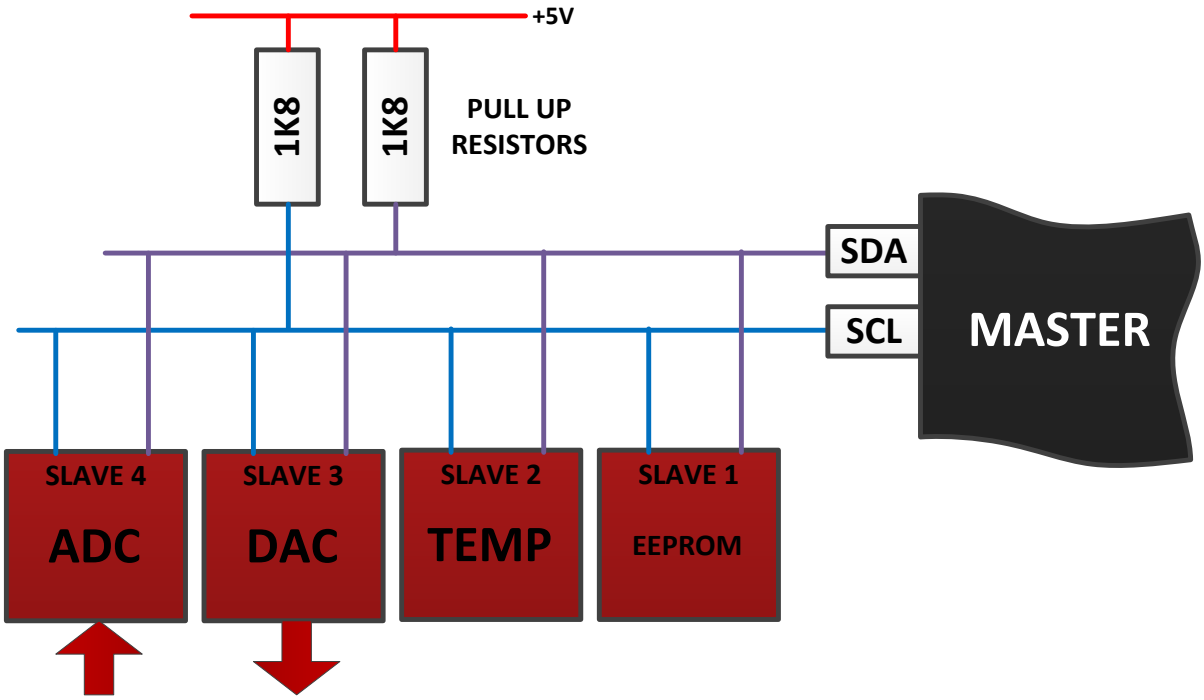
INHOUD

HET I2C PROTOCOL - THEORIE

SAMENVATTING:

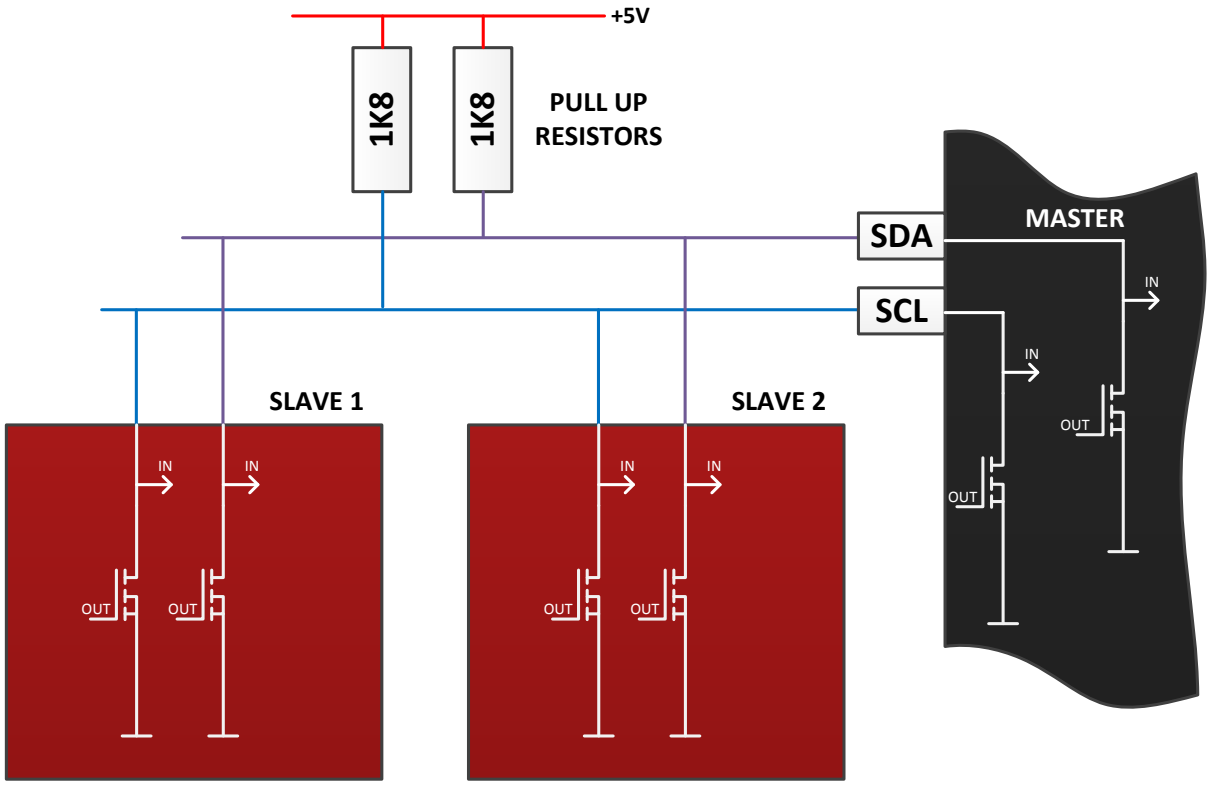
- I2C - IIC of Inter IC bus (ook wel eens "two wire interface" of "TWI" genoemd)
- In 1992 gepatenteerd door Phillips (nu NXP)
- Voor trage communicatie tussen IC's op korte afstanden (<1meter)
- Standaard versie 1.0 snelheid is 400kbps (versie 4.0 gaat tot 5Mhz)
- Synchroon protocol (datalijn en kloklijn)
- TTL signalen– 0 en 5V (0 en 3.3V ook mogelijk)
- 2 draden (+ gnd) – SCL (Serial clock) en SDA (Serial data)
- Master Slave principe (Multi-master mogelijk)
- De meeste modern uC hebben een hardware I2C module aan boord wat de communicatie met I2C slave modules heel eenvoudig maakt.
- Alle slaves hebben een uniek 7 (of 10) bit adres
- De SDA en SCL pins zijn van het type 'OPEN DRAIN' en moeten dus altijd via PULL up weerstanden naar de 5V verbonden worden.

HET I2C PROTOCOL



Op dit schema ziet u hoe 4 slaves – via I2C verbonden worden met één master. De data wordt gecommuniceerd via de SDA of serial data lijn – de klok wordt doorgegeven via de SCL of serial klok lijn.

Om elke slave afzonderlijk te kunnen aanspreken heeft elke slave een uniek adres in het netwerk. Dit adres is meestal een 7 bit adres, maar kan ook een 10 bit adres zijn.



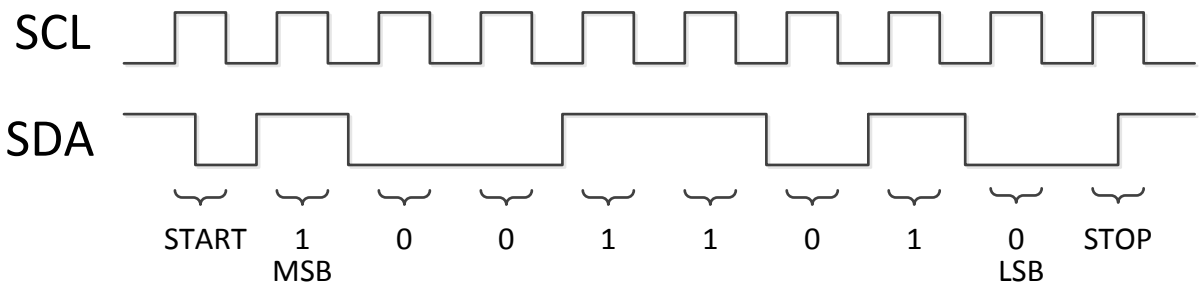
Het is heel belangrijk (maar vaak vergeten) om beide I2C lijnen via PULL up weerstanden met de 5 Volt te verbinden. Een ‘0’ wordt zo op de lijn gezet doordat één van de gebruikers de lijn laag trekt. Een ‘1’ wordt op de lijn gezet door eenvoudigweg de lijn los te laten. De pull up weerstand zorgt er dan voor dat de lijnspanning 5V wordt. Dit is ook meteen de idle toestand van de lijnen.

Enkel de master bepaald wanneer en wat er gecommuniceerd wordt. Een master kan zo data naar één van de slaves sturen en kan zo ook een van de slaves de opdracht geven om data op de bus te zetten. In beide gevallen genereert de master het kloksignaal op de SCL lijn.

