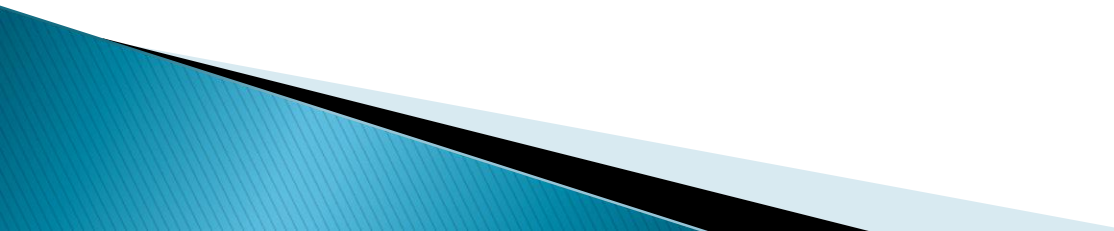


Arduino en I2C – Dag 4

Huiswerk

1. Test je eigen I2C slave met de toolkit.

Voor de C liefhebbers:

2. Print tijd als hh:mm:ss (Toegift b)
 3. Programma: print temperatuur in hoge resolutie (0.25 graden), (Oefening d)
- 

Lees tijd – Resultaat (b)

- ▶ print tijd als 'hh:mm:ss'

```
case 'b' :
{
    // Defening b
    I2cTxBuffer[0] = 0;           // register nummer
    r = I2cSendReceive(DS3231_I2C_ADDRESS, 1, 3, I2cTxBuffer, I2cRxBuffer);
    if (r == true) { // succes ?
        printf("Tijd: %02x:%02x:%02x\n", I2cRxBuffer[2], I2cRxBuffer[1], I2cRxBuffer[0]);
    }
    else {
        printf("I2C fout\n");
    }
    break;
}
```

Temperatuur in hoge resolutie

```
case 'd' :
{
    // Defining d
    I2cTxBuffer[0] = 0x11;    // register nummer
    r = I2cSendReceive(DS3231_I2C_ADDRESS, 1, 2, I2cTxBuffer, I2cRxBuffer);
    if (r == true) { // succes ?
        float temperatuur = I2cRxBuffer[0] + I2cRxBuffer[1] / 256.0;
        Serial.print("De temperatuur is ");
        Serial.print(temperatuur);
        Serial.print(" graden\n");
    }
    else {
        printf("I2C fout\n");
    }
    break;
}
```

De wereld vanuit de slave gezien

- ▶ Adressering wordt afgehandeld in hardware
- ▶ Slave krijgt vraag om data te leveren
- ▶ Slave krijgt data aangeboden
- ▶ Interrupt gedreven

```
void I2cSlaveRegistersInit(int Slave)
{
    Wire.begin(Slave);
    Wire.onRequest(requestEvent);
    Wire.onReceive(receiveEvent);
}
```

Slave registers setup

- ▶ I2cSlaveRegisters.ino
- ▶ Laag 'bovenop' Arduino Wire class
- ▶ Array van 'registers'.

```
#define SLAVE_ADDRESS    0x44  // slave address, any number from 0x01 to 0x7F
```

```
#define REG_MAP_SIZE    (6)
```

```
volatile byte I2cRegister[REG_MAP_SIZE];
```

