DIC-TEST

# **Mögliche Fragen**

## LCD

* 4bit Transfer aufzeichnen können
* 1 Byte hinaus schreiben (ähnlich der LCDOut4Bit Funktion)
* Wie schaut 4 Bit Modus aus?
* Wie sind RS-, Enable und R/W-Leitung zu setzen? (RS = Register Set)
* Busy Flag zum kompletten Transfer aufzeichnen (Impuls-Zeit-Diagramm)

## I2C

* Wie schaut ein I2C Bus aus?
* Wie steuert man einen I2C Bus an?
* 1 byte hinaus schreiben
* Seite 1-4 I2C-Bus auswendig

Inhalt

[**Mögliche Fragen** 1](#_Toc24798764)

[LCD 1](#_Toc24798765)

[I2C 1](#_Toc24798766)

[**LCD** 2](#_Toc24798767)

[4-Bit Modus 2](#_Toc24798768)

[LcdOut4Bit 4](#_Toc24798769)

[**I2C** 5](#_Toc24798770)

[Lern einfach die I2C-Word 5](#_Toc24798771)

[I2C Datentransfer 5](#_Toc24798772)

# **LCD**

Vee = Display Helligkeit (0V-0,5V)

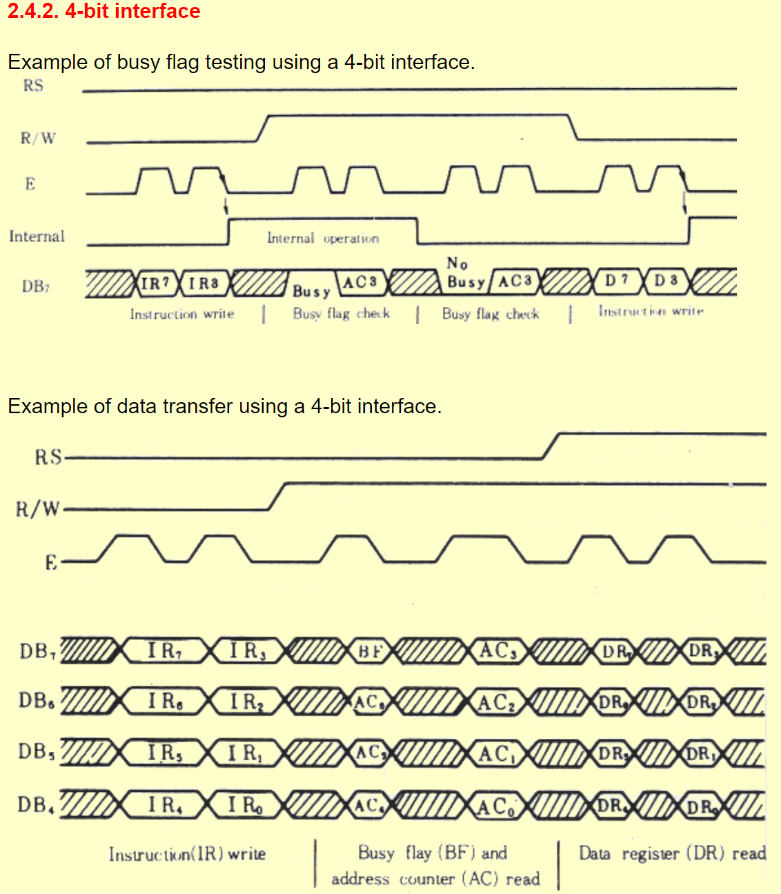
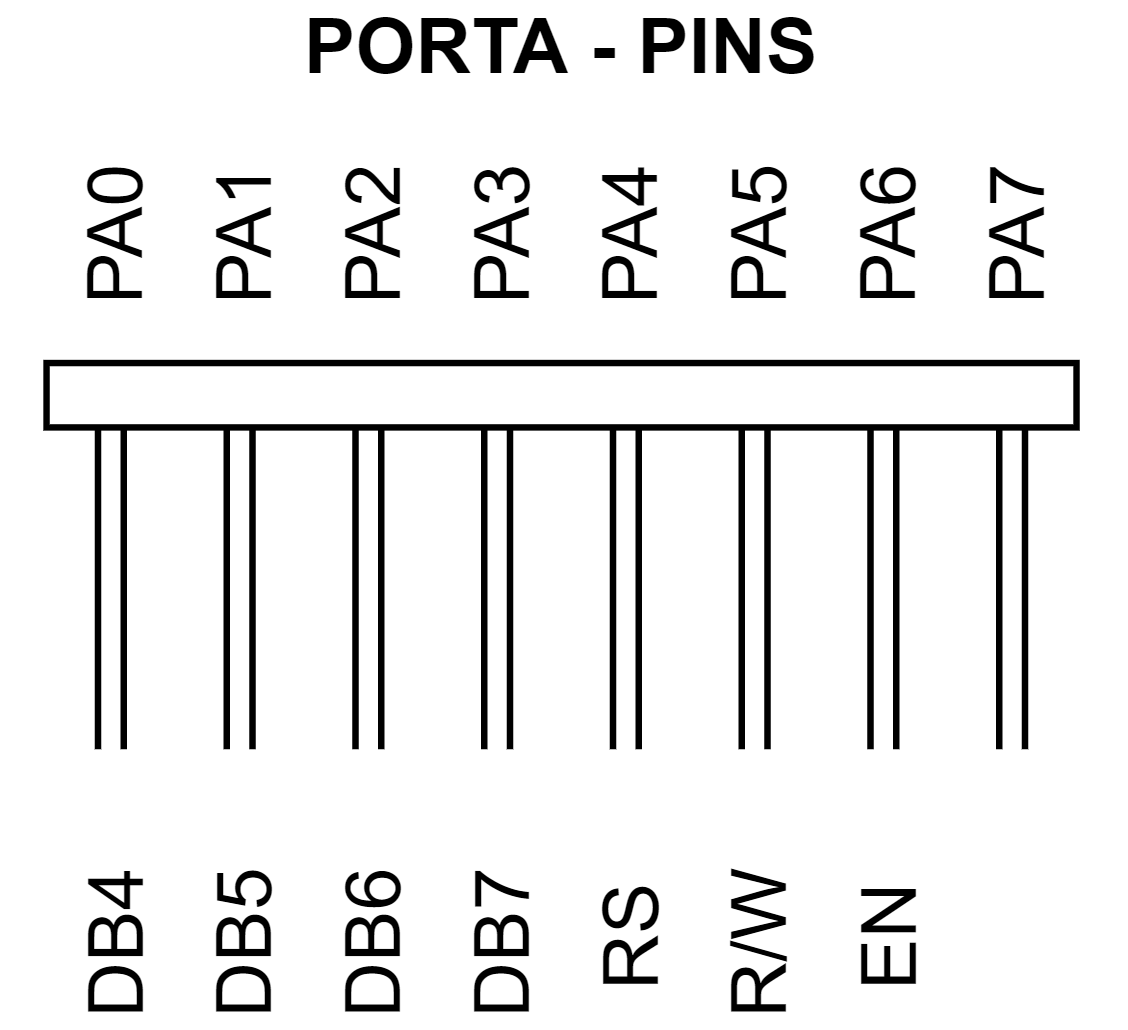
1: FÜR PORTD

RS = Register Select

RW = Read/Write (read = 1; write = 0)