Slaves zijn echter niet helemaal machteloos. Als de data op de bus te snel gaat voor de slave, dan kan de slave de kloklijn laag houden. De master merkt dit doordat deze ook constant de busniveau’s monitort kan zo z’n klok dus vertragen om de data trager te communiceren.

Zoals eerder vermeld wordt er gebruik gemaakt van een 7 bit adressering. Elke slave moet een uniek adres hebben op de bus. Dit laat dus 128 mogelijke slaves op de bus toe.

De master maakt de 8e bit in het slave adres laag om aan te geven of de master data wil schrijven naar de slave of hoog als de master met de eerstvolgende byte data wil ontvangen van de slave. Tijdens het ontvangen van data van slave naar master blijft de master wel de klok genereren.



Elk bericht bestaat uit meerdere blokken van 8 databits. De MSB wordt als eerste verzonden. De databits worden door de ontvanger enkel ingelezen als de SCL lijn hoog is – deze databits mogen dus slechts van toestand veranderen als de SCL lijn laag is. Een uitzondering hierop zijn de start en stop bits.

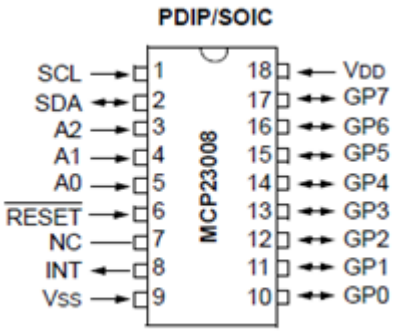
Elk bericht wordt begonnen met een startconditie en beëindigd met een stopconditie. Als de SDA lijn van hoog naar laag gaat als de SCL lijn hoog is, dan is dit een startconditie. Van laag naar hoog stelt dan een stopconditie voor.

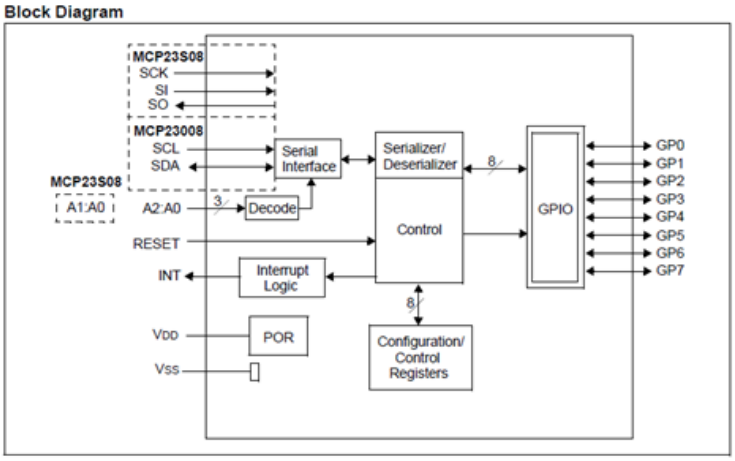
Na elke communicatie van 8 bits geeft de ontvangende module via een 9e ACK bit aan of de data correct ontvangen is. De ontvanger trekt dan de data-lijn gedurende 1 bittijd laag. Dit is een indicatie voor de zender dat de data correct is overgekomen. We spreken hier over zender en ontvanger omdat als de master opdracht geeft aan de slave om data door te sturen, de slave zender is en de master de ontvanger is.

VOORBEELDEN

Er bestaan heel wat verschillende IC's en modules die via I2C worden aangestuurd. We bespreken er hier kort enkele.

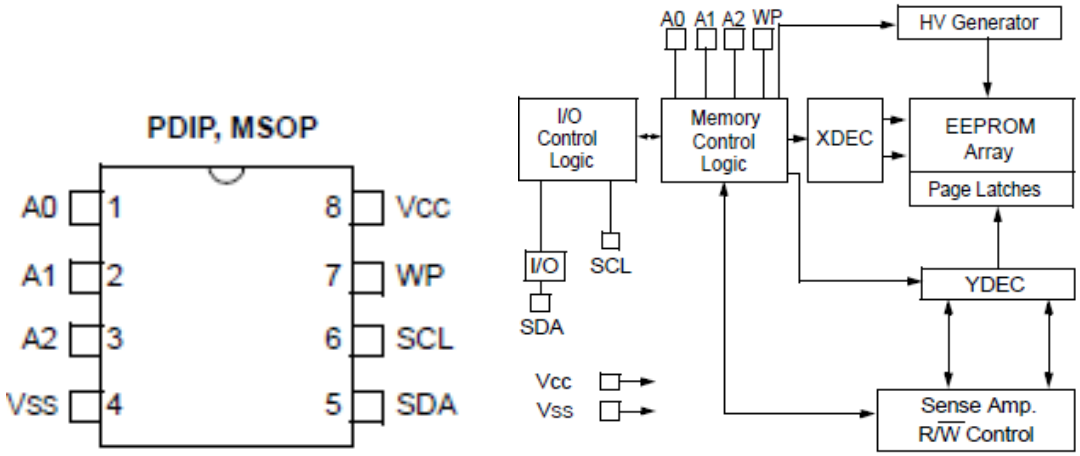
MCP23008 IO EXPANDER





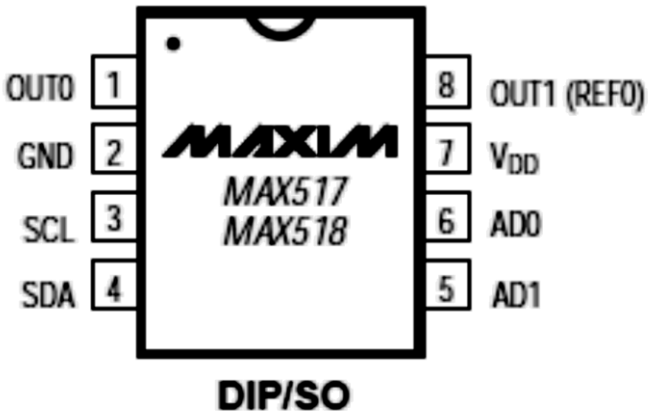
Zo heb je de MCP23008 IO expander. Deze IC heeft 8 lijnen die je identiek zoals bij je uC kan programmeren als input of als output om zo extra IO lijnen te creëren als je die te kort komt op je uC. Deze IO expander kan zelfs een interrupt genereren naar de uC als er iets gebeurt op één van de ingangen.

24LC32A 32KBIT EEPROM



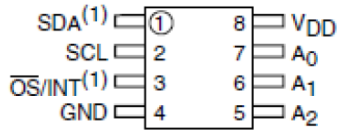
Deze 24LC32A is een 32Kbit Eeprom geheugen dat kan gebruikt worden om 32Kbit of 4Kbyte of dus 4000 bytes data permanent op te slaan. Dit is dus een serieuze uitbreiding op de 256 bytes Eeprom geheugen die standaard in onze microcontrollers aanwezig is.

MAX517 DAC

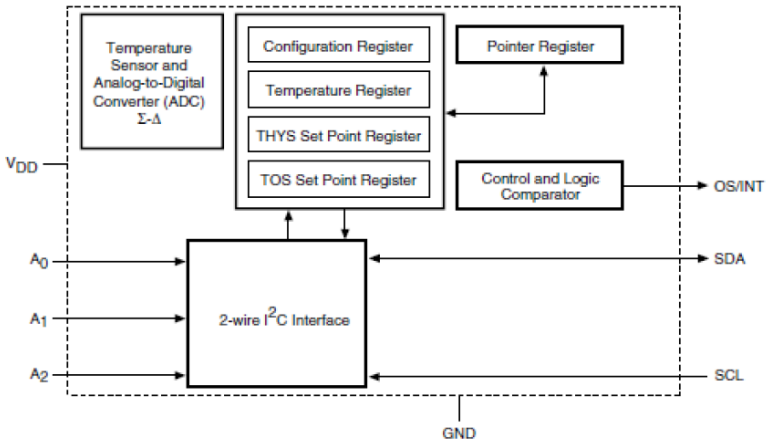


De MAX517 van MAXIM is een Digitaal Analoog omzetter die de 8 bit digitale waarden die het via I2C van de uC binnen krijgt terug omzet naar analoge waarden tussen gnd en de referentiespanning. Heel bruikbaar vermits de meeste uC geen DAC aan boord hebben.