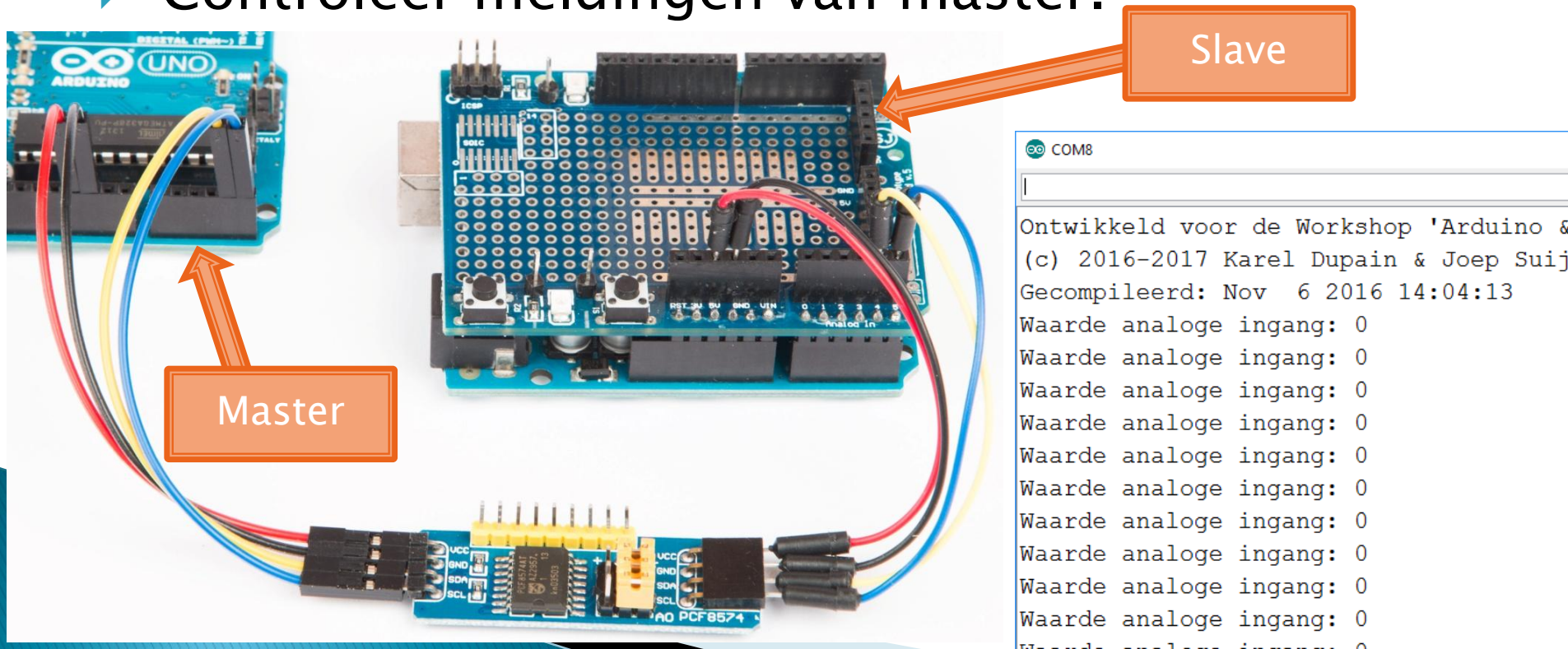
```
//-----
```

```
// i2c interface init
```

```
I2cSlaveRegistersInit(SLAVE_ADDRESS);
```

Opbouw hardware

- ▶ Eerst slave programmeren met i2c_slave_ws0
- ▶ USB kabel losmaken, bordjes verbinden
- ▶ Master programmeren met i2c_master_ws4
- ▶ Controleer meldingen van master.

