A--; A = A – 1;
  - A += B; A = A + B;
  - A -= B; A = A – B;
  - A *= B; A = A * B;
  - A /= B; A = A / B;
  - A %= B; A = A % B;
  - A &= B; A = A & B; // Bitwise AND
  - A |= B; A = A | B; // Bitwise OR

Bitwise Operatoren toepassen(1)

- Andere naam: **Bitmanipulatie**
- Voorbeeld met bitwise XOR : encryptie




```
void loop() {
  //Voorbeeld encryptie eerst 0x55 laten zien
  // daarna XOR met 0x0A en nog een keer
  waarde = 0b01010101; // = bin 01010101 = 0x55 = dec 85
  Serial.print("waarde = "); Serial.println(waarde);
  stuurLedsAan(waarde);
  delay(1000);
  waarde = waarde ^ 0b00001010; // = bin 01011111 = dec 95
  Serial.print("waarde = "); Serial.println(waarde);
  stuurLedsAan(waarde);
  delay(1000);
  waarde = waarde ^ 0b00001010; // = bin 01010101 = dec 85
  Serial.print("waarde = "); Serial.println(waarde);
  stuurLedsAan(waarde);
  delay(1000);
}
```

Originele waarde terug te halen met sleutel B

DE HAAGSE HOGESCHOOL

Bitwise Operatoren toepassen(2)

- In stuurLedsAan() ook **Bitmanipulatie**

Bitwise AND

Shift Left

```
void stuurLedsAan(uint8_t getal) {
  int  j = 1;
  uint8_t hulp;
  for (uint8_t k = 0; k < aantalLEDs; k++) {
    hulp = getal & j;  // bitwise AND
    if (hulp) {
      digitalWrite(ledPins[k], AAN);  // zet de LED aan
    } else {
      digitalWrite(ledPins[k], UIT);  // zet de LED uit
    }
    j = j << 1;
  }
}
```

Demo

Wat is de functie van stuurLedsAan()?

Waarde van getal op 8 leds zichtbaar maken.

Bitwise Operatoren toepassen(3)

- In loop() kan nu looplicht (KnightRider) gemaakt worden:

```
void loop() {
  waarde = 0b00000001;
  for (i = 0; i < aantalLEDs; i++) {
    Serial.print("waarde = "); Serial.println(waarde);
    stuurLedsAan(waarde);
    wacht = analogRead(0);  // waarde van de potmeter (max 1023)
    delay (wacht);
    waarde = waarde << 1;
  }
  waarde = 0b01000000;
  for (i = 0; i < aantalLEDs-2; i++) {
    Serial.print("waarde = "); Serial.println(waarde);
    stuurLedsAan(waarde);
    wacht = analogRead(0);  // waarde van de potmeter (max 1023)
    delay (wacht);
    waarde = waarde >> 1;
  }
}
```

Demo

Portmanipulation

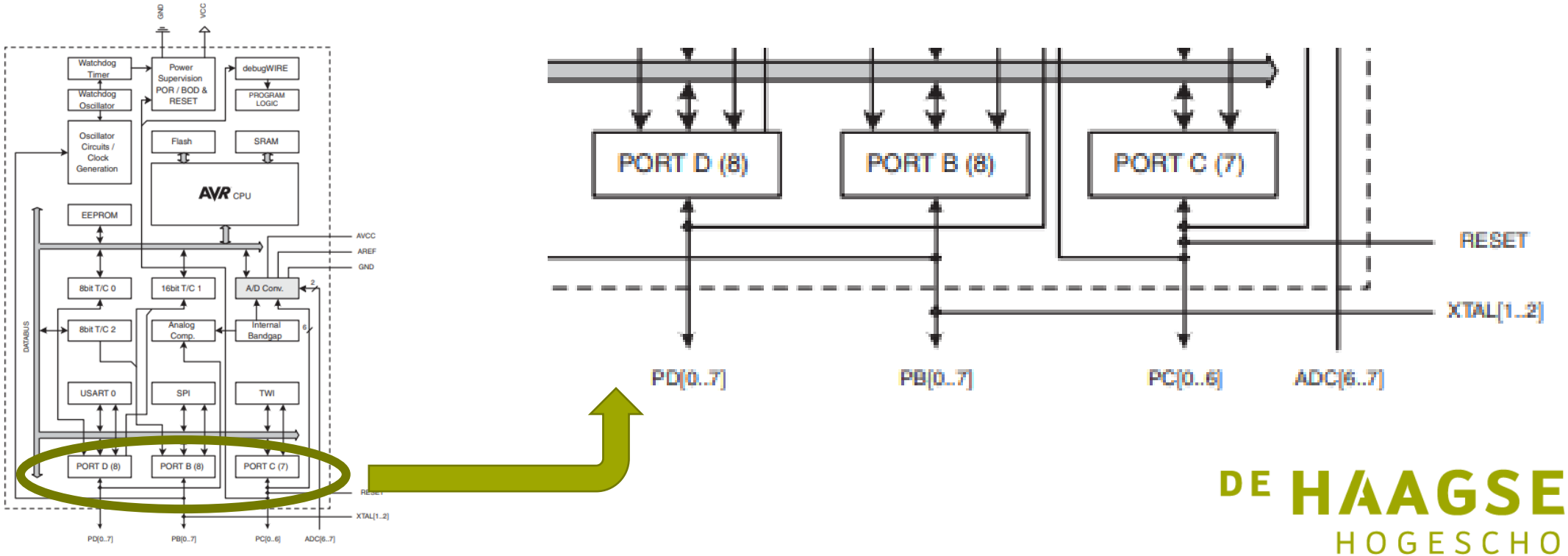
- Arduino heeft HAL-instructies gemaakt om eenvoudig 1 outputpin te kunnen besturen
- Bijv. `digitalWrite( 13, HIGH);`
- Waarde op uitgangen (leds) moet met een functie, zoals **`stuurLedsAan`**`(uint8_t getal);` uit de vorige les: encryptie

```
void stuurLedsAan(uint8_t getal) {
  int  j = 1;
  uint8_t hulp;
  for (uint8_t k = 0; k < aantalLEDs; k++) {
    hulp = getal & j;  // bitwise AND
    if (hulp) {
      digitalWrite(ledPins[k], AAN);  // zet de LED aan
    } else {
      digitalWrite(ledPins[k], UIT);  // zet de LED uit
    }
    j = j << 1;
  }
}
```

