Stoppuhr
&
Wecker

mit RTC-Chip

I^2C

Projektarbeit Microcontoller

Hak Heang Keo

Johann Gysin

Inhalt

Aufgabenstellung

1.1 Erhaltene Aufgabenstellung

Microcontroller: Projektarbeiten **TSU**

Aufgabe 1: Stoppuhr & Wecker

Schwerpunkte: I2C Bus serielles Auslesen eines RTC, Programmiertechnik

Auf dem H8 Board ist eine Epson RealTImeClock eingebaut, welche über eine I2C SS mit dem Prozessor verbunden ist. Entwickeln Sie eine I2C Bus –SW (ev. mit der Gruppe 2) und nutzen Sie diesen RTC einerseits, um eine Stoppuhr zu realisieren, andererseits um eine kontinuierliche Zeitanzeige auf dem Display zu ermöglichen.

Die ganze Stoppuhr soll in zwei Modi betrieben werden können:

- normale Zeitanzeige mit Datum und Uhrzeit
  In diesem Modus soll es möglich sein, über die vorhandenen Schalter die Uhr neu zu stellen sowie die Eingabe einer Weckzeit.
  Bei Erreichen der Weckzeit soll (mangels Lautsprecher) auf den LED's ein Blinken erfolgen
- Stoppuhr-Betrieb
  Mit den vorhandenen Schalter soll die Stoppuhr gestartet und gestoppt werden können, zudem soll die Möglichkeit einer Zwischenzeitanzeige vorhanden sein.

Unterlagen: Datenblatt-Kopien EPSON RTC8583

1.2 Präzisierung / Aufgabenstellung

Zeit/Datum anzeigen

Im Ruhezustand wird die aktuelle Uhrzeit und das Datum angezeigt.

Zeit/Datum einstellen

Wenn Schalter 7 in die obere Position (Hi) geschaltet wird, kann die Uhrzeit editiert werden. Mit den Schaltern 5 (links) und 6 (rechts) kann der Editiercursor gesetzt werden und mit den Schaltern 0 (down) und 1 (up) der Wert verändert werden.
Um die Einstellungen zu übernehmen muss der Schalter 7 wieder in die untere Position gebracht werden.

Weckzeit einstellen

Wie Zeit/Datums Einstellung.

Weckfunktion aktivieren

Schalter 4 (Hi) aktiviert die Weckfunktion.

Stoppuhrbetrieb

Schalter 3 (Hi) aktiviert den Stoppuhrbetrieb.
Ist dieser Modus aktiv, kann mit Schalter 0 gestartet/gestoppt werden und mit Schalter 1 die Zeit auf 0 zurückgesetzt werden.

Optionen

Weckzeit im SRAM vom RTC-Chip speichern.

Teilaufgaben / Strukturierung

2.1 Aufgabenaufteilung

Diese Projektarbeit lässt sich in zwei wesentliche Teilaufgaben aufteilen.

1. I^2C-Bus Ansteuerung

2. Eigentliche Programmfunktionalität

Aufgabe 1 übernimmt Johann Gysin und um die Aufgabe 2 kümmert sich Hak Heang Keo. Die beiden Aufgabenteile werden über eine Softwareschnittstelle miteinander verbunden.

2.2 Softwareschnittstellen

Die Schnittstelle zwischen den zwei Teilaufgaben in dieser Projektarbeit bilden zwei Funktionen, welche die I^2C-Bus Ansteuerung bereitstellt.

char GetI2CByte(char SlaveAddr, char Address);

Diese Funktion holt ein Byte von einem Slave auf dem I^2C-Bus. Als Parameter müssen die Slave-Adresse und die Speicheradresse übergeben werden.

void PutI2CByte(char SlaveAddr, char Address, char Data);

Mit dieser Funktion kann einem Slave auf dem I^2C-Bus ein Byte gesendet werden. Es müssen die Slave-Adresse, Speicheradresse sowie das zu sendende Byte als Parameter übergeben werden.

3.1 Einleitung

Der I²C-Bus besteht hardwaremässig aus zwei Leitungen; einer Datenleitung und einer Clockleitung. Am Bus können ein oder mehrere Mastergeräte vorhanden sein und ein oder mehrere Slavegeräte. Dies wird ermöglicht, indem sich im Ruhezustand alle Geräte auf den beiden Leitungen hochohmig verhalten.
Mit Pull-Up Widerständen werden die Daten- und Clockleitung auf V_{CC}-Potential gezogen. Um zu kommunizieren, können die Geräte auf dem Bus diese Leitungen mittels eines FET auf Massepotential schalten.

