Mobiles Hardware-Praktikum 2017

Jan Burchard, Tobias Schubert, Bernd Becker

Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau

Übungsblatt 8 (26 Punkte)

Abgabe: 12.07.2017 bis 17:00 Uhr

Ziel:

Der Roboter fährt die Form eines Quadrats mit Drehungen von möglichst exakt 90 Grad. Dazu wird ein Drehratensensor durch den Arduino ausgelesen, daraus die aktuelle Fahrtrichtung berechnet und diese an den FPGA mittels UART übertragen. Der FPGA übernimmt das Ansteuern der Motoren.

Prolog:

Auf dem Experimentierboard befindet sich ein analoger Drehratensensor. Dieser misst die Drehrate um zwei Achsen und gibt sie als Spannung aus. Ohne Rotation liegt diese Spannung bei 1.23 V.
Um die aktuelle Fahrtrichtung zu bestimmen, muss über die Drehrate integriert werden. Diese Berechnung würde auf dem FPGA auf Grund von Multiplikationen und Divisionen eine große Zahl an LEs belegen.

Das Ansteuern der Motoren ist dagegen auf dem FPGA sehr einfach (siehe Blatt 6) und auf dem Microcontroller mit höherem Rechenaufwand verbunden.

Daher werden die Aufgaben zwischen Microcontroller und FPGA aufgeteilt. Auf dem Microcontroller wird die aktuelle Fahrtrichtung berechnet (**Teil 1**). Diese wird dann an den FPGA gesendet (**Teil 2**). Daraufhin werden die Motoren des Roboters vom FPGA so angesteuert, dass der Roboter ein möglichst exaktes Quadrat fährt (**Teil 3**).

Teil 1:

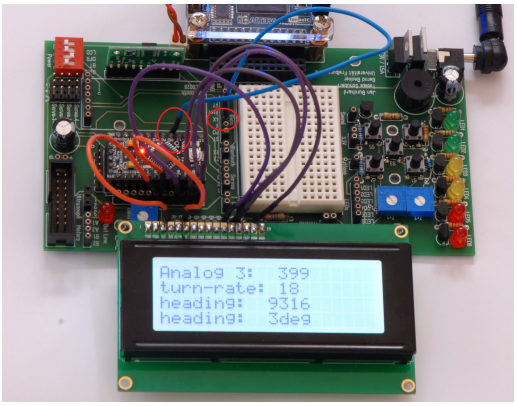
Öffnen Sie zunächst die Datei "TurnRateSensor.ino". Verbinden Sie den Drehratensensor (Buchsenleiste "Gyro") wie folgt mit dem Microcontroller:

Gyro	Microcontroller
4X	A3

Der $4X$ Ausgang ist 4 mal empfindlicher als der X Ausgang des Drehratensensors. Er eignet sich daher besonders für langsamere Drehungen, die mit dem X Ausgang nicht so gut detektierbar sind.

Verbinden Sie außerdem das LCD wie im Quelltext beschrieben. Schalten Sie den Drehratensensor und das LCD an (DIP-Schalter).

Beachten Sie, dass das Auslesen des Drehratensensors so oft wie möglich erfolgen sollte. Vermeiden Sie daher Anweisungen wie “delay()” und “lcd.clear()” in Ihrem Arduino Code.



Aufgabe 1 (1 Punkt):

Geben Sie den aktuellen Spannung am Analog-Pin A3 des Microcontrollers in der ersten Zeile des LCD aus. Wenn das Board nicht bewegt wird, sollte ein konstanter Wert gelesen werden (dieser entspricht 1.23 V, die der Drehratensensor ohne Rotation ausgibt).

Aufgabe 2 (1 Punkt):

Der Drehratensensor gibt die aktuelle Drehrate als Differenz zu dem in Aufgabe 1 beobachteten Wert aus. Speichern Sie die Ausgabe ohne Rotation in einer Variablen und geben Sie in der zweiten Zeile auf dem LCD die Drehrate aus (diese berechnet sich durch "aktueller Wert - Wert ohne Rotation").

Tipp: Warten Sie nach dem Einschalten des Boards einen Moment und speichern Sie dann den aktuellen Wert an A3 als Ausgabe des Drehratensensors ohne Rotation ab.

Tipp: Tipp beachten Sie, dass die Drehrate sowohl positiv als auch negativ sein kann (verwenden Sie zum Speichern also keine *unsigned* Variablen).

Aufgabe 3 (3 Punkte):

Berechnen Sie nun die Richtung des Boards. Dazu müssen Sie über die Drehrate integrieren. Führen Sie hierzu eine neue Variable "heading_int" ein. Messen Sie die aktuelle Drehrate und multiplizieren Sie diese mit der Zeit, die seit der letzten Messung (dem letzten Schleifendurchlauf) vergangen ist (Tipp: *millis()* gibt die Zeit seit Systemstart in Millisekunden aus). Addieren Sie das Ergebnis zur Variable "heading_int". Geben Sie den Wert von "heading_int" in der dritten Zeile des LCD aus.

Hinweis: Verwenden Sie Variablen mit einer hohen Anzahl an Bits um einen Overflow zu vermeiden.

Hinweis: Da der in Aufgabe 2 bestimmte Mittelwert leider nie genau in der Mitte liegt können Sie ein Deadband um die in Aufgabe 2 ermittelte Drehrate legen: Ist die absolute Drehrate kleiner als eine Schranke wird “heading_int” nicht verändert.

Hinweis: Wenn sich das Board nicht bewegt, sollte der Wert von "heading_int" quasi konstant bleiben, ansonsten je nach Drehrichtung entweder fallen oder steigen.

