Fachhochschule Münster University of Applied Sciences

Labor für Halbleiterbaulelemente und

Übung Bussysteme SS2015
Tutor: Diplom Informatiker Gregor Rebel

Versuch 4

Inter Integrated Circuit - I²C Bus

I²C nutzt zwei bidirektionale Open-Drain-Leitungen, Serial Data Line (SDA) und Serial Clock Line (SCL), mit jeweils einem Pull-Up-Widerstand. Typische Spannungsbereiche sind 5V oder 3,3V. Das I²C Referenzdesign hat einen 7 Bit oder 10 Bit Adressraum, je nach verwendetem Bauteil. Typische Geschwindigkeiten auf dem Bus sind 100 kbit/s im Standard Mode und 10 kbit/s im low-speed Mode. Aktuelle Ausführungen des I²C Controllers können mehr Slaves ansprechen und erreichen höhere Geschwindigkeiten zum Beispiel 400 kbit/s im Fast--modus, 1 MBit/s im Fast-Mode-Plus oder auch Fm+ und 3,4 MBit/s im Highspeed-Modus. Diese Geschwindigkeiten finden in eingebetteten Systemen häufiger Verwendung als in handelsüblichen PCs.
Man beachte, dass die Bitraten für die Übertragung zwischen Master und Slave reine Symbolraten sind. Die Nutzdatenrate ist aufgrund eines Protokoll-Overheads geringer. Der Protokoll-Overhead beinhaltet die Slave-Adresse, ein Register innerhalb des Slave-Devices ebenso wie ACK/NACK-Bits für jedes Byte.
Die maximale Anzahl an Slaves ist durch den Adressraum begrenzt und ebenso durch die die maximale Bus-Kapazität von 400 pF, welche die Entfernungen für Kommunikation in der Praxis auf einige Meter beschränkt. Sollten mehrere Busteilnehmer oder größere Distanzen in der Praxis nötig werden, so kann es nötig sein mit zusätzlichen Bustreibern den Bus in kleinere Segmente zu unterteilen. Dies kann erforderlich sein um die maximale Kapazität der einzelnen Segmente unter dem erlaubten Wert von 400 pF zu halten.

I²C ist zweckmäßig wenn Einfachheit und niedrige Produktionskosten wichtiger als hohe Geschwindigkeiten sind. Eine spezielle Stärke von I²C ist die Möglichkeit mit einem Mikrocontroller ein Netzwerk aus Geräten mit gerade einmal zwei GPIO-Pins und Software zu kontrollieren. Die meisten anderen Technologien mit ähnlichen Anwendungen, wie beispielsweise der SPI-Bus, benötigen mehr Pins und Signale um mehrere Geräte miteinander zu verbinden.

Referenzdesign
Das Referenzdesign ist ein Bus mit einer Taktleitung (SCL) und einer Datenleitung (SDA) und einen 7-Bit-Adressraum.
Auf dem Bus existieren zwei Rollen:

- **Master**: Das Gerät welches den Takt generiert und die Kommunikation mit den Slaves initiiert.
- **Slave**: Das Gerät welches den Takt empfängt und reagiert, sobald es vom Master adressiert wird.

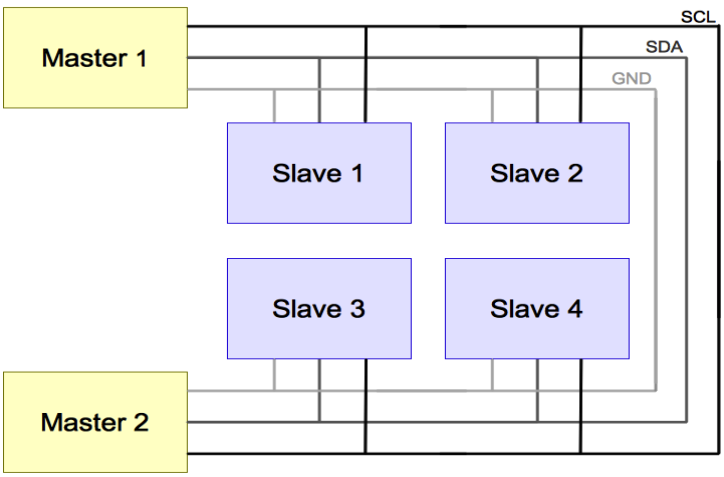

Bild 1: I2C Geräteschema

Der Bus ist ein Multi-Master-Bus, was bedeutet, dass eine beliebige Anzahl an Teilnehmern vorhanden sein kann. Zusätzlich dürfen Master und Slave auch zwischenzeitlich die Rollen tauschen.
Es gibt vier mögliche Modi für einen Busteilnehmer, jedoch nutzen die meisten nur eine Rolle und die zwei dazugehörigen Modi:

- Master sendet: Master schickt Daten zum Slave
- Master empfängt: Master erhält Daten vom Slave
- Slave sendet: Slave schickt Daten an den Master
- Slave empfängt: Slave erhält Daten vom Master

Der Master initiiert die Kommunikation durch das Senden des Start-Bits gefolgt durch die 7-Bit-Adresse des Slaves, welchen er anzusprechen wünscht. Danach erfolgt ein einzelnes Bit, welches anzeigt, ob geschrieben(0) oder gelesen(1) werden soll. Am Ende wird dann noch das Stopp-Bit gesendet.
Wenn ein Slave mit der Adresse auf dem Bus existiert, so quittiert er die Anfrage mit dem ACK-Bit (active low). Der Master beginnt dann Daten zu senden oder zu empfangen und der Slave befindet sich im jeweils komplementären Modus. Sowohl Adresse als auch Daten werden mit dem MSB zuerst gesendet. Wünscht der Master auf dem Slave zu schreiben, so sendet er wiederholt byteweise seine Daten was wiederum nach jedem Byte von dem Slave mit einem ACK bestätigt wird. Wünscht der Master zu lesen, so sendet der Slave byteweise seine Daten und der Master quittiert nach jedem Byte mit dem ACK-Bit. Das letzte Byte wird nicht mehr quittiert und der Slave hört auf zu senden. Der Master kann nun mit dem Stopp-Bit die Kommunikation beenden oder erneut eine Startbedingung generieren, den Bus übernehmen und eine weiter Übertragung beginnen.

Nachrichtenprotokolle
I²C definiert Grundtypen von Nachrichten, welche jeweils mit einem START beginnen und mit einem STOP enden:

- Einfache Nachricht: Master schreibt auf Slave.
- Einfache Nachricht: Master liest vom Slave.
- Kombinierte Nachricht: Master liest oder schreibt eine oder mehrere Nachrichten an einen oder mehrere Slaves.

In einer kombinierten Nachricht beginnt jeder Zugriff mit einem START und wird gefolgt von der Adresse des Slave. Die Start-Bits nach dem ersten Start-Bit werden auch repeated Start-Bits genannt und folgen nicht auf ein Stopp-Bit. Auf diese weise weiß der Slave, dass der nächste Transfer Bestandteil der gleichen Nachricht ist.

Reine I²C Systeme unterstützen arbiträre Nachrichtenstrukturen. SMBus ist beschränkt auf neun diese Strukturen, wie beispielsweise lese Wort N und schreibe Wort N, inklusive eines Slaves. PMBus erweitert SMBus mit einem Gruppenprotokoll, was erlaubt mehrere SMBus ähnliche Nachrichten in einer kombinierten Nachricht zu schicken. Das terminierende STOP zeigt an, wann diese gruppierten Nachrichten wirksam werden. Beispielsweise schaltet eine PMBus-Anweisung mehrere Lampen an (mit unterschiedlichen Adressen pro Lampe), und ihre neuen Zustände werden zur gleichen Zeit wirksam: wenn sie das STOP-Signal empfangen.
In der Praxis verwirklichen die meisten Slaves ein Nachfrage/Antwort Modell, in welchem ein oder mehrere Bytes nach einem Schreib-Bit als Befehl oder Adresse behandelt werden. Diese Bytes bestimmen wie wie geschriebene Bytes behandelt werden oder wie wie der Slave auf Leseanfragen zu regieren hat. Beispielsweise kann das erste Byte nach der Übertragung ein Register im Slave repräsentieren, welches gelesen oder beschrieben werden soll. Die meisten SMBus-Operationen beinhalten single Byte commands.

Physische Schicht
Auf der physischen Schicht sind beide Leitungen, SCL und SDA, Open-Drain getrieben, weswegen Pull-Up-Widerstände benötigt werden. Wird die Leitung auf Masse gezogen, so gilt dies als logische Null. Liegt die Leitung auf VCC so ist dies logisch Eins. High-Speed-Systeme nutzen eine Stromquelle als Pull-Up, zumindest auf SCL, um geringer An- und Abstiegszeiten zu realisieren sowie um eine höhere Bus-Kapazität zu erreichen.
Da die logische Null durch Masse repräsentiert wird, folgt daraus, dass Null der dominante Zustand ist. Ein Busteilnehmer der eine Leitung auf Masse zieht setzt sich also gegen einen anderen durch. Wird dieses Prinzip auf SCL angewandt, so redet man von Clock stretching. Dies gibt dem Slave die Möglichkeit die Clock auf 0 zu halten. Wird es auf SDA angewandt, so spricht man von Arbitrierung und es stellt sicher, dass nur ein Teilnehmer auf der Leitung spricht.
Im Ruhezustand sind beide Leitungen HIGH. Um eine Übertragung zu starten wird SDA herunter gezogen, während SCL auf HIGH bleibt. Anschließend wird auch SCL auf LOW gezogen. Da sich SDA sonst nur ändert, wenn SCL LOW ist, ist dieser Zustand einzigartig. Dies ist die Startbedingung. Ändert sich SDA von LOW auf HIGH während SCL HIGH ist so ist dies die Stoppbedingung. Nach dem START wird ein Bit übertragen indem SCL einmal von LOW über HIGH zurück auf LOW wechselt, während SDA den gewünschten logisch Pegel hat. SDA wird also nur verändert während SCL LOW ist. Auf SCL LOW findet also die Pegeländerung seitens des Senders statt, während auf SCL HIGH die Auswertung auf Seite des Empfängers stattfindet. So wird ein Bit übertragen.