STLM75M2E TEMPERATUUR SENSOR

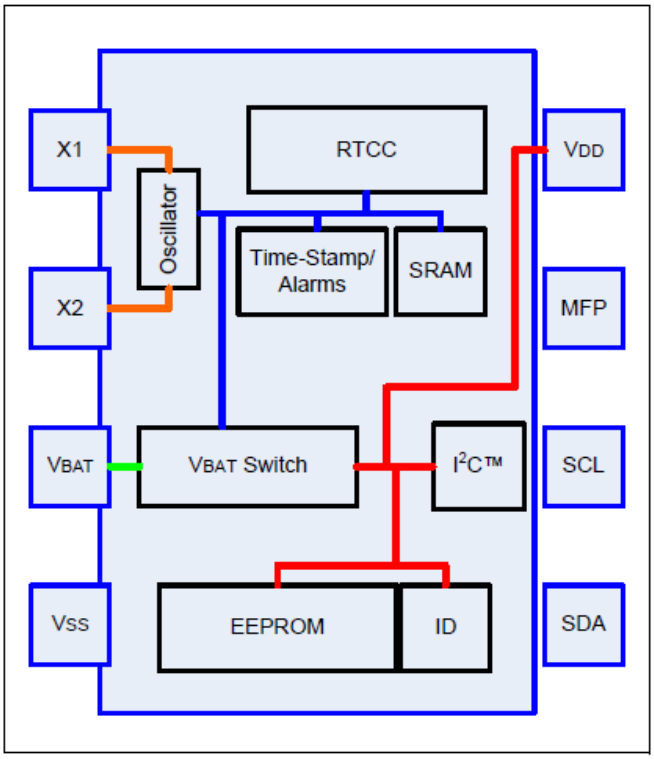


A2,A1 en A0 zijn selecteerbare adresingangen voor de 3 LSB's van het I²C interface adres. Ze kunnen hoog of laag worden om 8 unieke adres selecties te verkrijgen.



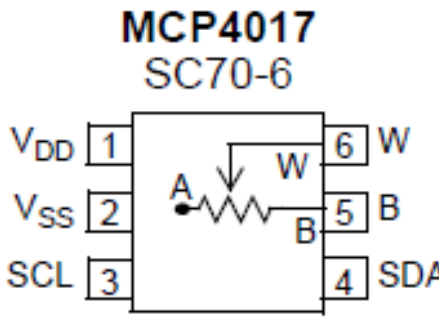
Deze STLM75M2E I2C ic is een temperatuursensor die een nauwkeurige temperatuurmeting kan doorgeven naar de UC. Deze IC heeft nogal wat mogelijkheden om het formaat te definiëren. De datasheet nauwkeurig lezen is dus de boodschap.

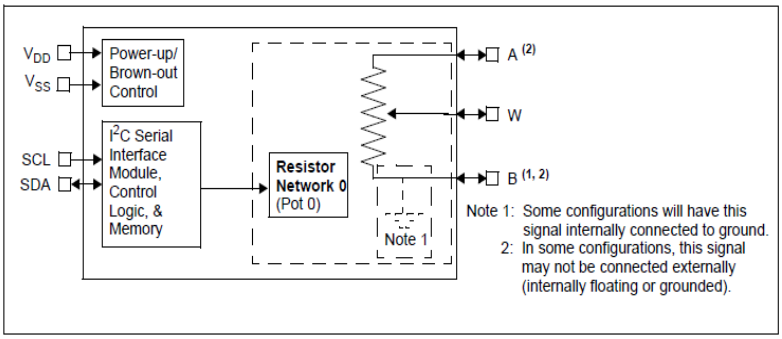
MCP79410 – RTCC (REAL TIME CLOCK AND CALENDER)



Deze MCP79410 Real Time clock and calender is één van de meer complexe IC's om te configureren en uit te lezen, maar eens je dit doorhebt, dan heb je met je uC wel toegang tot een nauwkeurige klok en kalender.

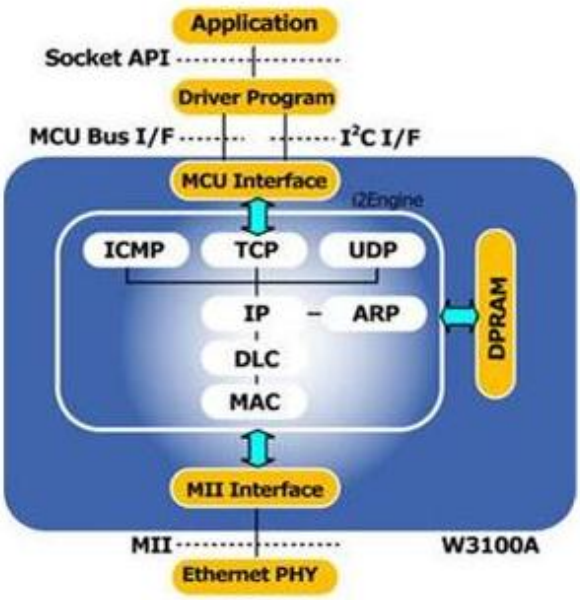
MCP4017 DIGITAL POTMETER





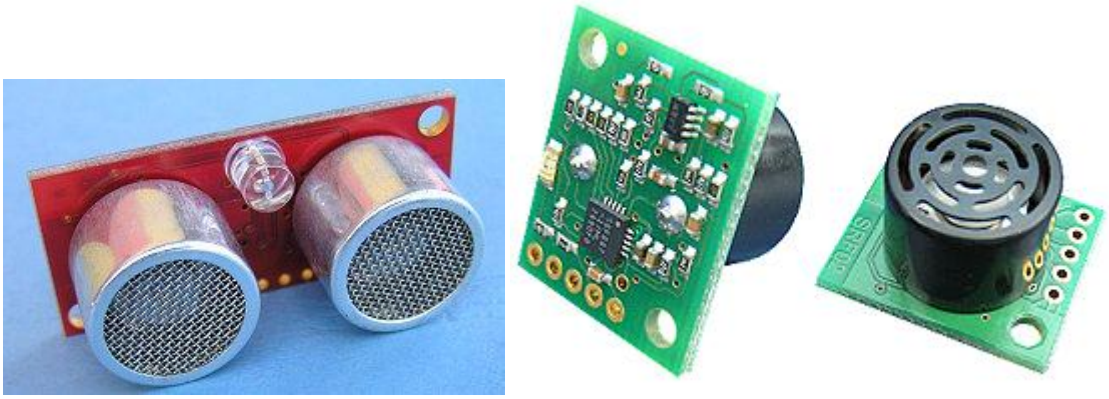
De MCP4017 is een digitale potentiometer. De ohmse waarde tussen pin 5 en 6 kan via I2C commando's worden aangepast tussen 0 en 5Kohm. Ideaal om een digitale volumeregeling te maken of iets dergelijks.

W3100A INTERNET MODULE



Deze W3100A module van WIZNET heeft alles aan boord om een uC via I2C met een netwerk of het internet te verbinden. Het complete internet bericht kan zo stap voor stap worden opgebouwd in de stack IC en kan zo van het moment het 'send' commando wordt gegeven via de Fysieke laag IC correct op het netwerk worden gezet. Ook het lezen van netwerkberichten is hiermee mogelijk. Deze module wordt trouwens ook gebruikt op de Internet E-block.

SRF08 OF SRF02 ULTRASONIC RANGE FINDER



Deze SRF08 of de goedkopere SRF02 modules van Devantech zijn ultrasone afstandsensoren die afstanden tot 6 meter zeer nauwkeurig kunnen meten. Het resultaat kan rechtstreeks in cm worden uitgelezen. Zeer toepasbaar in robotica toepassingen.

CMPS10 EN CMPS03 COMPASS MODULES

De CMPS10 en CMPS03 modules zijn digitale kompassen met een nauwkeurigheid van 0.5%. Ideaal als je soms het noorden kwijt bent.

MD03 H-BRUG

De MD03 is één van de vele H-bruggen in z'n reeks. Deze H brug kan 24V en maar liefst 20A aan kan. Ook deze kan worden aangestuurd via eenvoudige I2C commando's.

RD02 COMPLETE WHEEL SET

Deze RD02 is een complete set met wielen, motoren, encoders in de motoren, beugels en de volledige aansturing van de beide motoren. Beide wielen kunnen zo via I2C commando’s worden aangestuurd, de wheel encoders kunnen een zeer nauwkeurige waarde teruggeven van het aantal graden dat een wiel verdraaid is en de stroom en de spanning kan worden uitgelezen. Perfect dus om uw robotica project vooruit te laten gaan.

OPGAVEN:

- Bekijk de website van Devantech en zoek nog minstens 2 andere I2C compatibele producten.
- Ik wil 2 RGB leds aansturen via 6 PWM kanalen om zo alle kleuren te kunnen genereren. Zoek een I2C IC die minimaal 6 PWM uitgangen heeft die afzonderlijk in te stellen zijn via I2C commando’s
- Vul de datacom fiche zo volledig mogelijk aan:

DATACOM-FICHE

Naam:…… Klas: …….. Nr:…… Datum:……..

Te bespreken protocol: ……

Algemeen:

Parallel/Serieel		Karakterbeveiliging	pariteitsbit ? …?
Synchroon/Asynchroon		Karaktersynchronisatie	Startbit? Stopbit? …?
Simplex/half/Full duplex		Berichtbeveiliging	CRC? LRC/VRC? …?
Bitstuffing	Zoja - hoe?	Berichtsysnchronisatie	TSX ? , ETX ?, ….?
Hardware in uC en/ of bitbanging	Zit er hardware in de uC die deze communicatie rechtstreeks kan genereren? Zoja - plaats hier dan een blokschema. Zou bitbanging ook mogelijk zijn?		