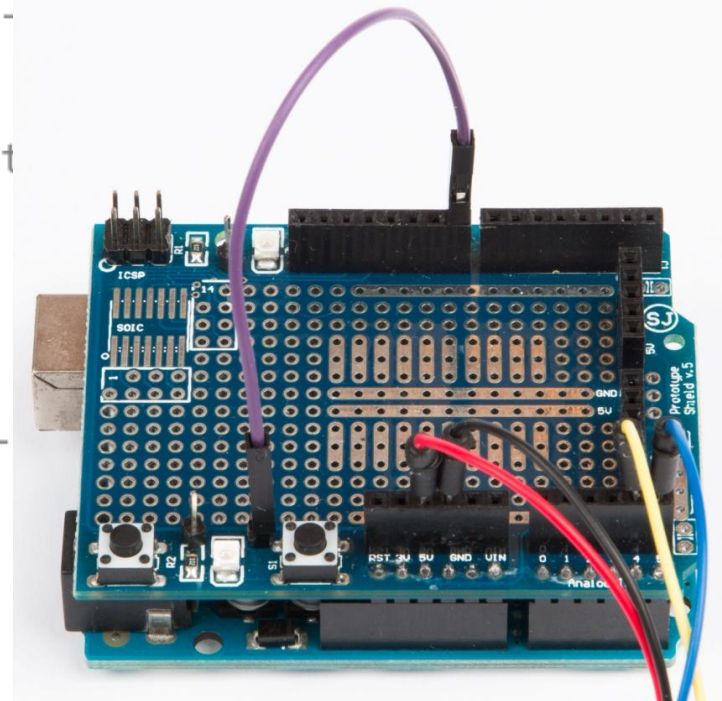


Digitale ingang – oefening (a)

- ▶ USB Kabel aan de slave
- ▶ i2c_slave_ws0
- ▶ Paarse draadje: knopje aansluiten op D8
- ▶ code voor setup & oefening a toevoegen.

```
//-----  
// i2c interface init  
I2cSlaveRegistersInit(SLAVE_ADDRESS);  
pinMode(8, INPUT_PULLUP); // LED_M input
```

```
// oefening a  
I2cRegister[R_LED_M] = digitalRead(8); // L
