

**Folien zur Vorlesung
Grundlagen systemnahes Programmieren
Sommersemester 2016
(Teil 5)**

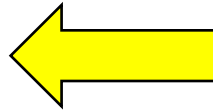
Prof. Dr. Franz Korf

Franz.Korf@haw-hamburg.de

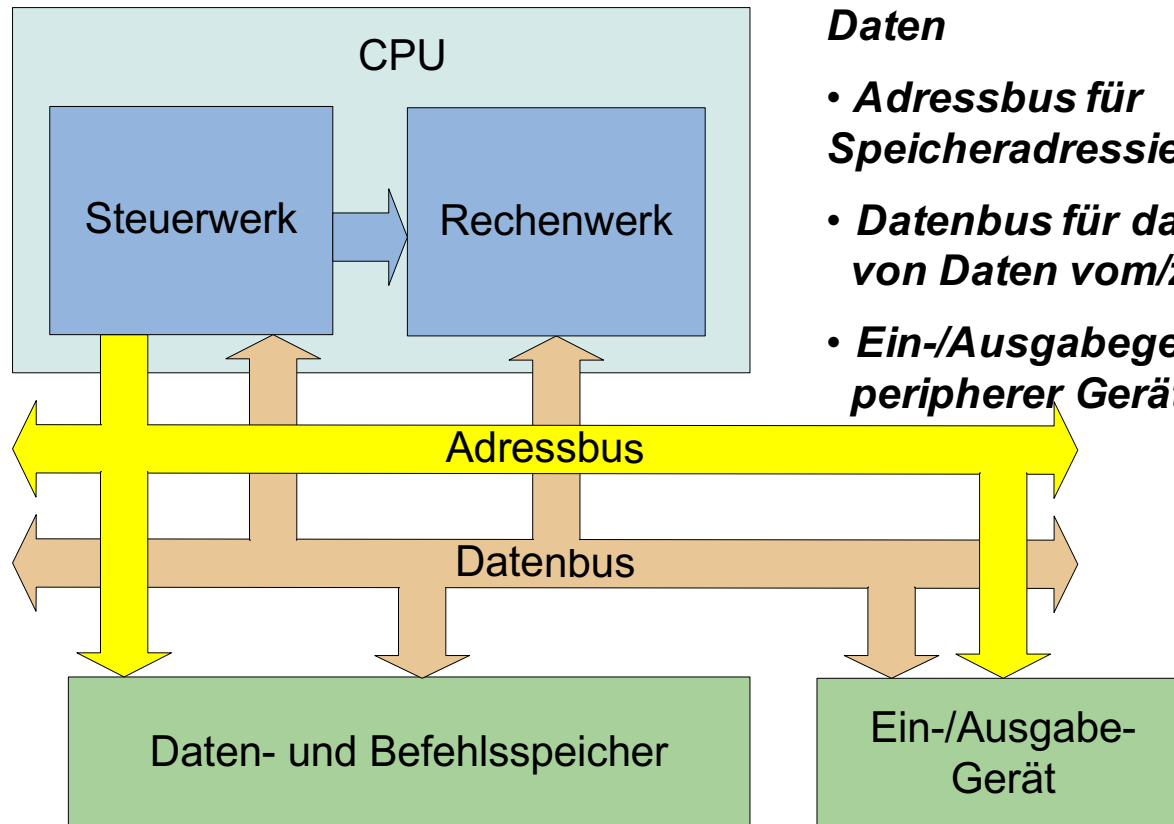
Kapitel 5: I/O Programmierung

Gliederung

- Einführung
- General Purpose Input/Output (am Beispiel von STM32F417ZG)
- Serielle Datenübertragung



Wiederholung: Von-Neumann-Architektur



- **ein Speicher für Befehle und Daten**
- **Adressbus für Speicheradressierung**
- **Datenbus für das Lesen/Schreiben von Daten vom/zum Speicher**
- **Ein-/Ausgabegerät zur Anbindung peripherer Geräte.**

Wie greift man auf Register eines externen Devices zu?

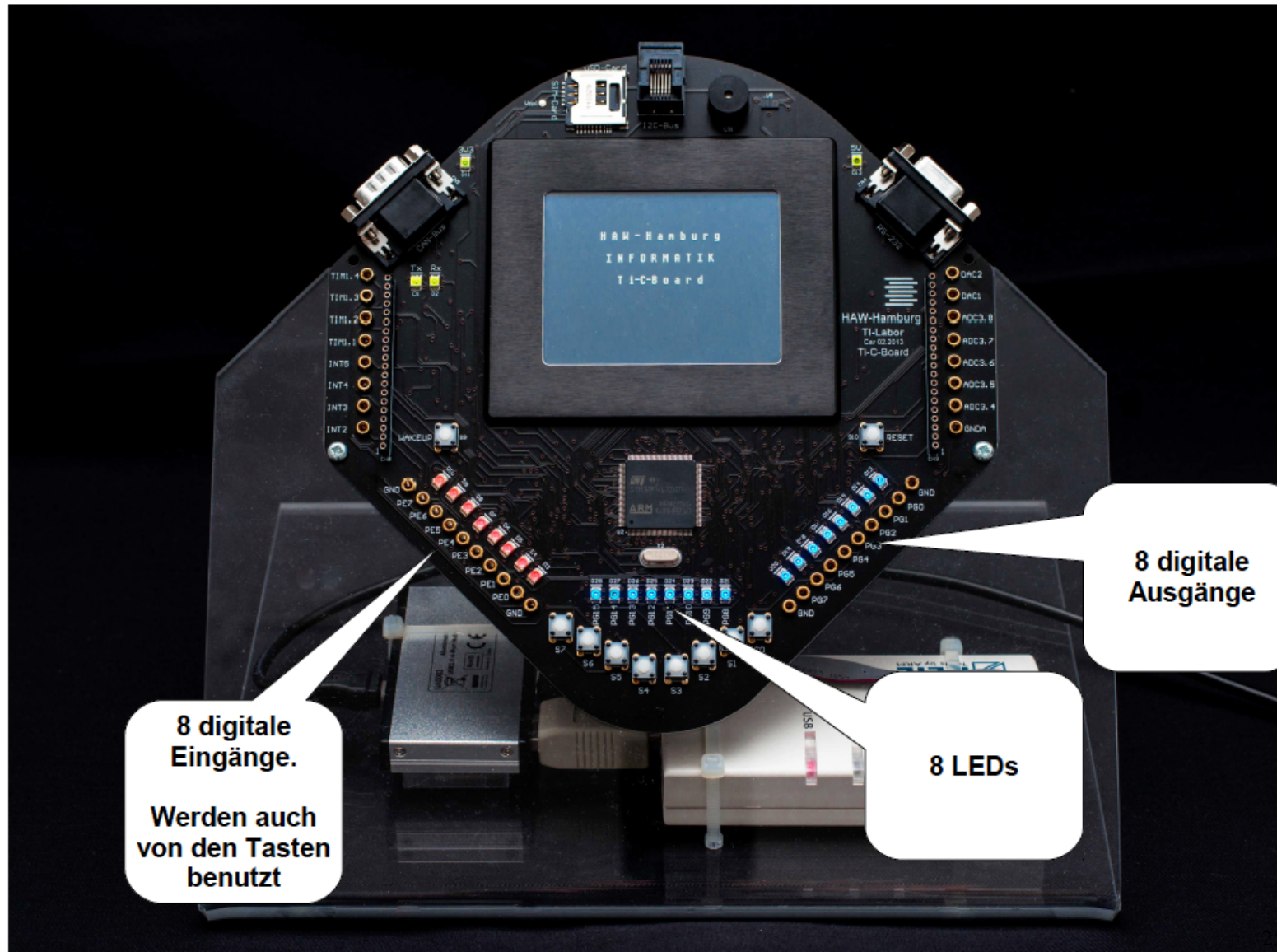
Alternative 1: Memory Mapped

- Die Register sind auf Hauptspeicheradressen abgebildet (mapped).
- Ein lesender / schreibender Zugriff auf diese Hauptspeicheradressen greift nicht auf den Speicher zu, sondern auf die entsprechenden Register des Devices.