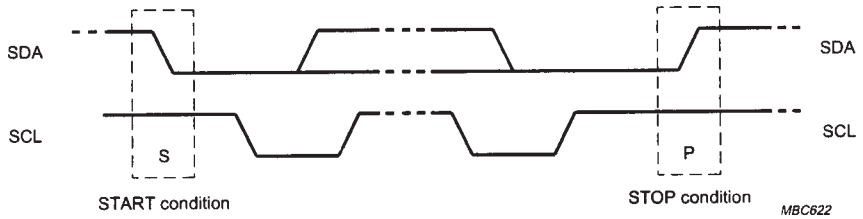
3.2 Start- und Stoppcondition (S. 9 I²C Spec.)

Ein Mastergerät kann einen Datentransfer von Master zu Client oder Client zu Master anfordern. Um dies den Clientgeräten mittzuteilen, wird auf dem I²C-Bus die Startcondition angelegt.
Diese sieht so aus, dass die Datenleitung auf Massepotential (Lo) gezogen wird, während die Clockleitung auf V_{CC} (Hi) verweilt. Das signalisiert allen Clients, dass Daten über den Bus geschickt werden und diese allenfalls aus dem Ruhezustand erwachen müssen.

Jetzt können Daten über den Bus geschickt werden. Dies wird später noch erläutert.

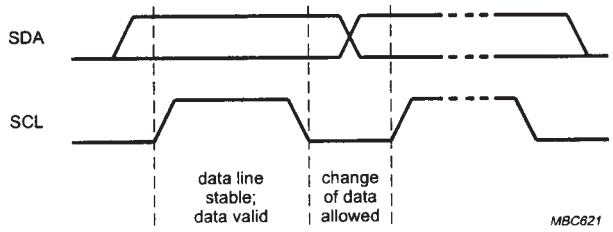
Sind alle Daten über den Bus verschickt worden, wird eine Stoppcondition gesendet, um dem Client das Ende des Datentransfers anzuzeigen und alle Geräte auf dem Bus wieder in den Ruhezustand zurückkehren können.

Per Definition ist geregelt, dass während des Datentransfers die Datenleitung nie ihren Zustand ändern darf, während sich die Clockleitung auf Hi-Potential befindet. Darum können Start- und Stoppcondition niemals unbeabsichtigt auf dem Bus auftreten.



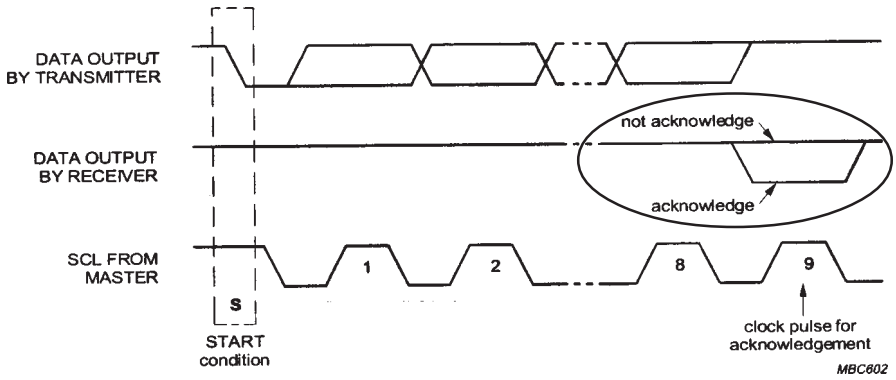
3.3 Bittransfer (S. 8 I²C Spec.)

Während sich die Clockleitung auf Lo-Potential befindet, darf der Zustand der Datenleitung geändert werden. Danach wird sie Clockleitung Hi und ein Master oder Client kann das Bit einlesen. Jetzt darf sich der Zustand der Datenleitung nicht mehr ändern, bis die Clockleitung wieder auf Lo geschaltet wird. Ein Hi der Datenleitung entspricht einem logischen 1 oder true. Wird ein Byte übertragen, wird immer mit dem höchstwertigen Bit begonnen.



3.4 Acknowledge (S. 11 I²C Spec.)

Nach den Transfer eines gesamten Bytes muss der Master oder der Client, welcher das Byte empfangen hat, ein Acknowledge (Bestätigung) schicken.
Dies geschieht, indem der Empfänger nach dem erhalt des letzten Datenbits die Datenleitung auf Lo zieht (Acknowledge) oder auf Hi belässt (Not Acknowledge), bis ein Clockpuls auf dem Bus erfolgt ist.