```



Digitale ingang – Resultaat

- ▶ #define in slave file

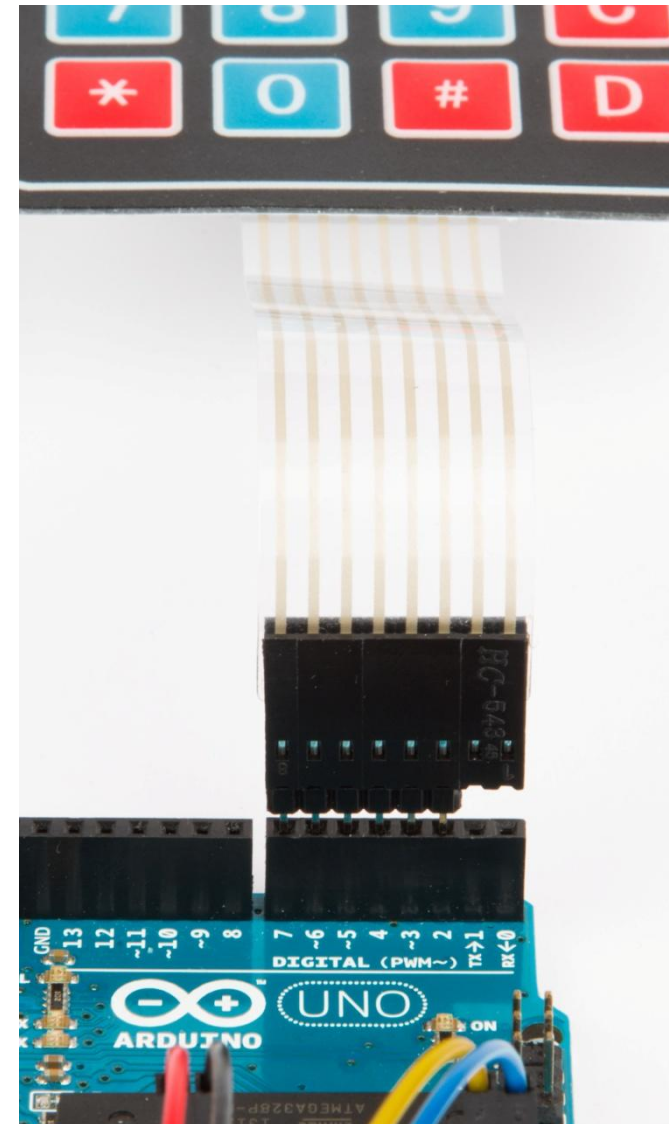
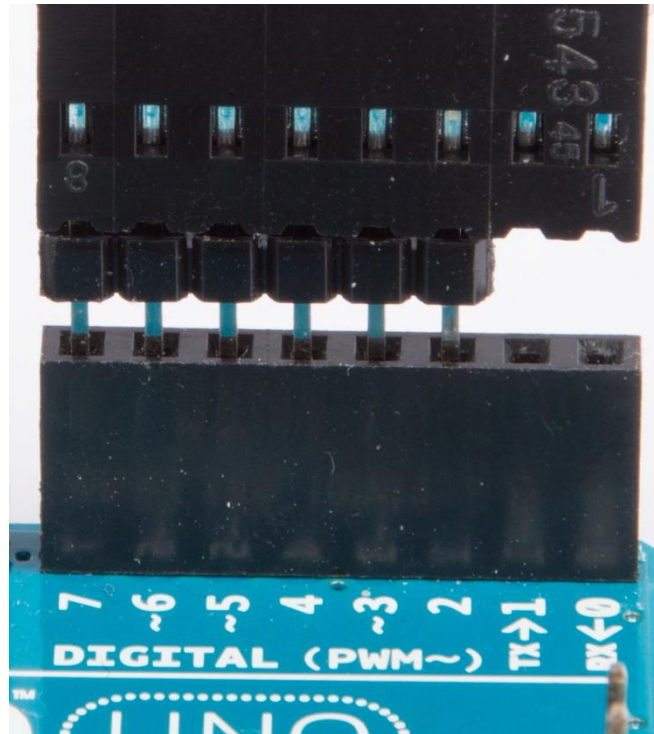
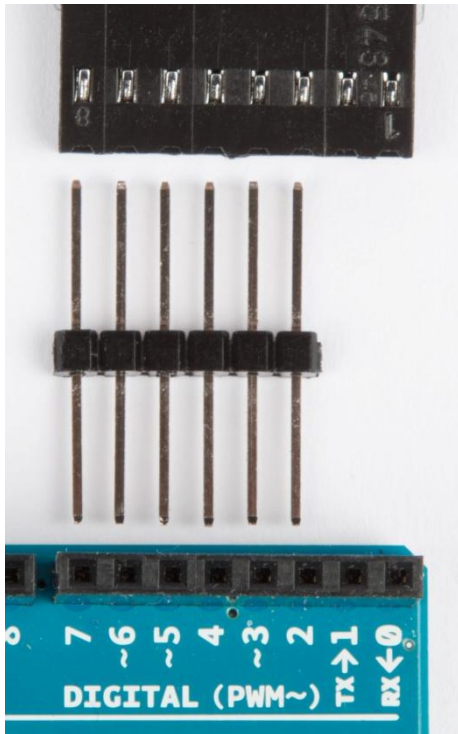
```
// I2c registers:  
#define R_CMD 0 // 7/8/* v  
#define R_LED_M 1 // Waarde  
#define R_ANALOG_H 2 // Hi-byte  
#define R_ANALOG_L 3 // Lo-byte  
#define R_LED_S 4 // 1/2 var  
#define R_PWM 5 // 4/5 var
```