Alternative 2: I/O Mapped (muss die CPU unterstützen, Intel tut dies)

- Es gibt einen weiteren Adressraum, so genannte **I/O Adressen**. Diese Adressen stehen in keiner Relation zu den Hauptspeicheradressen.
- Über spezielle Befehle (in, out Assembler Befehle) wird über I/O Adressen auf die Register eines Devices zugegriffen.

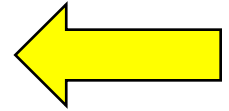
GPIO



Kapitel 5: I/O Programmierung

Gliederung

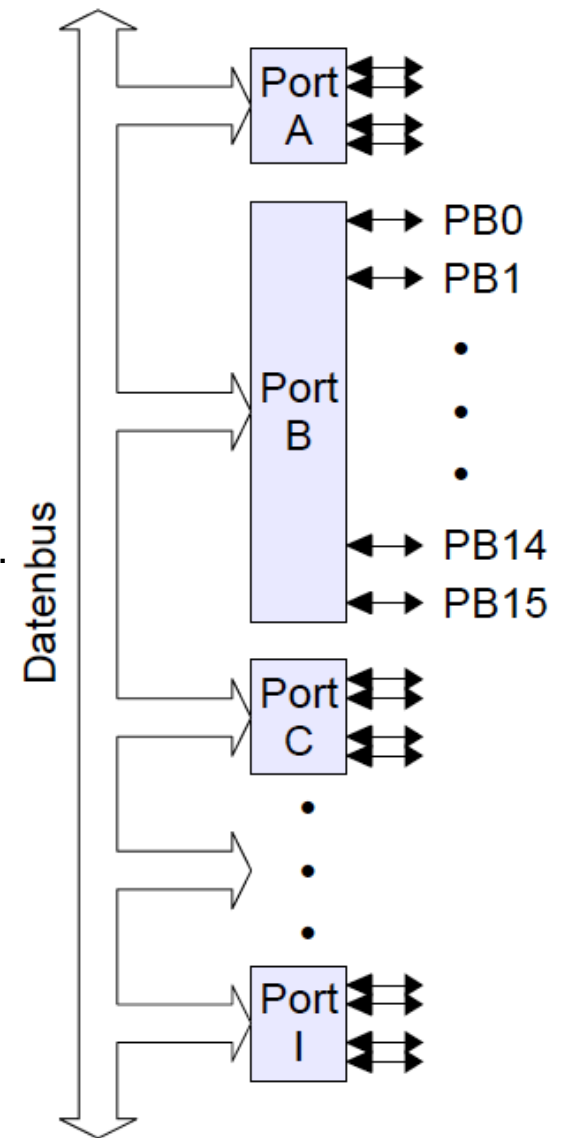
- Einführung
- General Purpose Input/Output (am Beispiel von STM32F417ZG)
- Serielle Datenübertragung



General Purpose Input/Output STM32F417ZG

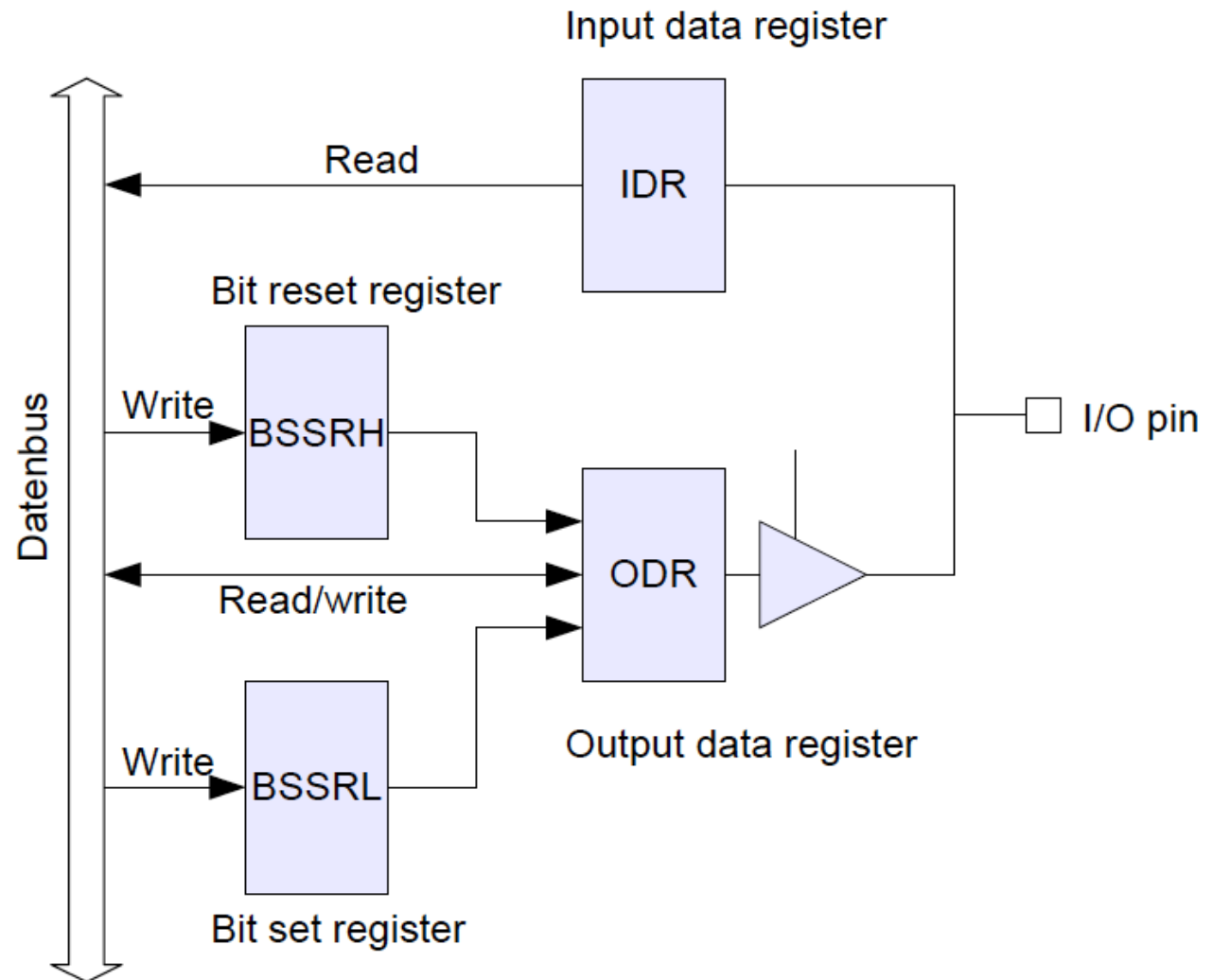
Eigenschaften:

- Neun identische 16-Bit Ports: PA bis PI.
- Jeder Anschluss kann einzeln für Ein- oder Ausgabe programmiert werden.
- Oft ist aber durch die externe Beschaltung die Datenrichtung vorgegeben.
TI-C-Board: Richtung vorgegeben durch `Init_TI_Board()`.
- Die meisten Ein- und Ausgabelleitungen können alternativ Spezialaufgaben übernehmen.
- Programmierung der Ports: Memory-Mapped.



General Purpose Input/Output: Schaltbild für einen Pin

- Schaltung ist pro Port 16 mal vorhanden.
- Alle Register: 16-Bit breit.