7 lagen model

Application	Is er een applicatie die de link vormt tussen mens en machine - heb je deze zelf geschreven? In welke taal? Naam programma. Welke link is er met de mens?
Presentation	Wordt de informatie uit de applicatie op één of andere manier vertaald naar een standaardformaat? Is er een besturingssysteem aanwezig?
Session	Is er een module die op bepaalde momenten de communicatie opzet en op andere momenten weer terug afbreekt?
Transport	Is er een bepaald systeem opgezet zodat de zender bevestiging krijgt dat de ontvanger de data goed ontvangen heeft?
Network	Is er een module die bepaalt welke route de data moet volgen om van zender naar ontvanger te geraken.
Data link	Wordt de data opgedeeld in pakketten of frames? Hoe groot zijn deze frames? Startbit, stopbit, pariteitsbit, CRC controle,
Physical	Welke kabels worden er gebruikt, hoe ver kan er maximaal gezonden worden? Welke spanningen worden er gebruikt? Datasnelheid? Wordt er gemoduleerd? Welke connectoren? Return to zero/non return to zero/differentieel/…, …

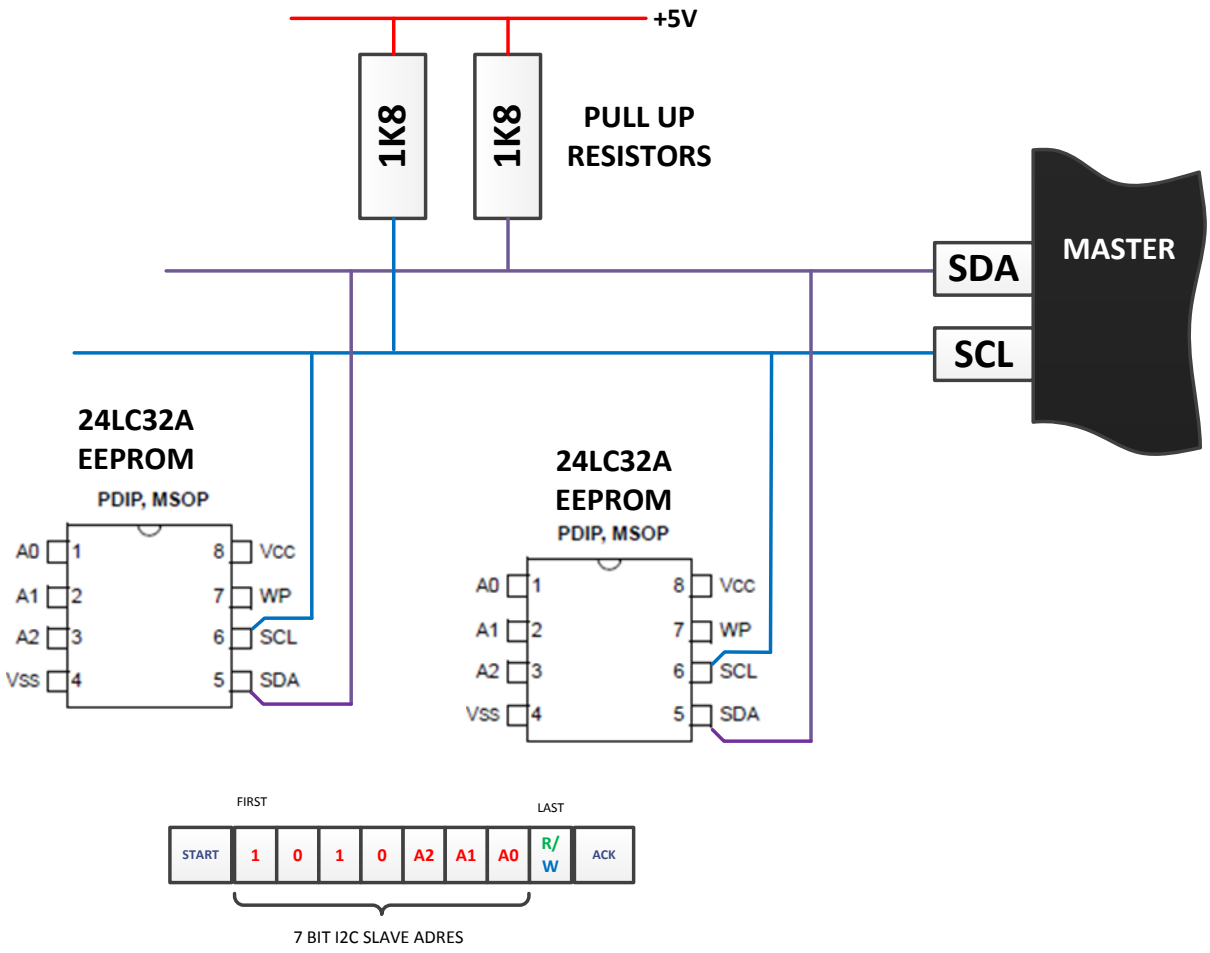
Metingen op signaal:

Meet met de oscilloscoop en/ of de logic analyser de datacommunicatie op en zet de beelden hier. Ontcijfer eventueel de data. Voorzie de beelden van een beetje uitleg.

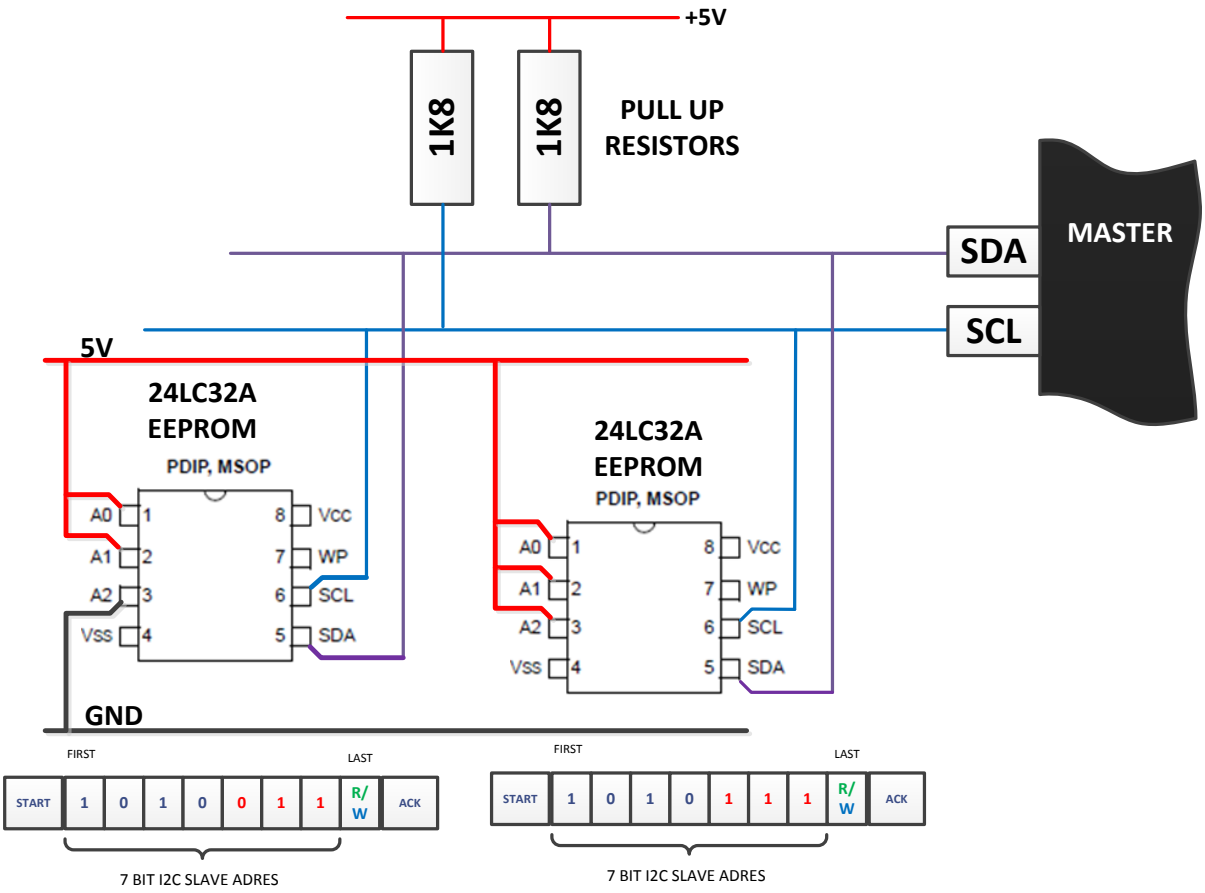
(verwijder al de oranje tekst - deze dient enkel als hulp)

HET I2C PROTOCOL - PRAKTISCH

ADRESSERING



Elke slave in een netwerk moet via een uniek adres te adresseren zijn. Stel dat we zoals hier 2 identieke 24LC32A Eeprom IC's willen aansluiten aan de I2C bus. In de datasheet zien we dat de 4 MSB's van het adres vastgelegd zijn door de fabrikant op 1010. Voor de 3 LSB's staat er A2, A1 en A0.



Deze letters komen overeen met 3 pins op de Eeprom IC. Deze 3 pins kunnen we verbinden met ofwel 5V ofwel GND om de 3 LSB's van het Eeprom adres uniek te maken binnen dit netwerk. Theoretisch zouden we op deze manier dus 7 verschillende identieke Eeproms uniek kunnen adresseren binnen dit I2C netwerk.

Een andere methode die meer gebruikt wordt bij grotere modules is dat elke module wordt geleverd met een standaard adres. Als je dit adres wil veranderen dan moet je deze module alleen aansluiten op het I2C netwerk en deze via enkele welomschreven stappen herprogrammeren naar een nieuw adres. Nadien kan je deze module dan met andere modules in een zelfde netwerk plaatsen.