- ▶ master code ingang slave -> uitgang master)

```
// Lees register 1 en stuur LED aan  
I2cTxBuffer[0] = 1; // te lezen register  
I2cSendReceive(SLAVE_ADDRESS, 1, 1, I2cTxBuffer, I2cRxBuffer); // schrijf  
digitalWrite(13, !I2cRxBuffer[0]); // Led aan of uit, afhankelijk van gele
```

Toetsenbord aansluiten

- ▶ Op de master Arduino



Digitale uitgang- Oefening (b)

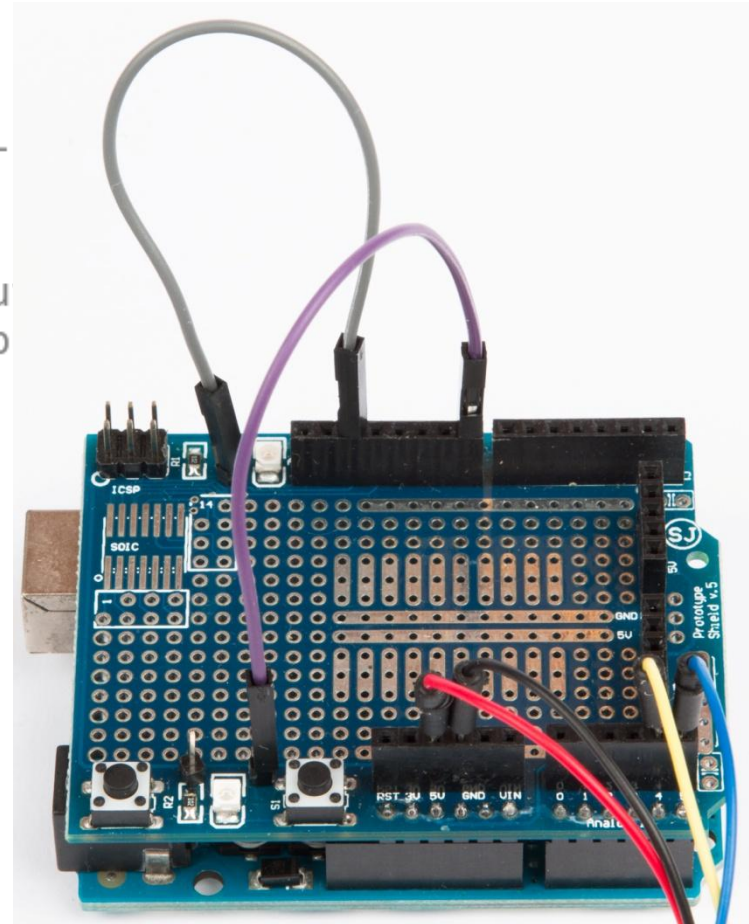
- ▶ Toetsen 1 en 2 (led aan, led uit)
- ▶ draadje pin13 / led op shield

```
#define R_LED_S
```

4

```
//-----  
// i2c interface init  
I2cSlaveRegistersInit(SLAVE_ADDRESS);  
pinMode(8, INPUT_PULLUP);    // LED_M input  
pinMode(13, OUTPUT);         // LED_S output
```

```
// Oefening b  
digitalWrite(13, I2cRegister[R_LED_S]);
```



Master code

```
// I2c registers:
#define R_CMD          0
#define R_LED_M        1
#define R_ANALOG_H     2
#define R_ANALOG_L     3
#define R_LED_S        4
#define R_PWM          5
```

```
// Defening y
char ch2 = MatrixKeyScan();
switch (ch2) {
case '1':
{
    SchrijfOnzeSlave(4, 1); // zet slave register 4 op 1
    break;
}
case '2':
{
    SchrijfOnzeSlave(4, 0); // zet slave register 4 op 0
    break;
}
```

```
void SchrijfOnzeSlave(byte Register, byte Waarde)
{
    byte I2cTxBuffer[2];

    I2cTxBuffer[0] = Register;
    I2cTxBuffer[1] = Waarde;
    I2cSendReceive(SLAVE_ADDRESS, 2, 0, I2cTxBuffer, NULL);
}
```

receiveEvent

i2c_slave_ws4

I2cSlaveRegisters

```
//-----  
// receiveEvent - process received data (interrupt handler)  
//-----  
//-----
```

```
void receiveEvent(int bytesReceived)  
{
```