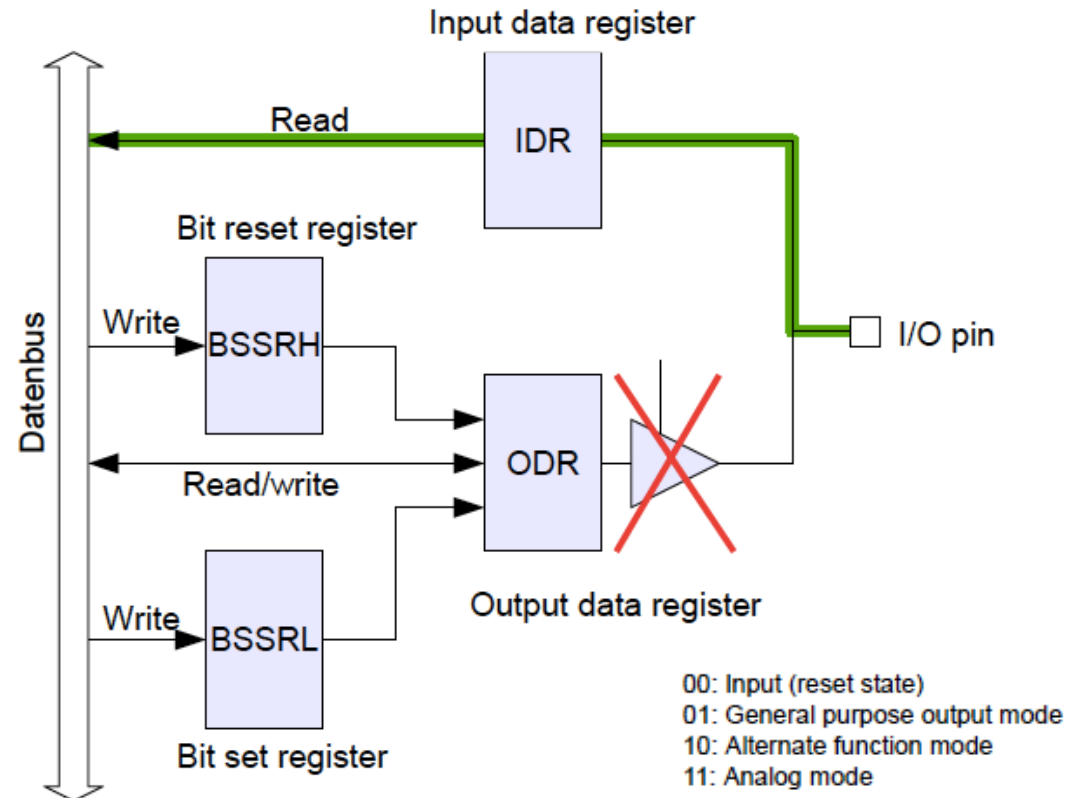


General Purpose Input/Output: Input

- Ausgabepuffer ist deaktiviert.
- Zustand des Pins kann über IDR gelesen werden.

```
// IDR auslesen:
daten = GPIOB->IDR;
```