Dat kan SNELLER !

Poort manipulatie

- Bij poort manipulatie worden de 8 bits poorten direct geschreven (outputs) of ingelezen (inputs).
- Ideaal als bijv. tellerstanden zichtbaar gemaakt moeten worden.
- Hoe zat het ook alweer met die poorten van de ATmega328?



DE HAAGSE HOGESCHOOL

Poorten op de Arduino UNO

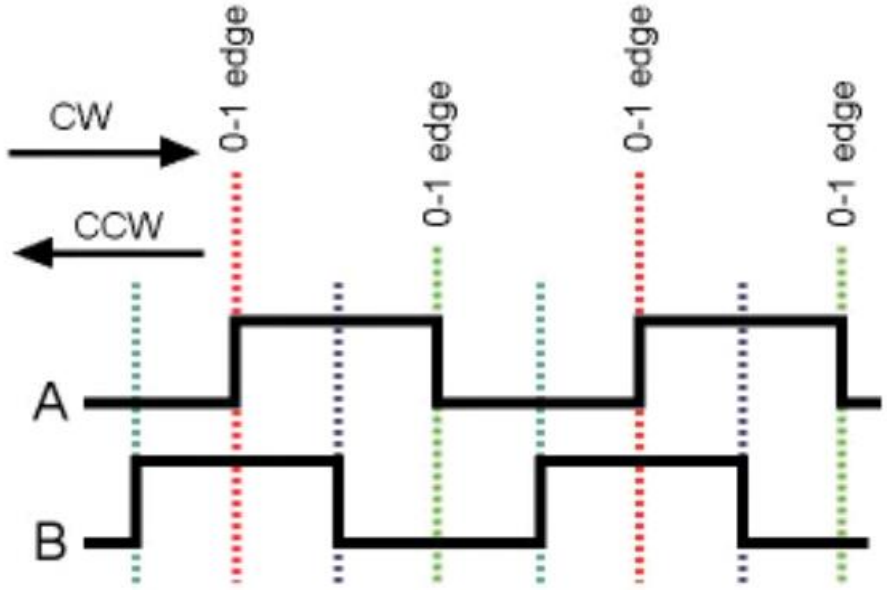
- 3 Poorten:
  1. Port B (digitale pin 8 t/m 13)
  2. Port C (analoge inputs 0 t/m 5)
  3. Port D (digitale pin 0 t/m 7)

Let op: pin 0 en 1 zijn gereserveerd voor seriële communicatie en kunnen bij gebruik problemen opleveren bij programmeren en debuggen.