I2C COMMANDO'S

Om I2C slave componenten aan te sturen heb je slechts 5 verschillende commando's nodig:

- START — Geeft het begin aan van een bericht
- SENDBYTE — Verstuur een 8 bit datablok (adres, index of data)
- RESTART — Om de lees-mode te starten (start zonder stop)
- READBYTE — Lees een 8 bit datablok in van de slave
- STOP — indicatie dat het bericht stopt

ACKNOWLEDGE BIT:

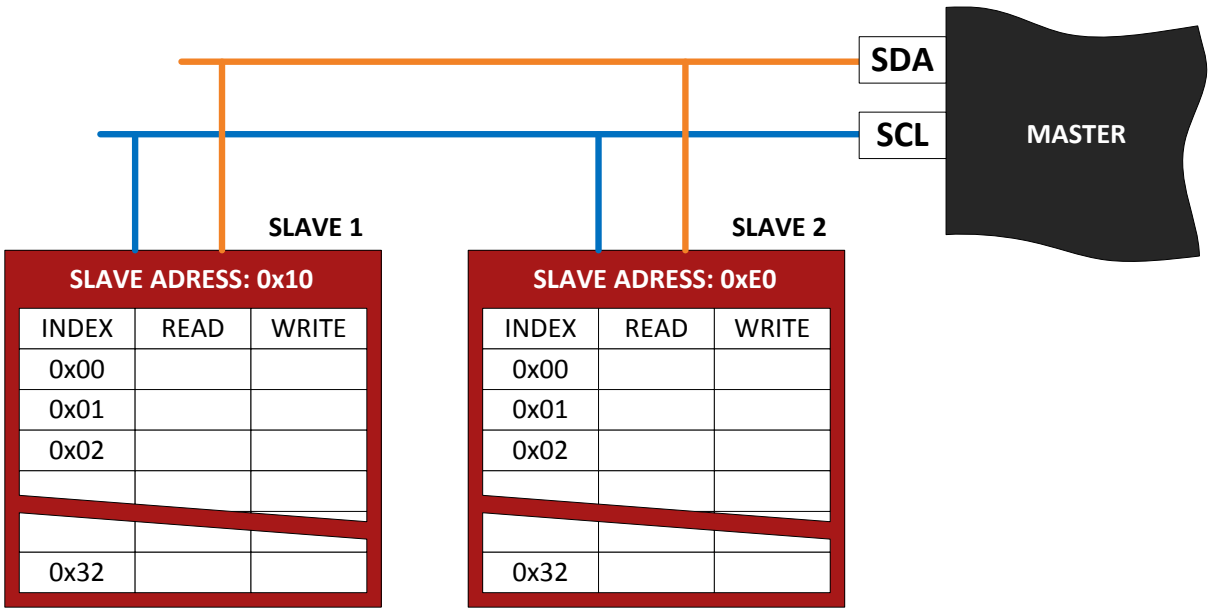
- Na elke SENDBYTE instructie zal de betreffende slave de lijn nog één extra 9e bittijd laag houden om aan de master duidelijk te maken dat de data goed ontvangen is.
- Na elke READBYTE instructie zal de master in dit geval de lijn één extra 9e bittijd laag houden zodat de slave weet dat de data correct is ontvangen.

READ OF WRITE

De meeste I2C modules werken op exact dezelfde manier. Er zijn 2 situaties:

1. WRITE data to the slave
2. READ data from the slave

We bekijken even dit blokschema. U ziet een master die via een SDA en SCL lijn verbonden is met 2 slaves. Beide slaves hebben een uniek adres. Slave 1 heeft als adres 0x10 en slave 2 heeft als adres 0xE0. Merk op dat slave adressen altijd even zijn vermits de LSB de bit is die bepaalt of er geschreven of gelezen wordt. 0 is schrijven en 1 is lezen.



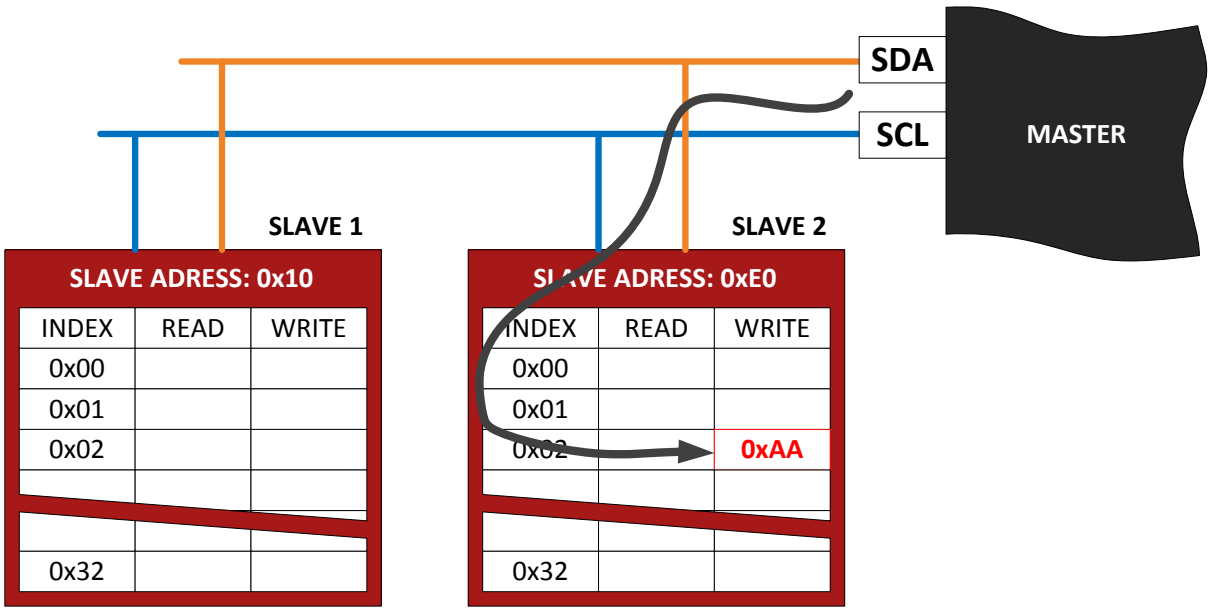
Naar de meeste I2C slave modules kan er worden geschreven maar er kan ook uit worden gelezen. In de module zelf wordt er via een index bepaald naar welke cel je schrijft of van welke cel je leest. De inhoud van deze cellen bepaalt hoe deze slave zal werken. Dikwijls – maar niet altijd – zijn de lees en schrijfcellen met dezelfde index toch gescheiden zoals je op dit blokschema kan zien.

WRITE DATA TO THE SLAVE

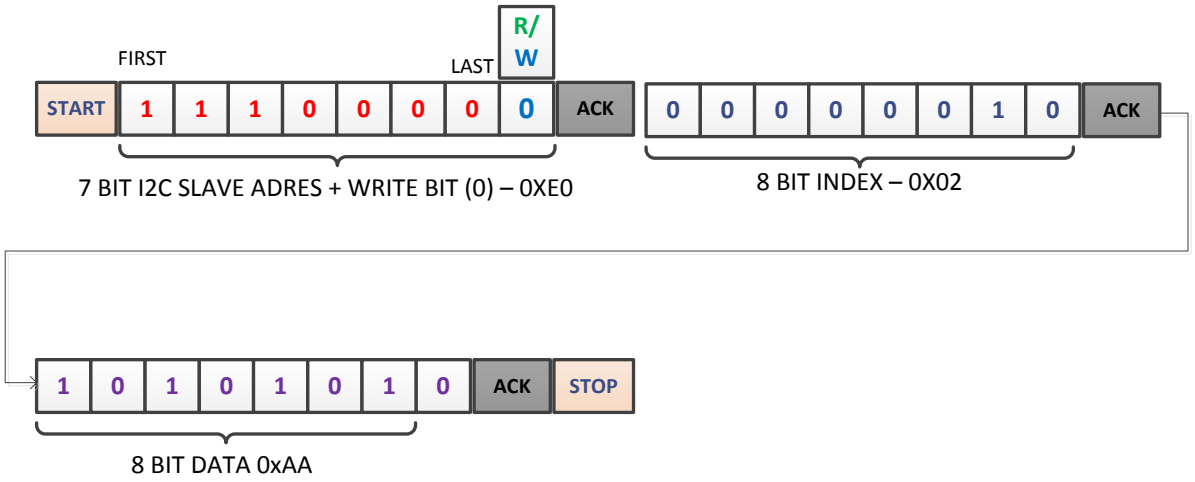
Voorbeeld: we schrijven de waarde 0xAA naar index 0x02 van slave-adres 0xE0 (SLAVE 2)

PROGRAMMA-STAPPEN SCHRIJVEN:

1	START	Start the message
2	SENDBYTE 0xE0	Send the slave address of Slave 2 0xE0 = 0b 1110 0000 - LSB is 0 so this is a write operation
3	SENDBYTE 0x02	Set the pointer in slave 2 at index 0x02
4	SENDBYTE 0xAA	Send data 0xAA to 'WRITE position' at index 0x02
5	STOP	Stop the message



GRAFISCHE VOORSTELLING SCHRIJVEN:



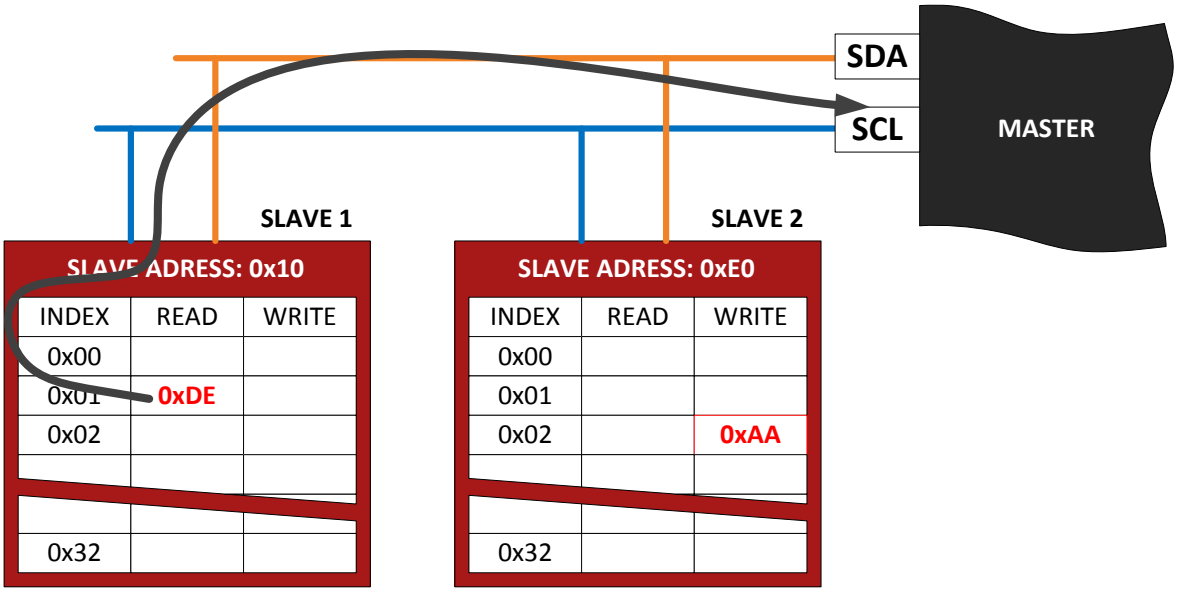
Een startconditie geeft het begin aan, dan volgt een 7 bit adres om de juiste slave te kiezen en 1 extra bit die we 0 maken waarmee we bepalen dat we naar de slave willen schrijven. Indien er in het netwerkt een slave staat met adres 0xE0 dan zal deze een ACK bit genereren om aan te geven dat het adres correct ontvangen is. Daarna stuurt de master de index van 8 bits door – in dit geval 0x02 om aan te geven naar welke plaats in de slave we data willen schrijven. Ook hier geeft de slave via een ACK bit aan dat de data goed ontvangen is. Meteen hierna zet de master de data op de bus die in index 0x02 van slave 0xE0 moet komen te staan. De slave bevestigt weerom met een ACK bit en de master sluit het bericht af met een STOPBIT.

DATA LEZEN VAN EEN SLAVE

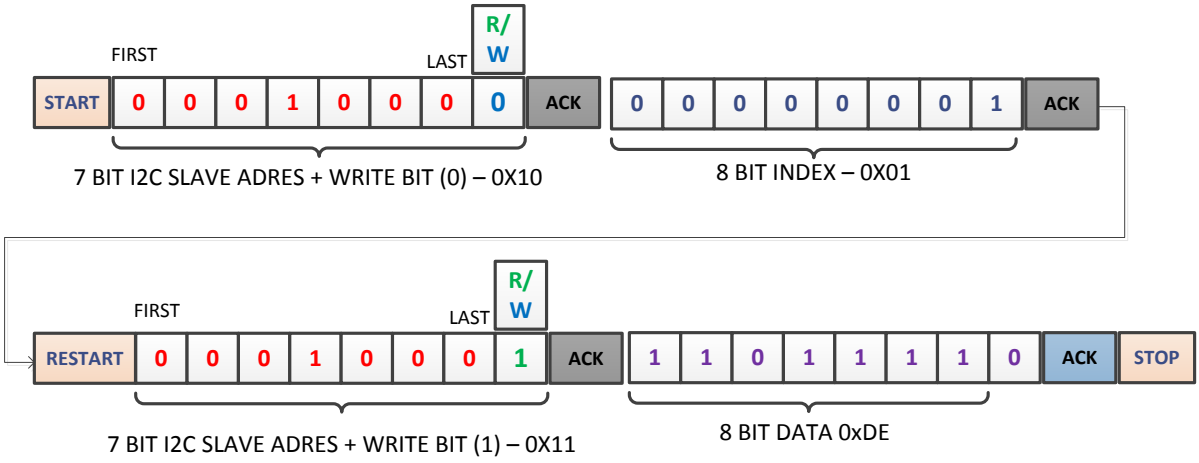
Bijvoorbeeld: we willen de data lezen van slave 1 (ADRES 0x10) op de index positie 0x01. We gaan er even vanuit dat deze data 0xDE is.

PROGRAMMASTAPPEN ONTVANGEN

1	START	Start the message
2	SENDBYTE 0x10	Send the slave address of Slave 2 0x10 = 0b 0001 0000 - LSB is 0 so this is a write operation
3	SENDBYTE 0x01	Set the pointer in slave 2 at index 0x01
4	RESTART	Restart instruction to indicate change to read mode
5	SENDBYTE 0x11	Send the slave address of Slave 2 – LSB = 1 0x11 = 0b 0001 0001 - LSB is 1 so this is a READ operation
6	READBYTE	Master reads data from 'READ position' at index 0x01 – (all that the master does here is generate 8 clock pulses and generate an ACK bit. To clock in the 8 bit data from the slave)
7	STOP	Stop the message



GRAFISCHE WEERGAVE ONTVANGEN

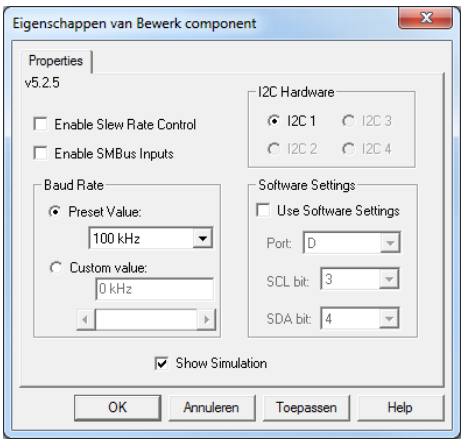


De start geeft het begin aan. Als eerste wordt het slave adres gestuurd – met als laatste bit nog steeds 0. De slave bevestigt met een ACK bit. Als tweede wordt de 8 bit index verstuurd om de slave duidelijk te maken van welke index we de data willen gaan lezen. De slave zal z'n pointer op deze index zetten en bevestigen met een ACK bit. Vervolgens wordt er een restart gegeven – dit is een start zonder dat er eerst een stop geweest is zodat de slave weet dat de volgende dat weerom een adres is. Het adres is nog steeds 0x10, maar vermits we nu aan de slave duidelijk willen maken dat we data willen lezen maken we de LSB hoog. Het volledige adres wordt nu dus 0x11 zoals hier te zien is en de slave bevestigt met een ACK bit. Meteen hierna zet de slave – op het ritme van de klok die de master nog steeds genereert – de gevraagde data uit index 0x01 op de SDA lijn. De master bevestigt deze keer de goede ontvangst met een ACK bit en sluit het bericht af met een stop bit.

DEMO-PROGRAMMA’S

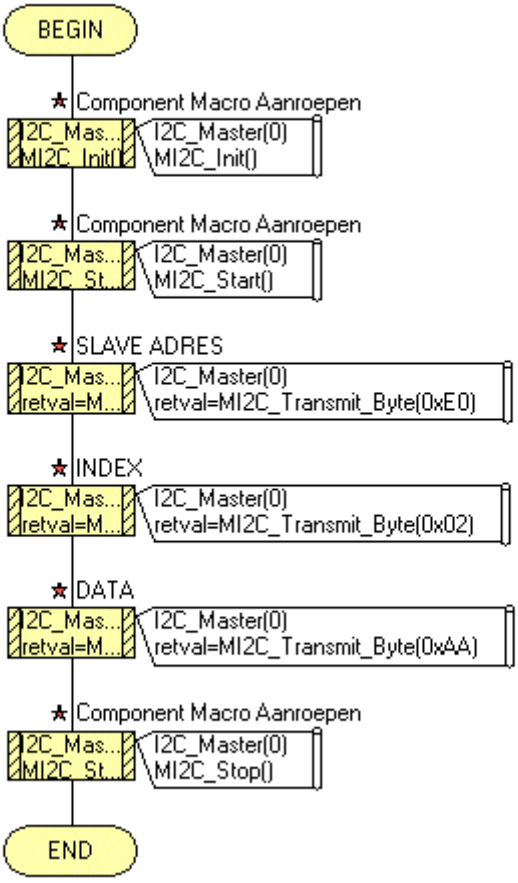
Onder de I2C map kan je voorbeeldprogramma’s terugvinden voor Flowcode en onder C.