- MODER Register:
Unschalten zwischen Input und Output mode

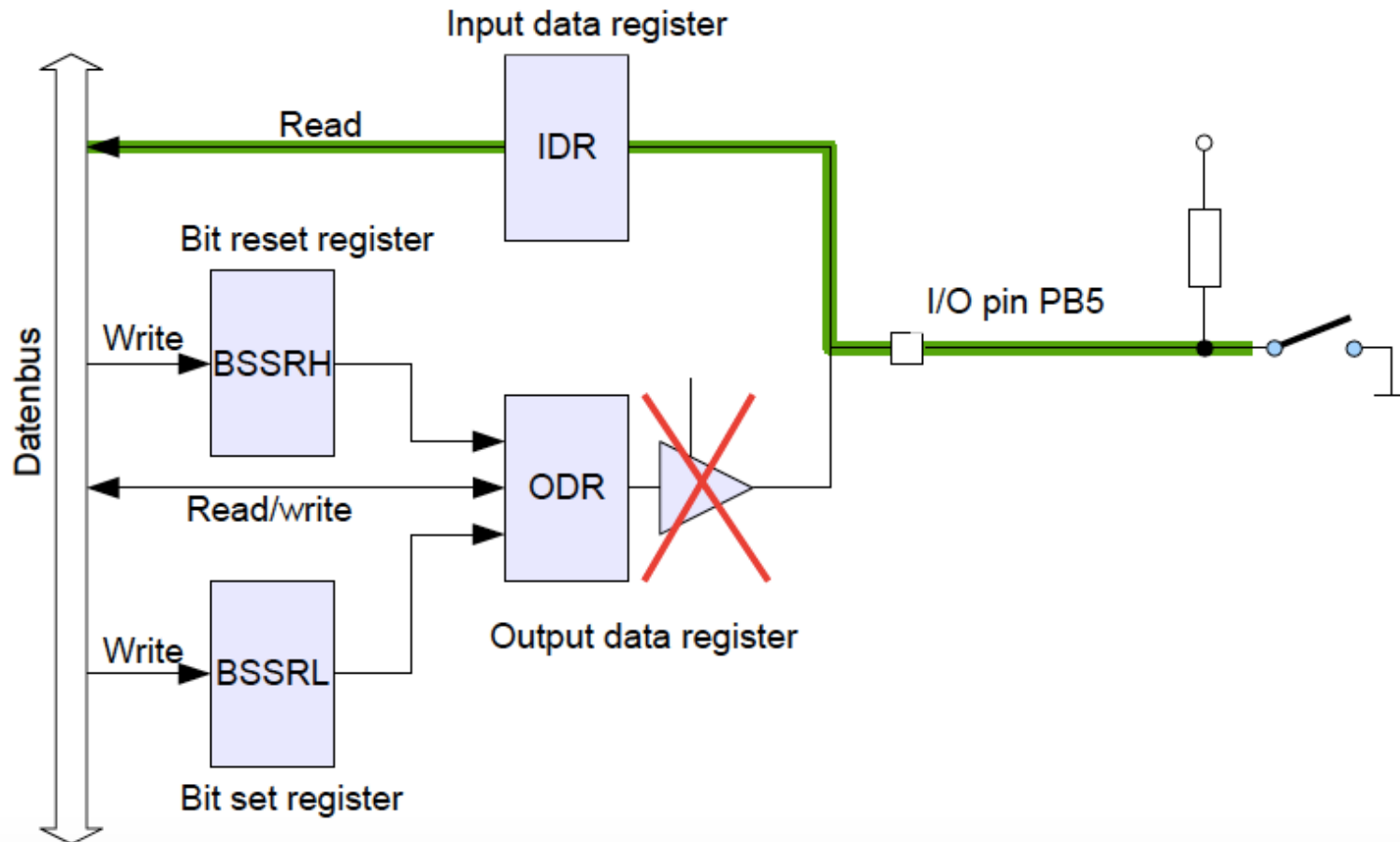


31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

General Purpose Input/Output: Lesen einzelner Bits

Beispiel: Abfrage einer Taste an PB5

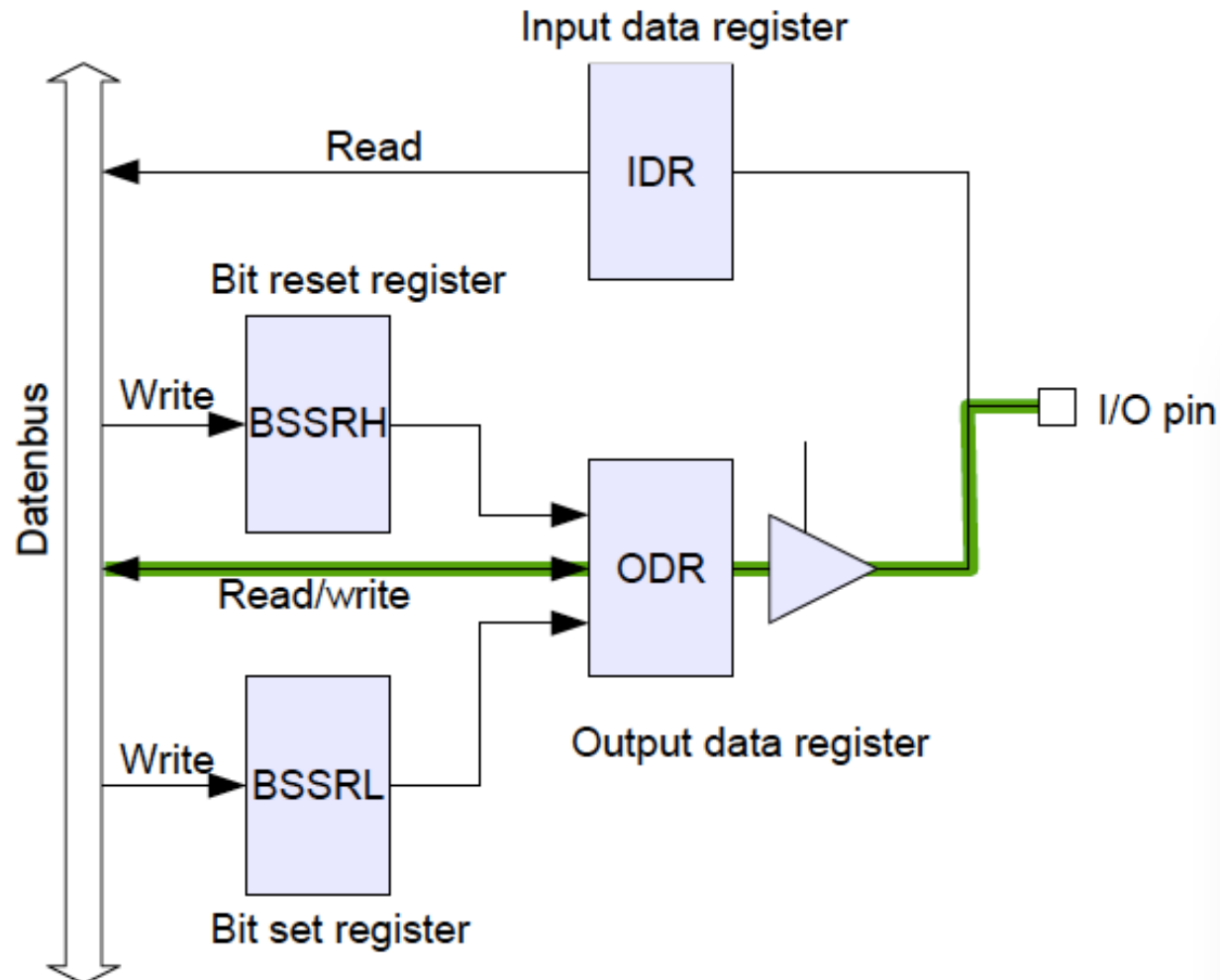
- Taste nicht gedrückt: PB5 = High (logisch 1)
- Taste gedrückt: PB5 = Low (logisch 0)



General Purpose Input/Output: Output

- Ausgabepuffer ist aktiviert.
- Zustand des Pins kann über ODR geschrieben werden.
- ODR lesen: Ergebnis sind die zuletzt in ODR geschriebenen Daten
- IDR lesen: Aktueller Zustand der I/O Pins.

```
// Alle Bits von ODR  
// setzen:  
GPIOB->ODR = datout;
```



General Purpose Input/Output: Ansteuern einzelner Bits

➤ Beispiel: Ansteuerung einer LED

PG14 = Low (logisch 0): LED leuchtet nicht.

PG14 = High (logisch 1): LED leuchtet.

➤ Ansteuerung von LED1 ohne Veränderung von LED2

Read - Modify - Write Zyklus!

➤ LED1 einschalten:

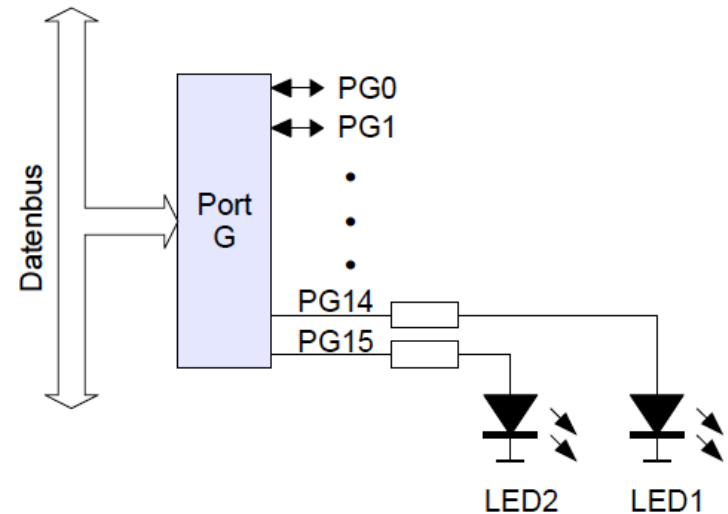
```
#define LED1PORT GPIOG->ODR
```

```
#define LED1BIT 14
```

```
LED1PORT = LED1PORT | (1<<LED1BIT)
```

➤ LED1 ausschalten:

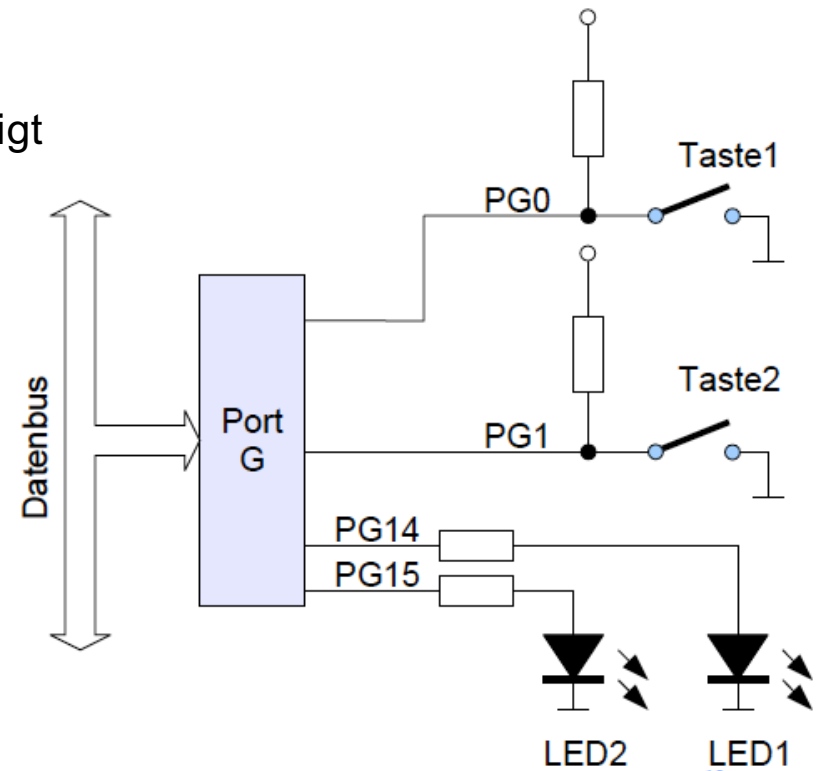
```
LED1PORT = LED1PORT & ~(1<<LED1BIT)
```



Kleine Übungsaufgabe zur Bitmanipulation

Erstellen Sie ein Programm mit folgendem Verhalten:

- LED1 nur dann an,
wenn Taste1 betätigt und Taste2 nicht betätigt
- LED2 nur dann an,
wenn Taste2 betätigt und Taste1 nicht betätigt



General Purpose I/O: STM32F417ZG Register

Input Data Register

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR																	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Output Data Register

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR																	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit Reset Register



Register	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BSRRH	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0: No action on the corresponding ODRx bit
 1: Resets the corresponding ODRx bit