3.5 Datentransfer (S. 14, 15 I²C Spec.)

Die folgenden Beschreibungen sind vom Master aus gesehen.

Byte Lesen

1. Startcondition herstellen
2. Slave-Adresse senden
3. Read/Write Bit = 0 senden (für Write)
4. Acknowledge von Slave empfangen
5. Speicheradresse senden
6. Acknowledge von Slave empfangen
7. Repeatet Startcondition senden
8. Slave-Adresse senden
9. Read/Write Bit = 1 senden (für Read)
10. Acknowledge von Slave empfangen
11. Datenbyte von Client empfangen
12. Acknowledge senden
13. Stopcondition senden

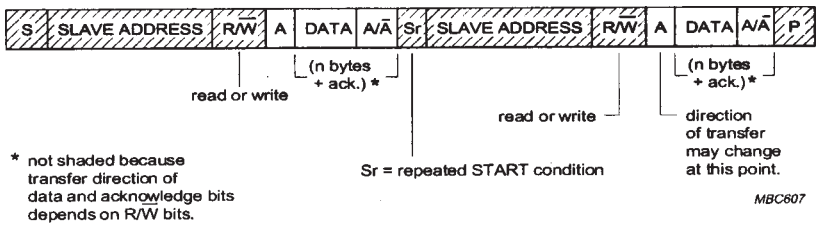


Fig.13 Combined format.

Byte Schreiben

1. Startcondition herstellen
2. Slave-Adresse senden
3. Read/Write Bit = 0 senden (für Write)
4. Acknowledge von Slave empfangen
5. Speicheradresse senden
6. Acknowledge von Slave empfangen
7. Datenbyte an Client senden
8. Acknowledge empfangen
9. Stopcondition senden

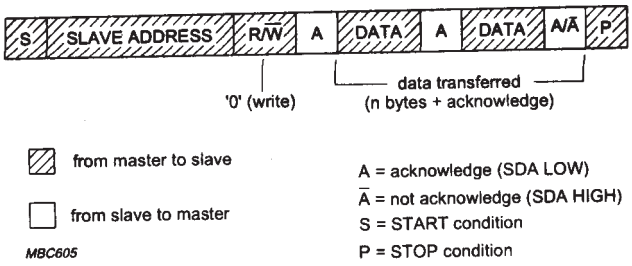


Fig.11 A master-transmitter addressing a slave receiver with a 7-bit address. The transfer direction is not changed.

3.6 Implementation

Die gesamte I²C-Bus funktionalität wird in einem C-File zusammengefasst. Im dazugehörigen Headerfile werden folgende, von aussen zugänglichen Funktionen, deklariert:

char GetI2CByte(char SlaveAddr, char Address);
Empfängt vom angegebenen Slave (SlaveAddr) den Inhalt einer Speicheradresse (Address).

void PutI2CByte(char SlaveAddr, char Address, char Data);
Sendet dem angegebenen Slave (SlaveAddr) ein Byte (Data) für eine Speicheradresse (Address).

char BCDTODEC(char BCD);
Konvertiert eine BCD-Zahl in eine dezimale. (Für RTC-Chip)

char DECTOBCD(char DEC);
Konvertiert eine dezimale Zahl in eine BCD-Codierte. (Für RTC-Chip)

Siehe File i2c.c für genauere Informationen und die Umsetzung...

4.1 RTC-Chip

Einleitung

Der RTC-8583 Chip von Epson hat einen eingebauten Quarz, diverse Features wie Kalender, Alarm, Zeit und universell verwendbares RAM. Angesteuert wird der Chip über eine Zweidrahtleitung mit dem I^2C-Protokoll.

Register

Im Register 0 ($00) können Einstellungen wie Mode (RTC/Counter), Stop, Hold, Mask (für Kalender), Alarm Enable eingestellt werden.

In den weiteren Registern bis Adresse $0F werden die Uhrzeit, das Datum sowie die Alarmzeit und das Alarmdatum gespeichert.

Der Speicherbereich von $10 bis $FF kann frei verwendet werden.

Slave-Adresse (I^2C-Bus)

Die ersten 6-Bit der Slave-Adresse sind fix einprogrammiert. Das siebte kann mit Pin 7 definiert werden. Somit können an einen I^2C-Bus nur maximal zwei von diesen Chips ageschlossen werden, was für die meisten Anwendungen jedoch ausreicht.