Nach jeweils 8 Datenbits in eine Richtung wird das ACK-Bit in die andere Richtung gesendet. Es werden also für ein Bit die Rollen getauscht und der vormalige Empfänger wird zum Sender einer einzelnen 0 (ACK). Seht der Sender eine 1(NACK), so kann dies folgendes bedeuten:

- Master zum Slave: Der Slave kann keine Daten akzeptieren. Es ist kein Slave vorhanden. Anweisung nicht verstanden. Es können keine Daten mehr empfangen werden.
- Slave zum Master: Der Master will die Übertragung nach diesem Byte beenden.

Nach dem ACK hat der Master drei Möglichkeiten:

- Eine weitere Übertragung empfangen/senden.
- STOP generieren.
- Repeated Start durchführen.

Clockstretching auf SCL

Einer der Vorteile des I²C Protokolls ist das Clockstretching. Ein Slave hat hierbei die Möglichkeit SCL auf LOW zu ziehen und so die Taktgenerierung zu unterbrechen. Er kann somit anzeigen, dass er noch nicht bereit ist mehr Daten zu verarbeiten. Da der Master nur nach einem Takt den Pegel auf SDA ändern darf ist dieser somit gezwungen zu warten bis SCL wieder freigegeben wird. Das gleiche passierte, wenn ein langsamer Master gleichzeitig versucht auf den Bus zu schreiben.
Der Master ist berechtigt SCL so lange auf LOW zu halten wie es ihm beliebt. Der Begriff Clockstretching wird in der Regel nur verwendet, wenn Slaves so verfahren. Dies ist auch die einzige Form in der ein Slave aktiv den Zustand auf SCL beeinflusst.
Um einen minimalen Datendurchsatz zu garantieren limitiert der SMBus wie lange die Clock gestretched werden darf. Master und Slaves, welche dieses Limit verletzen, können dein Bus nicht länger blockieren. Dies ist bei reinen I²C Systemen nicht garantiert.

Arbitrierung mittels SDA

Jeder Master beobachtet den Zustand des Busses nach START- und STOPs und beginnt keine Übertragung solange der Bus belegt ist. Nichtsdestotrotz kann es vorkommen, dass zwei Master gleichzeitig beginne auf dem Bus zu sprechen. Dies ist der Fall in dem Arbitrierung greift. I²C hat ein deterministisches Arbitrierungsprinzip. Jeder Sender beobachtet den Pegel von SDA und vergleicht ihn mit dem, welchen er gerade auf die Leitung legt. Der Sender bei dem sich beide Pegel unterscheiden hat den Vergleich verloren und hört auf auf dem Bus zu agieren. Dies ist möglich dank des Prinzips der dominanten NULL, da diese durch Masse repräsentiert wird. Sendet also ein Teilnehmer eine EINS aber sieht auf dem Bus eine NULL, so weiß er, dass ein anderer Teilnehmer sendet. Es verliert also der Teilnehmer der den Unterschied zuerst bemerkt. Handelt es sich um einen Master stoppt er die Taktgenerierung und wartet auf ein STOP. Danach kann er versuchen erneut zu senden. Da der Gewinner des Vergleichs keinen Unterschied bemerkt hat setzt er die Übertragung fort und beendet seine Übertragung auf gewohntem Wege. Versuchen also zwei Master unterschiedliche Slaves anzusprechen, so gewinnt der Master, welcher den Slave mit der niedrigeren Adresse anspricht. Hier findet die Arbitrierung in der Adressphase statt. Sprechen zwei Master den selben Slave an, so verlängert sich die Arbitrierung bis in die Datenphase. Arbitrierung findet sehr selten statt, aber sie ist nötig für einen reibungslosen Verlauf mit mehreren Mastern.

Aufgabe 1

Für diese Aufgabe kann ein beliebiges Prototypboard verwendet werden. Es soll die erste I²C-Schnittstelle auf dem aktuellen Board konfiguriert, initialisiert und zum Senden von Nachrichten verwendet werden. Die versendeten Nachrichten werden dann mit einem Oszilloskop dargestellt.