Bit Set Register



Register	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BSRRL	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0: No action on the corresponding ODRx bit
 1: Sets the corresponding ODRx bit

General Purpose Input/Output: Bit Set und Bit Reset Register

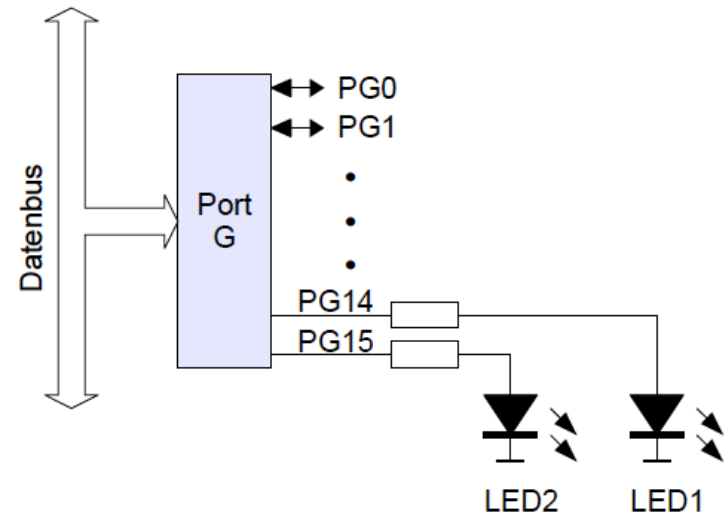
Ansteuerung einzelner Bits

➤ LED1 einschalten:

```
#define LED1PORTSET  GPIOG->BSRL  
#define LED1BIT  14  
LED1PORTSET = (1<<LED1BIT)
```

➤ LED1 ausschalten:

```
#define LED1PORTRESET GPIOG->BSRH  
LED1PORTSET = (1<<LED1BIT)
```



Kleine Übungsaufgabe zur Bitmanipulation

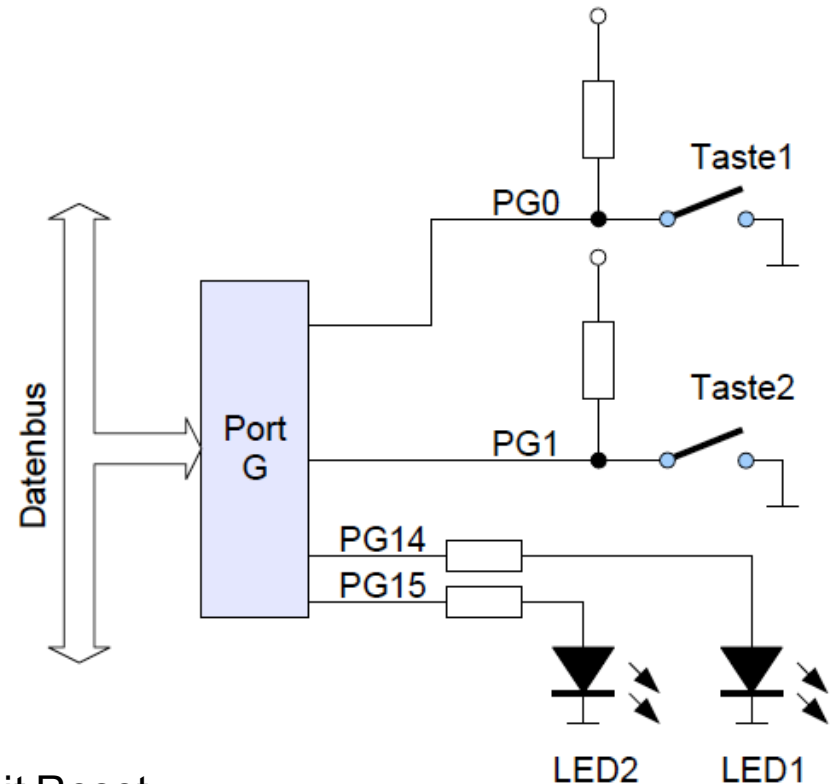
Erstellen Sie ein Unterprogramm zum Ansteuern der LEDs:

Es hat folgende Signature:

```
void setLed( int leds );
```

mit folgendem Verhalten:

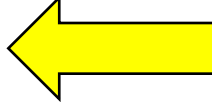
leds	LED1	LED2
0	aus	aus
1	ein	aus
2	aus	ein
3	ein	ein



- Version 1: Verwenden Sie die Bit Set und Bit Reset Register
- Version 1: Verwenden Sie die Bit Set und Bit Reset Register nicht.

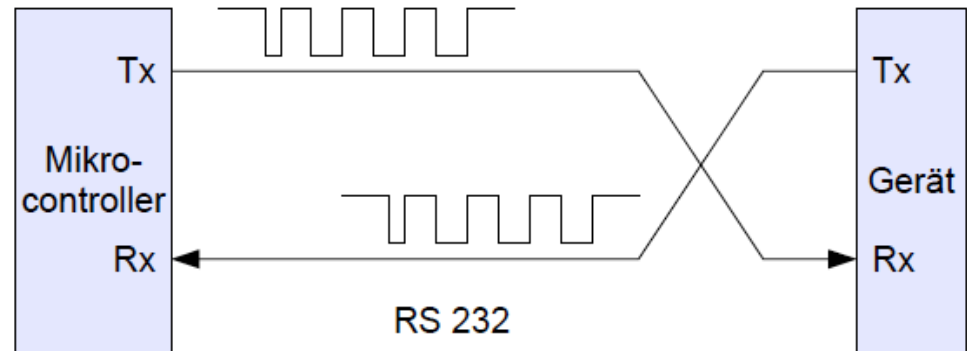
Kapitel 5: I/O Programmierung

Gliederung

- Einführung
- General Purpose Input/Output (am Beispiel von STM32F417ZG)
- Serielle Datenübertragung 

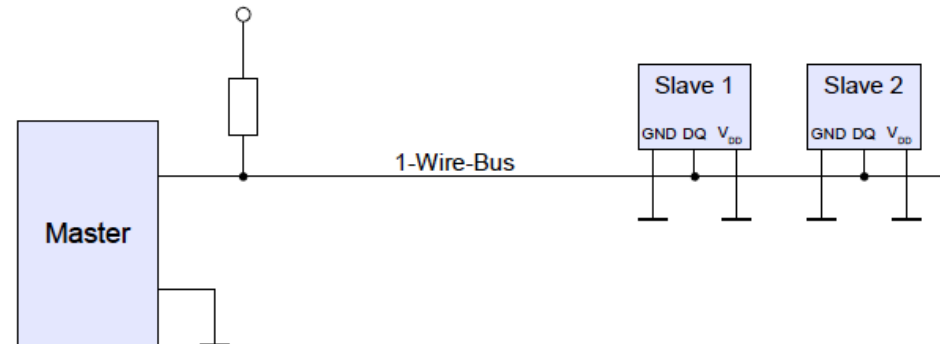
Serielle Datenübertragung

- Serielle Übertragung einzelner Bits
- Üblich: 2 Leitungen
 - Sendeleitung
 - Empfangsleitung
- Gleichzeitige Übertragung in beiden Richtungen möglich (Full Duplex)
- Übertragung beginnt mit Startbit
Danach folgen im festen Zeitabstand die n Bits des Datenwortes (ggf. + weitere Informationen)
- Zeitabstand ist durch Baudrate (Anzahl Symbole pro Sekunde) festgelegt.