Datenformat

Zeit- und Datumswerte sind in den Registern BCD-Codiert abgespeichert. Somit muss softwaremässig eine Umwandlung (z.B. in Dezimalzahlen) vorgenommen werden.

4.2 Funktionstest RTC-Chip

Byte ins RAM schreiben

Mittels der PutI2CByte-Funktion wird ein Byte mit dem Wert 123 auf Adresse $10 geschrieben.

Byte aus RAM lesen

Mittels der GetI2CByte-Funktion wird ein Byte von Adresse $10 gelesen.
Als Ergebnis erhalten wir 123. Somit waren die beiden Datentransfers über den I^2C-Bus fehlerfrei.

Zeitfunktion und Probleme

Ins Sekundenregister ($02) wird der Wert 0 geschrieben.
Nun lesen wir dieses Register periodisch aus.
Das Sekundenregister verhält sich nicht wie erwartet! Es ändert seinen Inhalt nicht mehr im Sekundentakt, sondern bleibt bei 0 stehen.

Als weiteren Versuch schreiben wir den Wert 1 ins Sekundenregister.
Wieder verhält sich der RTC-Chip unerwartet! Der Sekundenwert toggelt zwischen 0 und 1 und zählt nicht mehr weiter. Erstaunlicherweise geschieht dennoch nach 60 Sekunden der Übertrag ins Minutenregister. Es sieht aus, als würde der richtige Sekundenwert mit unserer 1, welche wir geschrieben haben, UND-Verknüpft werden.
Im Datenblatt des RTC-Chip sind keine Informationen über dieses Verhalten zu finden.
Selbst nach einem Power-On Reset über mehrere Stunden bleibt das Verhalten des Chips wie zuvor.

Wir suchten über einen langen Zeitraum (mehrere Tage) nach einer Lösung. Schliesslich schrieben wir 255 ins Sekundenregister (eigentlich unerlaubt, da der Sekundenwert BCD-Codiert sein muss) und seltsamerweise läuft der Sekundenwert wieder von 0 bis 59, wie es zu erwarten ist.

Mit Minuten- und Stundenregistern haben wir dieselben Tests durchgeführt und die selben Verhaltensweisen erhalten.

Wir haben keine Lösung gefunden, eine Zeit in den Chip zu schreiben, so dass er von dieser an weiterzählt.

Funktionstesttool

5.1 Einleitung

Weil trotz tagelanger Suche des Fehlers beim schreiben/lesen der Zeit keine Lösung gefunden werden konnte, können wir die geforderten Ziele der Aufgabenstellkung nicht erreichen.

Deshalb programmieren wir ein Funktionstesttool, welches das korrekte funktionieren der I^2C-Bus Kommunikation sowie das Verhalten der Zeitregister aufzeigt.

5.2 Bedienungsanleitung

LCD-Anzeige

In der ersten Zeile auf dem LCD des TSU-Mikrokontrollerboard ist der Programmtitel zu sehen.
Auf der zweiten Zeile wird der Inhalt des Sekundenregisters angezeigt ($02) und auf der dritten das Minutenregister ($03).
Der Wert auf Zeile vier repräsentiert den Wert im Register $10 (User RAM).
Alle Werte werden mehrmals in der Sekunde aktualisiert.

LED-Anzeige

Die Roten LEDs zeigen den Inhalt des Kontrollregisters ($00) an.
Die Grünen zeigen wie Zeile 2 des LCDs den Inhalt des Sekundenregisters ($02) an, jedoch in seiner ursprünglichen Form und nicht BCD=>Decimal decodiert.

Schalter

7	Schaltet die Hintergrundbeleuchtung des Displays ein/aus
6	Schreibt den Wert 1 ins Sekunden- und Minutenregister
5	Schreibt den Wert 255 ins Sekunden- und Minutenregister
4	Schreibt den Wert der Schalter 0 bis 3 ins Register $10

6.1 Schlusswort

Es war sehr enttäuschend, dass es nicht möglich war, die Zeitfunktionen zum funktionieren zu bringen. Somit konnten wir die geforderten Ziele der Aufgabenstellung nicht erreichen.

Positiv war jedoch, dass der I^2C-Teil unserer Software bestens funktioniert und somit wenigstens das User-RAM des RTC-Chip genutzt werden kann.

6.2 Quellennachweis

Bilder

Die eingescannten Grafiken stammen aus der I^2C-Bus Spezifikation V2.1 von Philips.