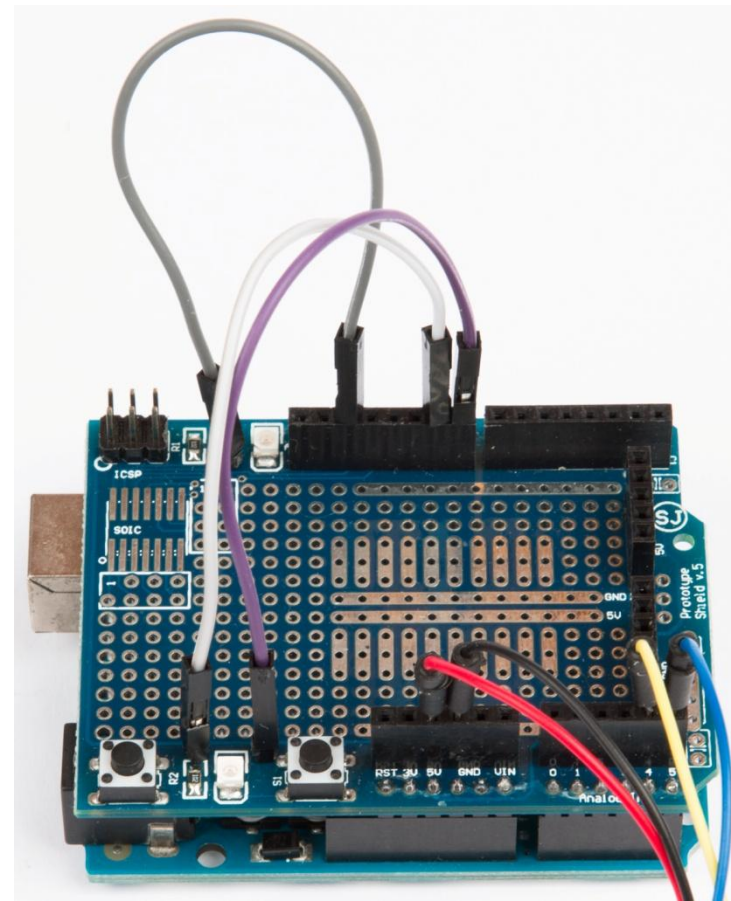
```
    _I2cNewDataFlag = true;
```

```
bool I2cNewData()  
{  
    bool r = _I2cNewDataFlag;  
    _I2cNewDataFlag = false;  
    return r;  
}
```


analogWrite – Oefening (c)

- ▶ Witte draadje D9 -> 2^e led van het shield
- ▶ Toetsen 4 en 5 (down / up)
- ▶ Register 5 = R_PWM

```
if (I2cNewData()) {  
    // Oefening c  
    analogWrite(9, I2cRegister[R_PWM]);  
}
```



Commando

- ▶ Register 0 = R_CMD
- ▶ Toetsen 7/8/* / 0 -> 1, 2, 3, 4
- ▶ Neem eenmalige actie en daarna terug op 0 zetten.

```
if (I2cNewData()) {  
  
    byte cmd = I2cRegister[R_CMD];  
    I2cRegister[R_CMD] = 0; // wis commando  
  
    if (cmd == 1) { // PWM LED uit  
        analogWrite(9, 0);  
        delay(2000);  
        analogWrite(9, I2cRegister[R_PWM]);  
    }  
  
    if (cmd == 2) { // PWM LED laag  
        analogWrite(9, 8);  
    }  
}
```



Commando – oefening (d)

- ▶ Toetsen 7/8/* / 0 -> 1, 2, 3, 4

```
// Oefening d
byte cmd = I2cRegister[R_CMD];
I2cRegister[R_CMD] = 0; // wis commando
```

```
if (cmd == 1) { // PWM LED uit
    analogWrite(9, 0);
    delay(2000);
    analogWrite(9, I2cRegister[R_PWM]);
}
```

```
if (cmd == 2) { // PWM LED laag
    analogWrite(9, 8);
    delay(2000);
    analogWrite(9, I2cRegister[R_PWM]);
}
```



```
if (cmd == 3) { // PWM LED hoog
    analogWrite(9, 32);
    delay(2000);
    analogWrite(9, I2cRegister[R_PWM]);
}
```

```
if (cmd == 4) { // PWM LED vol aan
    analogWrite(9, 255);
    delay(2000);
    analogWrite(9, I2cRegister[R_PWM]);
}
```

Commando – resultaat

- ▶ Onze commando's blokkeren hoofdlus
- ▶ => geen update van andere registers, R_LED_M, R_LED_S.
- ▶ => snel commando's achter elkaar -> laatste blijft staan.

Interrupts (events)

- ▶ Interrupts onderbreken programma op willekeurig punt.
- ▶ Acties met bytes (8 bits) zijn 'atomic'.
- ▶ Overige acties met samenhang beschermen!