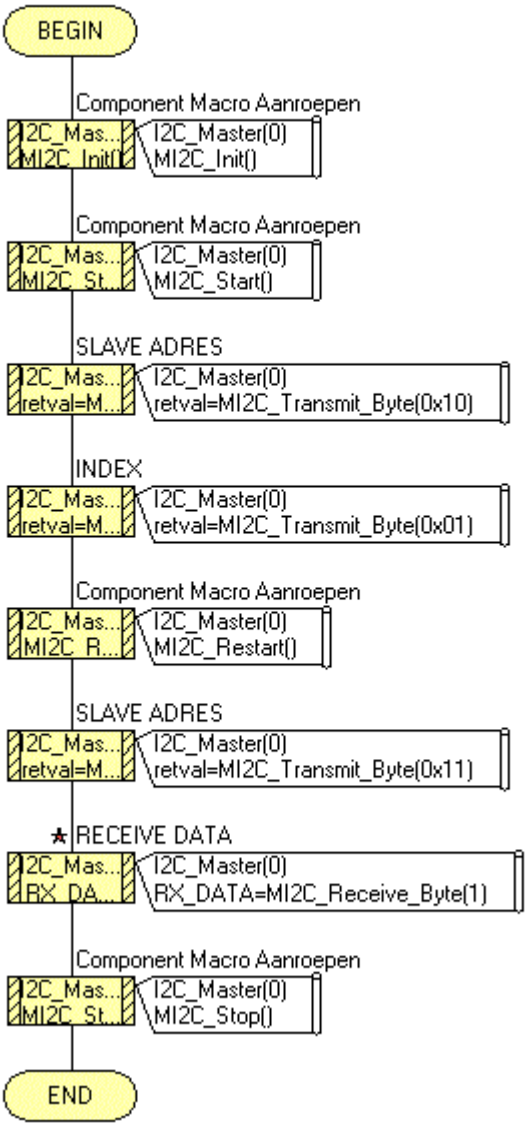
FLOWCODE EN I2C



Onder Flowcode kan je kiezen of de I2C signalen via de daarvoor voorziene I2C hardware wordt gegenereerd of dat de I2C signalen op de pins naar uw keuze worden ‘gebitbanged’ via de ‘use software settings’ optie. Om de processor te ontlasten gaat de voorkeur naar de hardware.

Het linkse voorbeeld is het ‘FC I2C WRITE’ programma en het rechtse is het ‘FC I2C READ’ programma – beiden terug te vinden onder de I2C map.


BEGIN
Component Macro Aanroepen
I2C_Mas...
MI2C_Init()
I2C_Master(0)
MI2C_Init()
Component Macro Aanroepen
I2C_Mas...
MI2C_St...
I2C_Master(0)
MI2C_Start()
SLAVE ADRES
I2C_Mas...
retval=M...
I2C_Master(0)
retval=MI2C_Transmit_Byte(0xE0)
INDEX
I2C_Mas...
retval=M...
I2C_Master(0)
retval=MI2C_Transmit_Byte(0x02)
DATA
I2C_Mas...
retval=M...
I2C_Master(0)
retval=MI2C_Transmit_Byte(0xAA)
Component Macro Aanroepen
I2C_Mas...
MI2C_St...
I2C_Master(0)
MI2C_Stop()
END


BEGIN
Component Macro Aanroepen
I2C_Mas...
MI2C_Init()
I2C_Master(0)
MI2C_Init()
Component Macro Aanroepen
I2C_Mas...
MI2C_St...
I2C_Master(0)
MI2C_Start()
SLAVE ADRES
I2C_Mas...
retval=M...
I2C_Master(0)
retval=MI2C_Transmit_Byte(0x10)
INDEX
I2C_Mas...
retval=M...
I2C_Master(0)
retval=MI2C_Transmit_Byte(0x01)
Component Macro Aanroepen
I2C_Mas...
MI2C_R...
I2C_Master(0)
MI2C_Restart()
SLAVE ADRES
I2C_Mas...
retval=M...
I2C_Master(0)
retval=MI2C_Transmit_Byte(0x11)
RECEIVE DATA
I2C_Mas...
RX_DA...
I2C_Master(0)
RX_DATA=MI2C_Receive_Byte(1)
Component Macro Aanroepen
I2C_Mas...
MI2C_St...
I2C_Master(0)
MI2C_Stop()
END

HITECH C EN I2C

Voor de C programma's is er een speciale I2C library geschreven. Importeer de bestanden I2C_LIB.c en I2C_LIB.h in uw C programma. Door gebruik te maken van de functies I2C_SEND_MESSAGE en I2C_RECEIVE_MESSAGE uit deze library wordt de communicatie in C wel heel eenvoudig. U kan vanzelfsprekend ook de individuele functies gebruiken. Bekijk de lib.h file voor de correcte benamingen.

I2C WRITE in C

```
#include <htc.h>
#include "I2C_LIB.h"
#define _XTAL_FREQ  19660800
__CONFIG(0x2ff2);  // set HS Xtal etc
__CONFIG(0x3fff);

void main(){
ANSEL = 0;        // ad inputs as normal IO
ANSELH = 0;       // ad inputs as normal IO
OPTION_REG = 0xC0;
__delay_ms(250);                    //wait for stable power supply

I2C_INIT(); // initialize I2C

// write data 0xAA to index 0x02 of slave adres 0xE0
I2C_SEND_MESSAGE(0xE0,0x02,0xAA);

while (1){} //do nothing
}
```

I2C READ in C

```
#include <htc.h>
#include "I2C_LIB.h"
#define _XTAL_FREQ  19660800
__CONFIG(0x2ff2);  // set HS Xtal etc
__CONFIG(0x3fff);

void main()
{
ANSEL = 0;        // ad inputs as normal IO
ANSELH = 0;       // ad inputs as normal IO
OPTION_REG = 0xC0;

__delay_ms(250); //wait for stable power supply

volatile unsigned char X=0;
I2C_INIT(); // initialize I2C

// read data from slave adres 0x10, index 0x01
// put data in variable X
X = I2C_RECEIVE_MESSAGE(0x10,0x01);

while (1){} //do nothing
}
```

I2C_LIC.c LIBRARY

```
#include <htc.h>
#define _XTAL_FREQ  19660800     // oscillator frequency for _delay()

//Initialise PIC16F887 for Hardware I2C operation
void I2C_INIT()
{
   SSPSTAT |=0b10000000;      //_st_bit (SSPSTAT,SMP),Slew Rate Control Disabled
   SSPSTAT |=0b01000000;      //_cr_bit (SSPSTAT,CKE),Disable SMBus specific inputs
   SSPCON = 0x28;            //Setup I2C into Master Mode
   SSPADD = 98;              //Set the Baud Rate
   SSPCON2 = 0x00;           //Clear the control bits
   GIE = 1;                 //_st_bit(INTCON, GIE);
   TRISC = TRISC|0b00001000;  //Configure SCL as Input
   TRISC = TRISC|0b00010000;  //Configure SDA as Input
}

// send a start bit sequence
void I2C_START_MESSAGE()
{
   PIR1 &=~0b00001000;       //cr_bit(PIR1,SSPIF),Clear SSP interrupt flag
   SSPCON2 |= 0x01;          //st_bit(SSPCON2,SEN),Initiate start condition
   while(SSPCON2 & 0x01);     // - as long as SEN is High - Wait for start bit to be generated
}

// send a repeated start bit sequence
void I2C_RESTART()
{
   PIR1 &=~0b00001000;       //cr_bit(PIR1,SSPIF),Clear SSP interrupt flag
   SSPCON2 |= 0x02;          //st_bit(SSPCON2,RSEN),Initiate restart condition
   while(SSPCON2 & 0x02);     //  while(ts_bit(SSPCON2,RSEN)),Wait for restart bit to be generated
}

// send a stop bit sequence
void I2C_STOP_MESSAGE()
{
   PIR1 &=~0b00001000;       //cr_bit(PIR1,SSPIF),Clear SSP interrupt flag
   SSPCON2 |= 0x04;          //st_bit(SSPCON2,PEN),Initiate stop condition
   while(SSPCON2 & 0x02);     //while(ts_bit(SSPCON2,PEN)),Wait for stop bit to be generated
   __delay_ms(10);           //Wait before reusing the I2C BUS
}

// transmit a byte (this can be a devide adress - an index or just data - of 8 bit
// if return = 1 : No ack bit was received from the slave - if return = 0 = a correct acknowledge bit was received
char I2C_TRANSMIT_BYTE(char Data)
{
   PIR1 &=~0b00001000;             //cr_bit(PIR1,SSPIF),Clear SSP interrupt flag
   SSPBUF=Data;                   //Send byte
     while((PIR1 & 0x08) == 0);     //Wait for control bit to be sent
       if(SSPCON2 & 0b01000000)    //Check Acknowledgement
          return (1);           //No Acknowledgement
          else return (0);       //Acknowledgement received
}

//receive a byte of data - requested by the master and sent by the slave
```