E = Enable

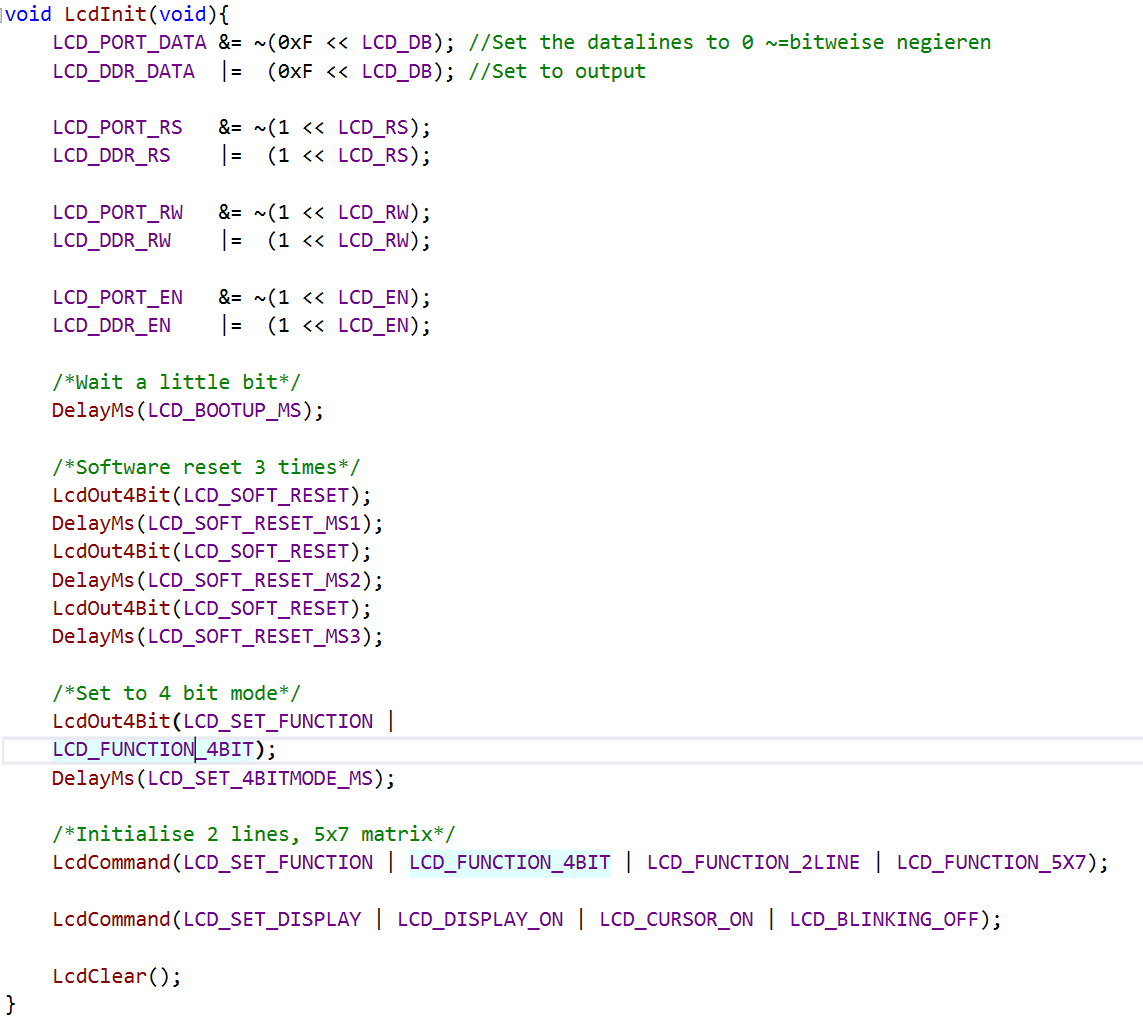
### 4-Bit Modus



3: 4-Bit Modus

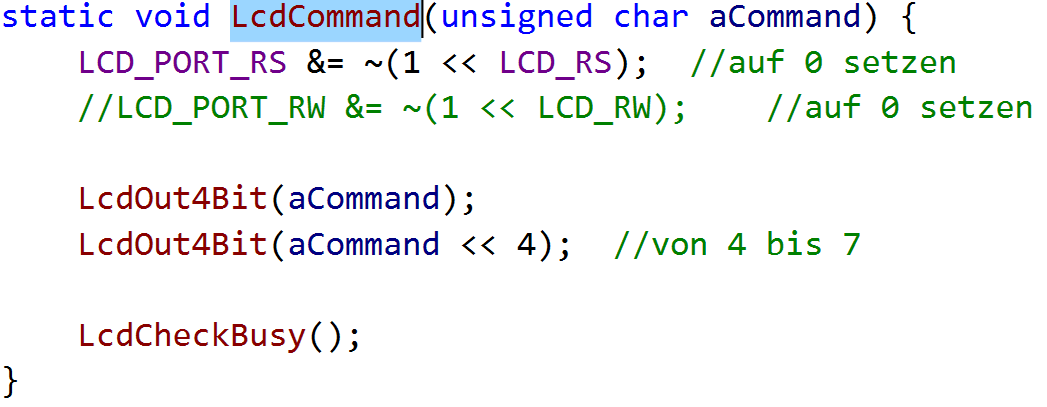
Im 4 Bit Modus werden nur 4 Bits für die Datenübertragung verwendet. Dies bedeutet, dass ein Display über einen PORT (z.B.: B oder D) angesteuert werden kann. Um mit 4 Bits zu schreiben sind die Leitungen RS, R/W und E wie unten erklärt zu setzen. Dies wird im Programm durch die „void LcdInit(void)“ erledigt.