- Iedere poort heeft 3 registers:
  1. een Data Direction Register, waarin wordt bepaald of het om een in- of uitgang gaat.
  2. Een PORT register, die beheert of de output HIGH of LOW is.
  3. Een PIN register, die de status van de INPUT pins weergeeft, die als input geset zijn met pinMode().
- De Arduino compiler kent de namen van de variabelen:
  - Data Direction Registers: DDRB, DDRC en DDRD (hoofdletters) maar geeft ze geen ander kleurtje!)
  - Port registers: PORTB, PORTC en PORTD (zowel te schrijven als te lezen)
  - PIN registers: PINB, PINC en PIND (alleen te lezen)

Rotary encoder

- In het volgende voorbeeld wordt een rotary encoder gebruikt om een tellerstand te genereren, waarna de tellerstand zichtbaar gemaakt wordt op 6 bits van PortB van pin 13 t/m pin 8



DE HAAGSE HOGESCHOOL

Voorbeeld: 6 bits teller met rotary encoder (setup)

```
uint8_t teller = 32;
unsigned long actueleTijd;
unsigned long cyclusTijd;
const int pin_A = 2;  // pin 2 de CLK aansluiting van de Rotary encoder
const int pin_B = 3;  // pin 3 De DT aansluiting van de Rotary encoder
const int SW = 4;     // pin 4 de switch van de rotary encoder
unsigned char encoder_A;
unsigned char encoder_B;
unsigned char encoder_A_vorige=0;

void setup()  {
  pinMode(pin_A, INPUT_PULLUP);
  pinMode(pin_B, INPUT_PULLUP);
  pinMode(SW, INPUT_PULLUP);
  Serial.begin(9600);
  DDRB =0xFF; // PortB als uitgangen pin 13 tm pin 8
  actueleTijd = millis();
  cyclusTijd = actueleTijd;
}
```

Voorbeeld: 6 bits teller met rotary encoder (loop)

```
void loop()  {
  // haal de verstreken tijd op sinds laatste sample
  actueleTijd = millis();
  if(actueleTijd >= (cyclusTijd + 5)){
    // 5ms sinds de laatste test van de encoder = 200Hz
    encoder_A = digitalRead(pin_A);     // Lees de waarde van de encoder pins
    encoder_B = digitalRead(pin_B);
    if((!encoder_A) && (encoder_A_vorige)){
      // A is van hoog naar laag gegaan
      if(encoder_B) {
        // B is hoog, dus de encoder is "met de klok mee" (naar rechts) gedraaid
        // verhoog de teler, tot maximaal 255 (8 bits)
        if(teller+1 <= 64) teller++;
      }
      else {
        // B is laag, dus de encoder is "tegen de klok in" (naar links) gedraaid
        // verlaag de teller, tot minimaal 0
        if(teller - 1 >= 0) teller--;
      }

    }
    encoder_A_vorige = encoder_A;      // Sla de waarde van A op voor de volgende keer
    if (!digitalRead(SW)) teller = 64; // als toets wordt gedrukt
    // Stuur de tellerstand naar de monitor
    Serial.println(teller);
    PORTB = ~teller; // inverteren omdat de led aan = 0

    cyclusTijd = actueleTijd;  // actualiseer de cyclusTijd
  }
  // Hier kunnen andere activiteiten worden uitgevoerd (wel binnen 5 ms blijven)

}
```

Voorbeeld: 6 bits teller met port inputs(loop)

Hieronder is het gedeelte te zien om ook de signalen A en B via poortmanipulatie binnen te halen:

```
void setup()  {
  pinMode(pin_A, INPUT_PULLUP);
  pinMode(pin_B, INPUT_PULLUP);
  pinMode(SW, INPUT_PULLUP);
  Serial.begin(9600);
  DDRB =0xFF; // PortB als uitgangen pin 13 tm pin 8
  DDRD = B11100011; // Pin_A op 2 en Pin_B op 3 en SW op 4
  actueleTijd = millis();
  cyclusTijd = actueleTijd;
}

void loop()  {
  // haal de verstreken tijd op sinds laatste sample
  actueleTijd = millis();
  if(actueleTijd >= (cyclusTijd + 5)){
    // 5ms sinds de laatste test van de encoder = 200Hz
    portWaarde = PIND & B00011100;
    encoder_A = portWaarde & B00000100;    // A op 2
    encoder_B = portWaarde & B00001000;;   // B op 3
    if((!encoder_A) && (encoder_A_vorige)){
      // A is van hoog naar laag gegaan
```


DE HAAGSE HOGESCHOOL

Nadelen gebruik poort manipulatie

- Naast het grote voordeel van snelheid zijn er ook nadelen bij het gebruik van poortmanipulatie:
- De code is veel moeilijker te debuggen én te onderhouden (fout zoeken)
- De code is minder “portable” (niet eenvoudig over te zetten naar andere systemen (hardware met andere I/O poorten).
  - `digitalRead()` en `digitalWrite()` werkt wel op alle Atmel processoren
- Grote kans op onverwachte fouten.
  - Voorbeeld: `DDRD = B11111110;` zorgt ervoor dat pin 0 een input is wat de ontvangpin RX is. Als je ergens in de sketch `DDRD = B11111111;` zet, wordt het opeens een output en je ontvangt helemaal niets meer.
- **Voordelen:** snelheid en alle outputs gelijk veranderd en minder geheugen nodig.

let’s change