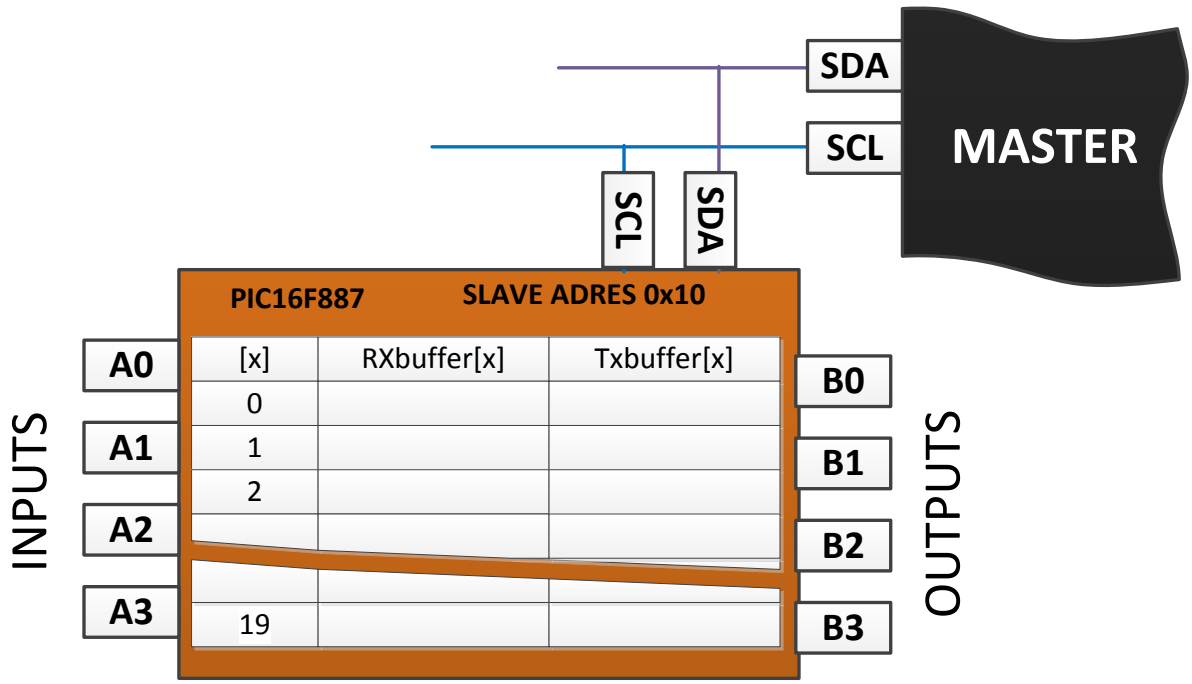
```
// returns a byte of incoming data
// if last = 1 - last byte was received - If last = 0 - the master expects further I2C_RECEIVE_BYTE instructions
char I2C_RECEIVE_BYTE(char Last)
{
  PIR1 &=~0b00001000;            //cr_bit(PIR1,SSPIF);
  SSPCON2 |=0x08;               //st_bit(SSPCON2,RCEN),Initiate Read
  while((PIR1 & 0x08) == 0);      //while(ts_bit(PIR1,SSPIF) == 0),Wait for data read
    if (Last)
      SSPCON2 |=0b00100000;     //st_bit(SSPCON2,ACKDT),Send Nack
    else SSPCON2 &=~0b00100000;   //cr_bit(SSPCON2,ACKDT),Send Ack
      SSPCON2 |=0b00010000;     //st_bit(SSPCON2,ACKEN),Initiate Nack
  while(SSPCON2&0b00010000);      //(ts_bit(SSPCON2,ACKEN)) Wait for data read
  return(SSPBUF);               //Store incoming data
}

//combination of all the I2C instructions to send a byte of date to a specific
// index in a certain device
void I2C_SEND_MESSAGE(char SLAVE_ADRES, char INDEX, char DATA)
{
    char RetVal = 0;
    I2C_START_MESSAGE();            //Start transaction
    I2C_TRANSMIT_BYTE(SLAVE_ADRES); //Transmit Device Address
    I2C_TRANSMIT_BYTE(INDEX);     //Transmit Internal Address
    I2C_TRANSMIT_BYTE(DATA);      //Send Data byte
    I2C_STOP_MESSAGE();             //Stop Transaction
}

//combination of all the I2C instructions to receive a byte of date from a specific
// index in a certain device
char I2C_RECEIVE_MESSAGE(char SLAVE_ADRES, char INDEX)
{

    char RetVal = 0;

    I2C_START_MESSAGE();              //Start transaction
    I2C_TRANSMIT_BYTE(SLAVE_ADRES);   //Transmit Device Address
    I2C_TRANSMIT_BYTE(INDEX);       //Transmit Internal Address

    I2C_RESTART();              //Restart transaction

    SLAVE_ADRES = SLAVE_ADRES | 0x01;     //Change Device ID to read mode
    I2C_TRANSMIT_BYTE(SLAVE_ADRES);   //Transmit Device Address
    RetVal = I2C_RECEIVE_BYTE(1);     //Read data at address
    I2C_STOP_MESSAGE();               //Stop Transaction

    return RetVal;              //Return data

}
```

OPGAVEN:

- Test alle demo-programma's uit. Je zal ze wel eerst moeten aanpassen aan de specifieke slave die je gebruikt (slave adres – index – data aanpassen volgens datasheet)

ZELF EEN I2C SLAVE-IC MAKEN.

Met onze PIC16F887 microcontroller is het ook mogelijk om zelf een I2C slave IC te maken. De datasheet biedt een bepaalde ondersteuning maar de voorbeeldcode is onbruikbaar en ook op het internet is het hier nog niet veel bruikbare code over te vinden.



In dit geval ontwerpen we en I2C slave waar er 20 index-registers zijn om naar te schrijven (van master naar slave) en eveneens 20 index registers zijn om van te lezen (van slave naar master). De master bepaalt dus na een correct I2C schrijf bericht wat de data is in de RXbuffer registers en de master kan de data uit de TXbuffer registers lezen.

Data in de RX buffer registers kan zo mogelijk gebruikt worden om outputs aan te sturen en mogelijke input waarden kunnen zo in het programma in de TXbuffer registers geplaatst worden om op hun beurt te worden uitgelezen door de master. De plaats waar deze registers best kunnen worden gelezen en geschreven staat duidelijk in het programma aangegeven. Het volledige programma is te vinden onder de I2C map onder de naam: "DEMO I2C SLAVE 16F887"

I2C slave programma

```
#include <htc.h>
#define _XTAL_FREQ  19660800
__CONFIG(0x2ff2);  // set HS Xtal etc
__CONFIG(0x3fff);

unsigned char TXbuffer[20];    // Array registers for TX from slave to master
unsigned char RXbuffer[20];    //Array register for RX from Master to slave

void main()
{

//I2C Slave settings
SSPADD = 0x10;     //!!!!I2C device SLAVE ADRESS 0x10!!!!
```

```
SSPSTAT = 0x00;    // not a lot
SSPCON = 0x39;     // enable mask access
SSPADD = 0xff;     // no masking
SSPCON = 0x36;     // 7-bit slave mode

SSPIF = 0;         // Clear I2C interrupt flag
SSPIE = 1;         // enable I2C interrupts

// Enable global interrupts
PEIE = 1;      // enable peripheral interrupts
GIE = 1;       // enable global interrupts

   while (1)
     {
     // read and use data from RXbuffer[x] (x = 0-19)
     // write data to TXbuffer[x] (x = 0-19)
     }
}
void interrupt I2C(void) // the interrupt
{
static unsigned char data, idx=0, wr_addr=0;
char i2c_data;

// if I2C Interrupt Occurs:
 if(SSPIF) {            // I2C interrupt
  SSPIF = 0;

  if(!D_nA) {           // low = address
   wr_addr=0;
  }
  if(R_nW) {            // high = read from this program
   SSPBUF = data;         // send data
   CKP = 1;             // release I2C clock line
   if(idx<19) ++idx;      // limit index to 20 bytes
   data = TXbuffer[idx];  // speed up next read
  }
  else {
   i2c_data = SSPBUF;     // read incoming data
   wr_addr++;
   if(wr_addr==2) {       // 1st byte written is internal location
    idx = i2c_data;       // get 0-19 index
    if(idx>19) idx=19;    // limit index
    data = TXbuffer[idx]; // to speed up possible read
   }
   else {
    if(wr_addr>2)         // only 1st 3 bytes are writable
     {
     RXbuffer[idx]=i2c_data;
     ++idx;
    }
   }
  }
  SSPOV = 0;
 }
}
```

OPGAVEN:

- Ontwerp zelf een I2C slave IC met je 16F887 die de 8 inputs van PORTB van de slave uitleesbaar maakt door de master.
- Ontwerp zelf een I2C slave IC met je 16F887 die de 8 outputs van PORTD van de slave aanstuurbaar maakt door de master.
- Ontwerp met je 16F887 en I2C slave IC die een 8 bit IO expander is. Je moet dus via een bepaald register kunnen bepalen welke pins Input en welke pins output zijn. De waarde van de input pins moet uit te lezen zijn door de master en de waarde van de output pins moet aan te sturen zijn door de master.
- Ontwerp een 8 kanaals I2C PWM module om 8 leds te dimmen. De frequentie voor leds moet slechts 50Hz zijn. De PWM waarden moeten individueel regelbaar zijn met een waarde van 0-255. Gebruik hiervoor zeker een timer.