High-Level Treiber ttc_spi

Die generelle Verwendung von ttc_-Treibern wurde bereits in den vorigen Praktika vorgestellt. Lesen Sie diesen Abschnitt bei Bedarf nach.

Schritte

1. Erstellen Sie ein neues Projekt mit dem Namen BS_Gruppe1_Versuch4 bzw. BS_Gruppe2_Versuch4.
2. Ändern Sie die Konfiguration in activate_project.sh
   1. Aktivieren Sie entsprechend Prototypboard + Programmer
   2. Aktivieren Sie **500_ttc_i2c, 500_ttc_gpio**
   3. Deaktivieren Sie 600_example_leds
3. Fügen Sie eine neue Quelltextdatei namens test_i2c mittels _/source.pl zum Projekt hinzu
4. Legen Sie die Funktion **void test_i2c_start()** in test_spi.c/.h an und rufen Sie diese von taskMain() aus 1x auf. Hierzu können Sie wieder _/source.pl nutzen um das Hinzufügen von Funktionen zu vereinfachen: _/source.pl function public 'void start()' test_i2c
5. Implementieren Sie test_i2c_start()
   1. Laden Sie die Konfiguration der ersten SPI-Schnittstelle.
   2. Konfigurieren als **Master** für 7-**Bit Adressen** im **Standard Mode** und eine Datenrate von **200 kBit/s**
   3. Initialisieren Sie die konfigurierte Schnittstelle
   4. Senden Sie den Text „SOS“ im Abstand von 500 ms
   5. Verwenden Sie die erste LED indem Sie deren Zustand bei jedem Sendevorgang invertieren
6. Stellen Sie die Signale von SCL und SDA mit einem Digitaloszilloskop dar
7. Zeichnen Sie die Signale in den Abbildungen 1 und 2 nach

Abbildung 1: SCL

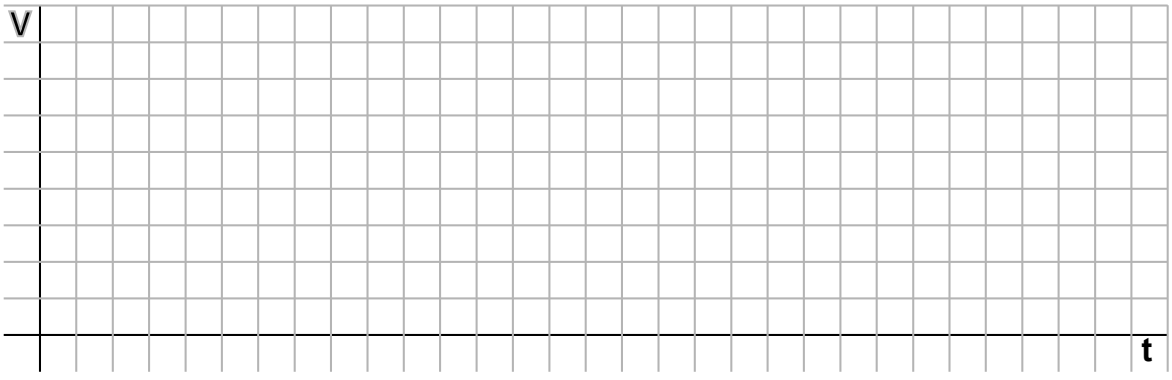

Abbildung 2: SDA

Aufgabe 2

Für diese Aufgabe müssen Sie sich mit einer Nachbargruppe zusammentun und zwei Protoboards per Kabel verbinden. Schließen Sie dazu beide Boards am selben Rechner an. Dadurch wird vermieden, dass die Ausgangstreiber durchbrennen. Diese sind nicht gegen Überspannungen und unterschiedliche Massepotentiale geschützt!
Verfolgen Sie die Datenübertragung auf der SDA und SCL Leitung mit einem Oszilloskop.

Gruppe A

1. Initialisiert den I²C wie zuvor beschrieben.
2. Initialisiert zusätzlich den ersten UART mit 115200 bit/s 8n1
3. Initialisiert zu Beginn eine Variable u8_t Data als 0.
4. In einer Endlosschleife
    1. Empfängt ein Byte vom Slave #127 und schreibt dessen Wert in Data
    2. Sendet den Integerwert von Data als ASCII-Zahl + „\n“ über die serielle Schnittstelle
5. Zeigt die empfangenen Werte mithilfe von grTerminal an

Gruppe B

1. Initialisiert den I²C als Slave mit der Adresse 127.
2. Initialisiert zu Beginn eine Variable u8_t Data als 0.
3. In einer Endlosschleife
    1. Wartet auf eine eingehende Leseanfrage
    2. Zählt Data um eins hoch
    3. Sendet Data