Interrupts (events) – multi-byte

- ▶ 255 -> 256
- ▶ 0x00FF -> 0x0100

```
int a = analogRead(A0);  
I2cRegister[R_ANALOG_H] = a / 256;  
I2cRegister[R_ANALOG_L] = a;
```

- ▶ 0x00 0xFF
- ▶ 0x01 0xFF
- ▶ 0x01 0x00

Interrupts (events)

[Reference](#) [Language](#) | [Libraries](#) | [Comparison](#) | [Changes](#)

noInterrupts()

Description

Disables interrupts (you can re-enable them with `interrupts()`). Interrupts allow certain important tasks to happen in the background and are enabled by default. Some functions will not work while interrupts are disabled, and incoming communication may be ignored. Interrupts can slightly disrupt the timing of code, however, and may be disabled for particularly critical sections of code.

Example

```
void setup() {}

void loop()
{
  noInterrupts();
  // critical, time-sensitive code here
  interrupts();
  // other code here
}
```

Interrupts (events) – Oefening (e)

- ▶ Slave code
- ▶ Master print resultaat

```
// Oefening e
int a = analogRead(A0);
noInterrupts();
// Deze code wordt niet onderbroken door de (i2c) interrupt routine.
I2cRegister[R_ANALOG_H] = a / 256;
I2cRegister[R_ANALOG_L] = a;
interrupts();
```

COM8

```
Waarde analoge ingang: 707
Waarde analoge ingang: 706
Waarde analoge ingang: 707
Waarde analoge ingang: 707
Waarde analoge ingang: 707
Waarde analoge ingang: 707
Waarde analoge ingang: 707
Waarde analoge ingang: 707
```

Mogelijkheden SlaveRegister

Geschikt voor:

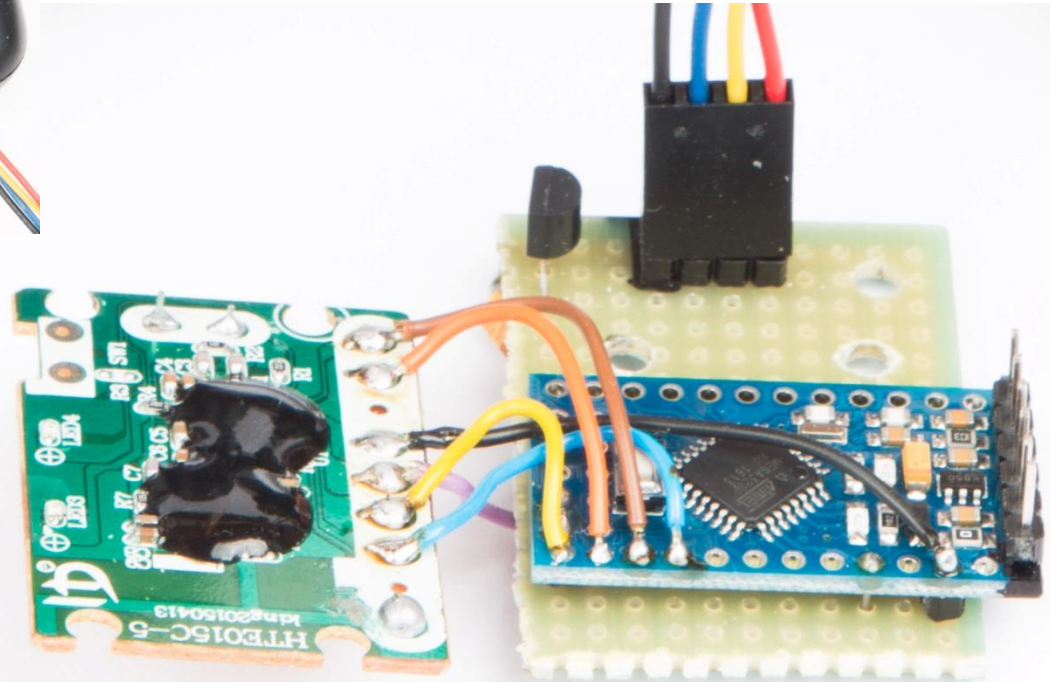
1. Waardes van master → slave.
2. Waardes van slave → master
3. Commando's van master → slave

Gebruik Wire class direct voor:

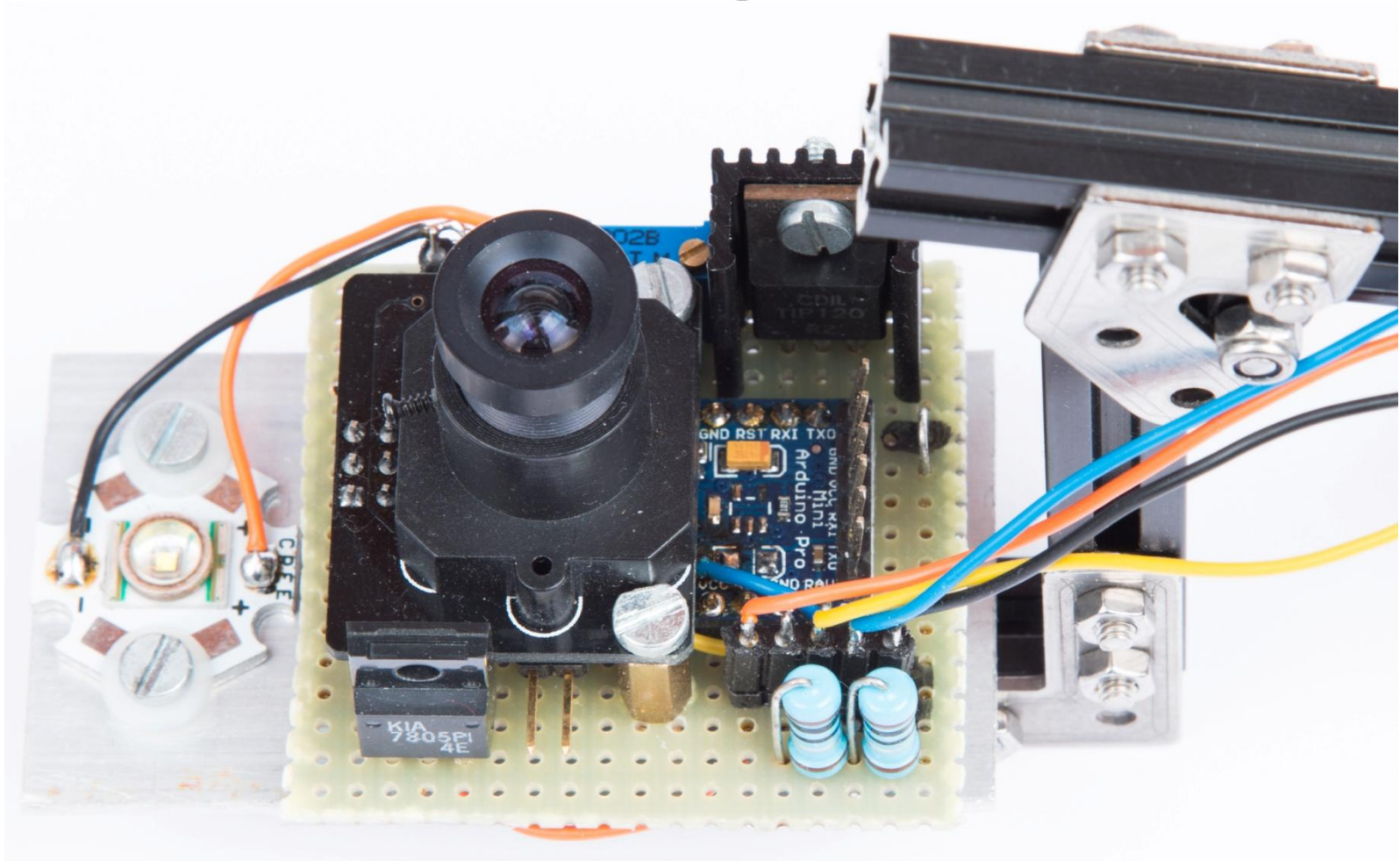
- ▶ 1 enkel byte aan data
- ▶ Stroom van data

Kortom, als je geen register/memory adres gebruikt.

Toepassing: Arduino library ontsluiten



Toepassing: Ontlasten hoofprocessor



Toepassing: Modulair ontwikkelen, flexibiliteit