Ist R/W auf Low, so wird geschrieben. Um 8 Bit/1 Byte zu schicken, müssen logischerweise 2 Mal 4 Bit gesendet werden. Anschließend ist die die R/W Leitung auf HIGH zu schalten und das Busy-Flag zu lesen. Ist Das Busy-Flag HIGH, so muss es erneut gelesen werden. Ist es auf Low, kann entweder ein weiteres Byte geschrieben oder das Data-Register gelesen werden.

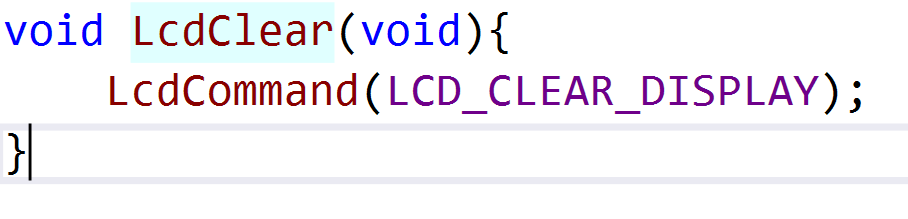


1. Alle Leitungen auf LOW setzen (danach delay, da LCD-Display auch mit BIOS starten muss)
2. Danach 3 Mal reseten, um sicher zu gehen, dass keine „alten“ Einstellungen beibehalten werden.
3. Danach dem Display sagen, dass man 1. Eine Funktion schreiben möchte und anschließend, dass man den 4 Bit Modus verwenden möchte. (Die LcdOut4Bit wird 2 Mal aufgerufen) (danach wieder Delay, da Befehle vom Display immer verarbeiten müssen)
4. Danach wird über LcdCommand(unsigned char aCommand) die LcdOut4Bit(aCommand); aufgerufen und es werden alle Einstellungen übertragen.

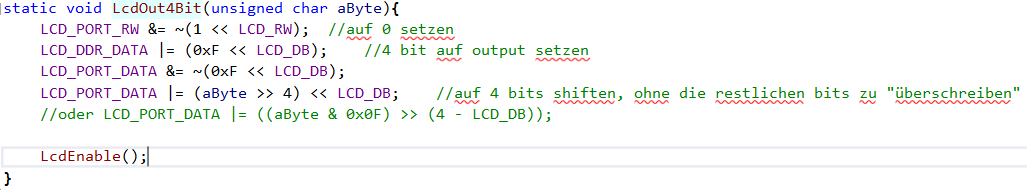
(mittels Variablendeklarationen (siehe: LcdOut4Bit 4.))



1. Anschließend wird das Display noch bereinigt:



### LcdOut4Bit

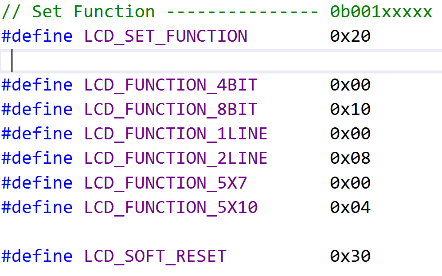
// 0xF = 1111

1. RW-Leitung auf Write (0)
2. Data Direction Register auf 0

(0 = Output -> damit alle 4 Pins von PORTA ausgeben können)

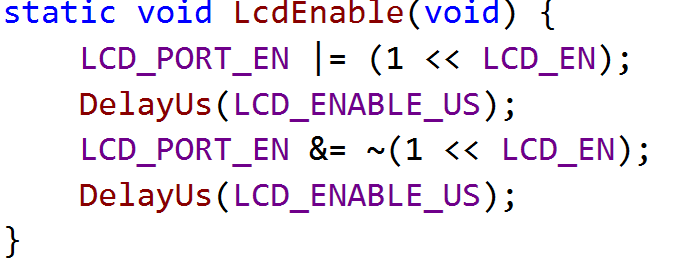
1. Mit aByte werden die Datenpins DB4-DB7 angesteuert

Durch die LcdConstants.h können Werte-Deklarationen verwendet werden wie:



Das LCD-Display weiß wie diese zu verwerten sind und macht den Rest.