1-Wire Bus

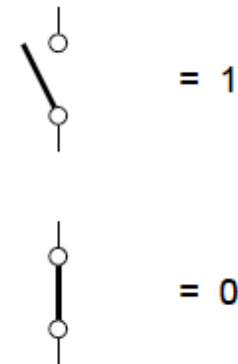
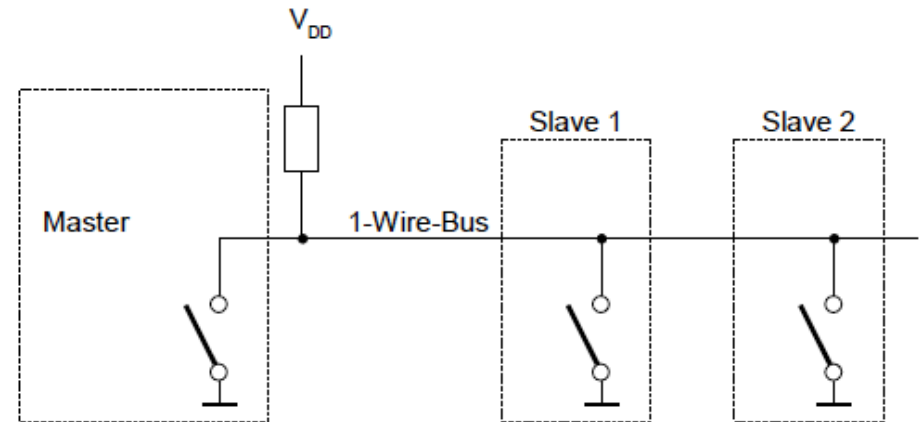
- Eine Signalleitung für Datenübertragung und Spannungsversorgung
- Datenaustausch erfolgt nur in eine Richtung zur Zeit (Half Duplex)
- Gesendete Daten werden von allen Geräten empfangen (Broadcast-Übertragung)
- Übertragung vom Master an einem bestimmten Slave erfordert Adressierung: Slaves müssen eindeutige Adressen haben.
- Zu jedem Zeitpunkt darf maximal nur ein Gerät Daten senden.
- Zugriffsprotokoll gemäß Master-Slave-Prinzip:
 - Master bestimmt, wer als nächstes Daten senden darf.
 - Ein Slave sendet nur Daten, wenn er vom Master aufgefordert wird.



1-Wire Bus: Wired-And Prinzip

Im Ruhezustand ist der Bus high

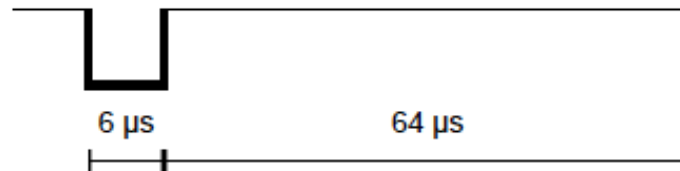
Master	Slave	Bus
		1
		0
		0
		0



1-Wire Bus: Lesen & Schreiben

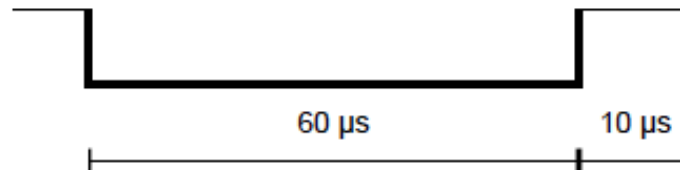
Schreibe „1“

Bus auf Low setzen
6 μ s warten
Bus freigeben
64 μ s warten



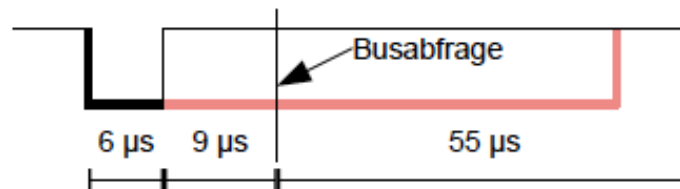
Schreibe „0“

Bus auf Low setzen
60 μ s warten
Bus freigeben
10 μ s warten



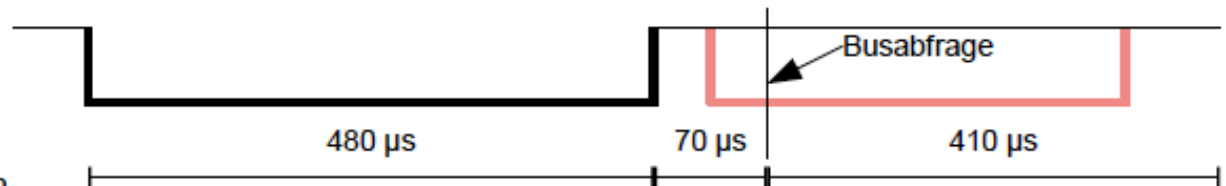
Lese Bit

Bus auf Low setzen
6 μ s warten
Bus freigeben
9 μ s warten
Bit vom Bus lesen
55 μ s warten



Reset

Bus auf Low setzen
480 μ s warten
Bus freigeben
70 μ s warten
Buszustand abfragen
410 μ s warten



IO-Port open-drain Mode

Schaltung:

- Die Leitung wird über einen (relativ hohen) Pull Widerstand auf High gezogen.
- Beim Senden einer 0 zieht ein Teilnehmer die Leitung auf 0.
- Beim Senden einer 1 setzt der Teilnehmer seinen Ausgang auf hochohmig

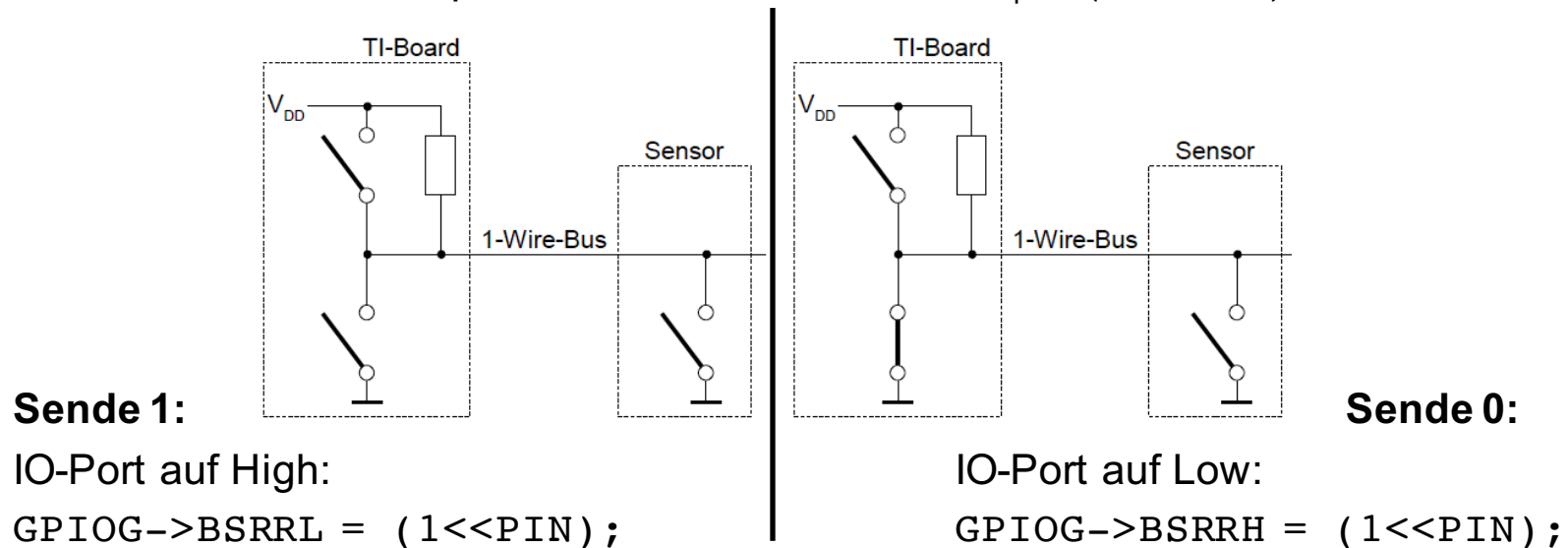
Vorteil

- Mehrere Teilnehmer sind an den Bus anschließen und können zeitgleich über Wired-And senden.