1. LcdEnable ist eine Funktion, welche den EN-PIN/PORT 2-Mal aktiviert und wieder deaktiviert. Dies ist in den Impuls-Zeit-Diagrammen zu sehen.

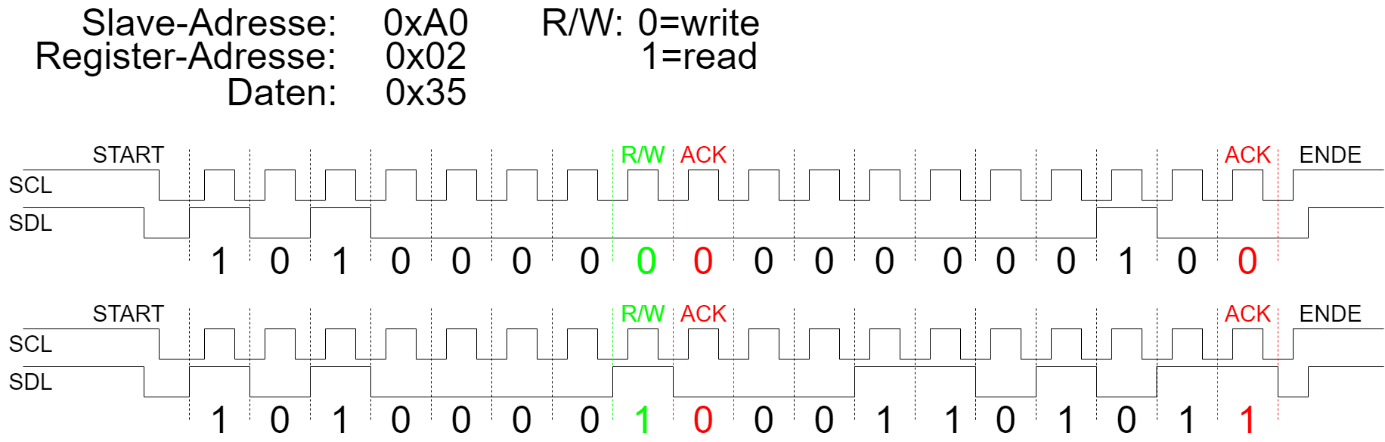


# **I2C**

### Lern einfach die I2C-Word

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### I2C Datentransfer

In der obigen Grafik ist der Verlauf der SCL (serial clock line) und der SDL (serial data line) zu sehen. Der Master will vom Slave aus einem Register lesen.

1. Start Condition
2. Adresse des Slaves (muss immer nach Start-Condition sein) = 7Bit
3. R/W = 0, da der Master dem Slave sagt, von wo er lesen möchte
4. ACK = 0, da Slave alle 8 Bit korrekt empfangen hat und die Leitung somit auf Low zieht
5. Register-Adresse: 0000 0010 = 2 = 0x02 (Adresse, von welche Master lesen will)
6. ACK = 0, da Slave alle 8 Bit korrekt empfangen hat und die Leitung somit auf Low zieht
7. Ende-Condition (Es kann nicht gleichzeitig gelesen oder geschrieben werden)

--------------------------------------------------------------------------------------------------------------------------------------

1. Start Condition (Neue Start-Condition für die Lese-Bitfolge)
2. Adresse des Slaves (muss immer nach Start-Condition sein) = 7Bit
3. R/W = 1, da der Master nun lesen möchte
4. ACK = 0, da Slave alle 8 Bit korrekt empfangen hat und die Leitung somit auf Low zieht
5. Daten: 0011 0101 = 53 = 0x35 (Daten aus Register z.B.: Uhrzeit, Amplitude…)
6. ACK = 1, weil:

*„Werden Daten vom Slave zum Master übertragen, so sendet der Master die Adresse mit gesetztem R/W Bit. Die nachfolgenden Bytes werden vom Slave auf den Bus gegeben. Will der Master die Übertragung beenden, so wird das letzte Byte nicht mit der ACK-Condition bestätigt. Der Slave muss daraufhin die Datenleitung freigeben, damit der Master die Stop-Condition erzeugen kann.“*

1. Ende-Condition