Nachteil:

- Eine Stromversorgung über die Datenleitung ist nicht möglich.

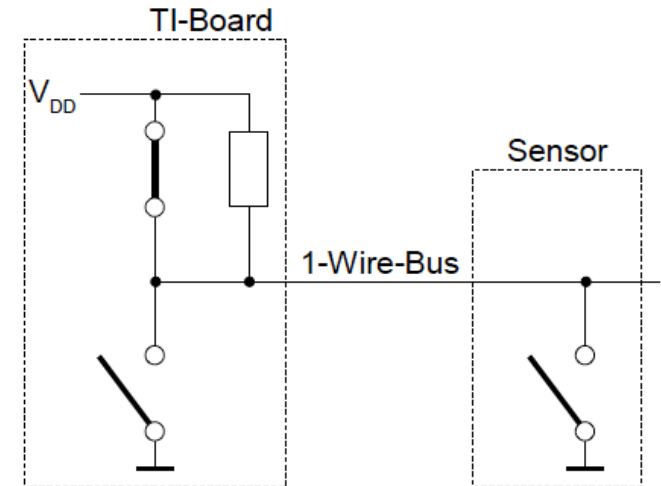
IO-Port auf open-drain: $\text{GPIOG} \rightarrow \text{OTYPER} = (1 \ll \text{PIN});$



IO-Port push-pull Mode

Schaltung:

- Die Leitung wird explizit getrieben.
- Beim Senden einer 0 zieht ein Teilnehmer die Leitung auf 0.
- Beim Senden einer 1 wird die Leitung mit VDD getrieben.



Vorteil

- Die Leitung kann (im begrenzten Maß) als Spannungsquelle dienen.

Nachteil:

- Kurzschluss, wenn mehrere Teilnehmer die Leitung mit unterschiedlichen Werten treiben.

IO-Port auf push-pull:

```
GPIOG->OTYPER &= ~(1<<PIN);
```

IO-Port auf High:

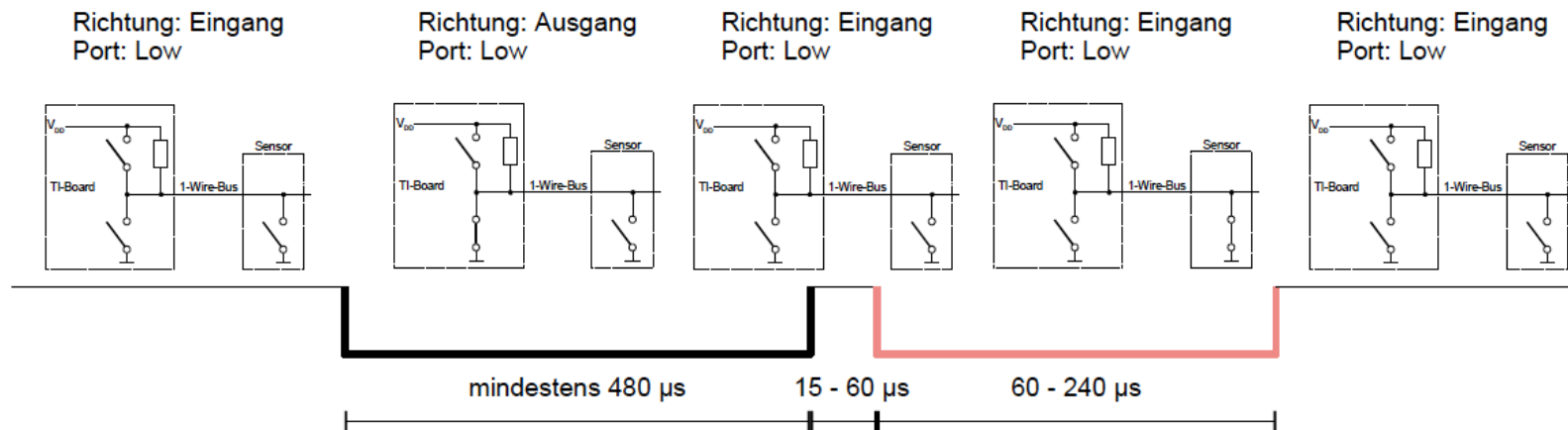
```
GPIOG->BSRRL = (1<<PIN);
```

Nur während der Spannungsmessung Bus auf push-pull mode stellen.

Während der Kommunikation Bus immer auf open-drain mode stellen.

Ablauf Reset

- Bus im open-drain Modus betreiben.



Auslesen des ROMs des Temperatursensors

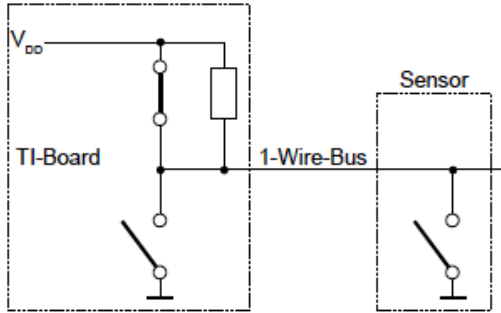
- Bus im open-drain Modus betreiben.

Reset
Sende Read ROM command (0x33)
Read Family Code (1 Byte)
Read Serial Number (6 Byte)
Read CRC (1 Byte)

64-Bit 'Registration' ROM Number					
MSB				LSB	
8-Bit CRC		48-Bit Serial Number		8-Bit Family Code	
MSB	LSB	MSB	LSB	MSB	LSB

- Funktioniert nur, wenn nur ein Sensor am Bus angeschlossen ist!
Sonst: Search ROM Command verwenden.

Durchführung einer Temperaturmessung

Wähle Slave aus			Führe Messung durch		Wähle Slave aus	Lese Ergebnis				
Reset	Sende Match ROM command (0x55)	Sende ROM code (8 Bytes)	Sende Convert T command (0x44)	<p>Versorge Sensor aus niederohmiger Spannungsquelle</p> <p>Richtung: Ausgang Port: High</p> 	Warte auf Beendigung der Messung (mindestens 750 msec)	Reset	Sende Match ROM command (0x55)	Sende ROM code (8 Bytes)	Sende Read Scratchpad command (0xBE)	Lese 9 Bytes (vollständiges Scratchpad inklusive Checksumme)
open-drain mode			push-pull mode		open-drain mode					
Reset	Skip ROM command (0xCC)						Reset	Skip ROM command (0xCC)		

Alternative, wenn nur ein Sensor am Bus angeschlossen ist

Bestimmung der Temperatursensoren am Bus

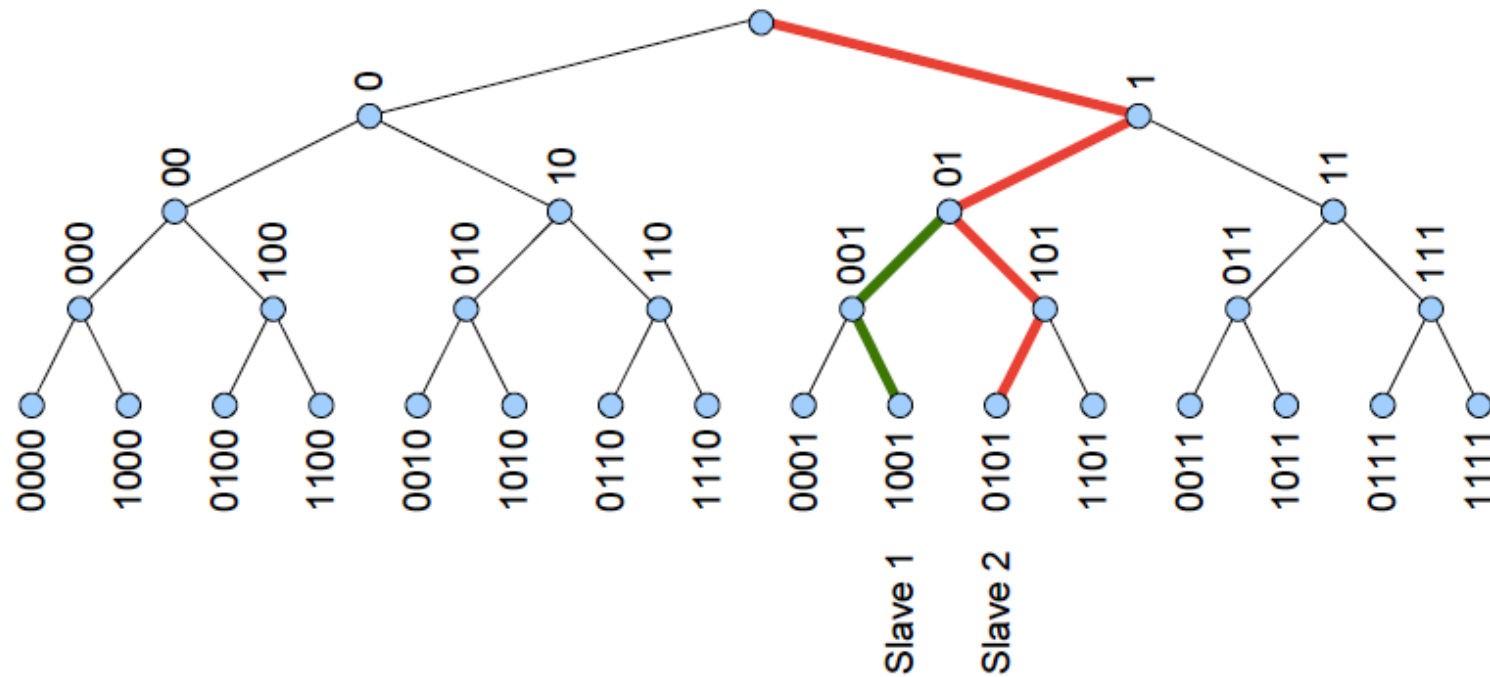
- Beispiel eines einfachen Low-Level Protokolls
- Ablauf:

Starte Algorithmus		Führe Algorithmus aus									
Reset	Sende Search ROM command (0xF0)	Empfange Bit 0	Empfange Komplement Bit 0	Sende Bit 0	Empfange Bit 1	Empfange Komplement Bit 1	Sende Bit 1	• • • • • • •			
									Empfange Bit 63	Empfange Komplement Bit 63	Sende Bit 63

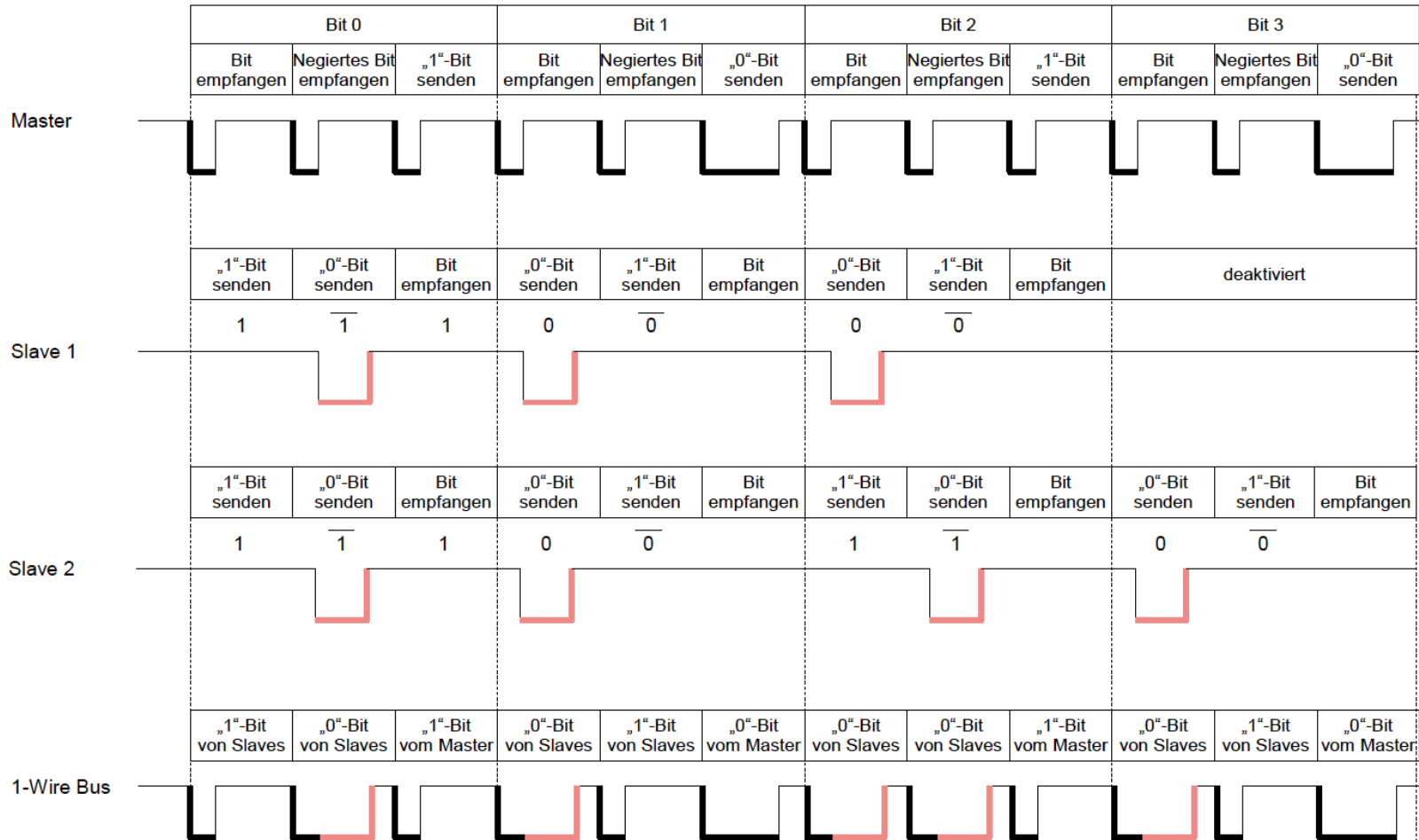
Bestimmung der Temperatursensoren am Bus

1. Schritt: Alle Sensoren senden gleichzeitig ihr erstes Bit
Signal auf dem Bus ist die UND-Verknüpfung dieser Bit
 2. Schritt: Alle Sensoren senden gleichzeitig das Inverse ihres ersten Bits
Signal auf dem Bus ist wiederum die UND-Verknüpfung dieser Bit
 3. Schritt: Analyse
 - Master hat die Bits '0' und '1' empfangen:
Das erste Bit aller Sensoren ist '0'
Master sendet eine '0' als Bestätigung
 - Master hat die Bits '1' und '0' empfangen:
Das erste Bit aller Sensoren ist '1'
Master sendet eine '1' als Bestätigung
 - Master hat die Bits '0' und '0' empfangen:
Das erste Bit der Sensoren ist nicht einheitlich
Master sendet ein Bit zurück zur Auswahl der Sensoren,
Mit denen die Suche fortgesetzt werden soll, Alle anderen Sensoren werden
inaktiv.
 - Master hat die Bits '1' und '1' empfangen:
Fehler ist aufgetreten, Verbindung zu den Sensoren ist unterbrochen.
- Wiederhole die Schritte 1 bis 3 mit den noch aktiven Sensoren solange, bis alle 64 Bits ausgewertet sind.

Bestimmung der Temperatursensoren am Bus: Beispiel



Bestimmung der Temperatursensoren am Bus: Beispiel



Zusammenfassung