Aufgabe 4 (1 Punkt)

Geben Sie in der vierten Zeile des LCD die Richtung des Boards "heading" in Grad (eine ganze Umdrehung entsprechen 360 Grad) aus. Finden Sie dazu einen passenden Skalierungsfaktor für "heading_int". Beachten Sie außerdem, dass die Ausgabe zwischen 0 und 359 Grad liegen soll (eine Drehung um 360 Grad soll also als 0 Grad ausgegeben werden). Im Folgenden wird mit diesem

skalierten Wert gearbeitet.

Teil 2:

Achtung: Vergewissern Sie sich, dass Teil 1 vollständig funktioniert bevor Sie Teil 2 implementieren da das LCD hier vom Microcontroller getrennt wird.

Der vom Microcontroller berechnete Wert muss nun an den FPGA übertragen werden. Zur effizienten Übertragung von Daten zwischen Geräten gibt es verschiedene Protokolle. Eines davon (SPI) haben Sie auf dem letzten Übungsblatt implementiert. Hier wird eine einfachere Methode gewählt: Serielle Datenübertragung mittels UART (universal asynchronous receiver/transmitter).
Als Besonderheit (im Vergleich zu anderen digitalen Übertragungsmethoden) wird bei UART kein Clock-Signal übertragen. Um einzelne Bit zu erkennen wird ein besonderes Übertragungsformat und eine feste Bitrate (hier: 38400 Bit pro Sekunde) verwendet.

Auf dem Microcontroller können Daten über UART mittels der Seriellen Schnittstelle übertragen werden:

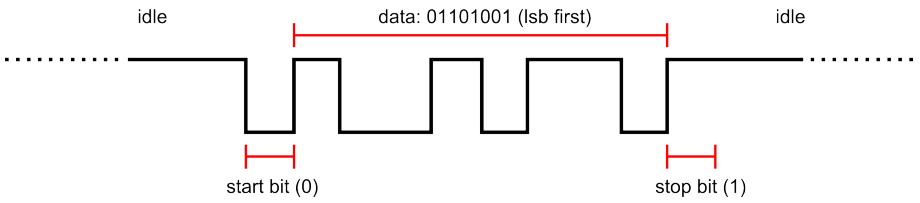
```
...
void setup() {
    Serial.begin(38400);
}
...
void loop() {
    ...
    Serial.write(123); // transmit the number 123 (note: maximum is 255!)
    ...
}
```

Mehr Informationen zu den Arduino Serial-Funktionen finden Sie auf der Arduino Website unter https://www.arduino.cc/en/Reference/Serial.

Damit der FPGA die aktuelle Fahrtrichtung (heading) vom Arduino empfangen kann wird ein UART-Empfänger benötigt. Dieser wird in Teil 2 entworfen.

Übertragungsformat:

Übertragen werden immer 8 Bit. Werden keine Daten übertragen, liegt die Leitung auf logisch '1'. Zu Beginn der Übertragung wird zunächst immer eine logische '0', das "Start-Bit", gesendet. Dann folgen die 8 Bit Daten (das "Least Significant Bit" (lsb) zu erst) gefolgt von einer '1', dem "Stop-Bit". Die Länge eines Bits ist durch die Bitrate festgelegt, beträgt hier also $\frac{1\,s}{38400} \approx 26\,\mu s$. Die Leitung wird immer vom Sender kontrolliert (in diesem Fall vom Microcontroller). Der Empfänger liest nur Daten von der Leitung.



Aufgabe 5 (1 Punkt):

Erstellen Sie ein neues Projekt in Quartus II. Fügen Sie Ihrem Projekt eine neue VHDL-Datei hinzu in der die Entity "UARTreceiver" definiert wird. Diese soll über einen Eingang für die 50 MHz Clock und einen für die Serielle Schnittstelle (jeweils std_logic) verfügen. Die zuletzt empfangenen Daten sollen an einem 8 Bit Ausgang ausgegeben werden. Zusätzlich soll das Modul über

einen “dataReady” Ausgang verfügen, der mit einem ‘1’-Puls anzeigt, dass neue Daten empfangen wurden.
Erstellen Sie eine Symbol-File für die Entity und fügen Sie diese zu einem neuen Blockdiagramm hinzu. Verbinden Sie den Clock Eingang mit Pin R8 und den Seriellen Daten Eingang mit dem Pin E15.

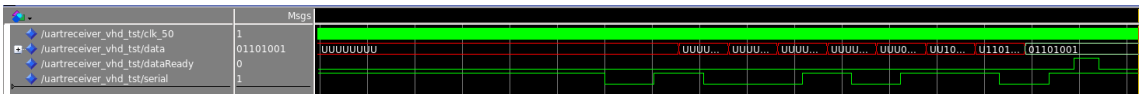
Aufgabe 6 (8 Punkte):

Implementieren Sie die Funktionalität des “UARTreceiver”. Es steht Ihnen dabei frei wie sie vorgehen, die folgenden Tipps sollen nur als Hilfestellung dienen.

- Verwenden Sie einen Zustandsautomaten mit einem Zustand für jedes empfangene Bit.
- Der Übergang vom “idle”-Zustand in den ersten Empfangszustand erfolgt an der fallenden Flanke der seriellen Verbindung.
- Die weiteren Zustandsübergänge erfolgen jeweils nach einer festen Zeit (ca. $26\,\mu s$, siehe oben).
- Lesen Sie das empfangene Bit immer in der Mitte des Zustands.
- Das “dataReady”-Signal wird im gesetzt wenn das “stop”-Bit erfolgreich empfangen wird.
- Sie können die empfangenen Bits direkt an den Ausgang schreiben. Das “dataReady” Signal garantiert, dass der Ausgang nur gelesen wird wenn er stabil ist.

Aufgabe 7 (4 Punkte):

Simulieren Sie Ihren “UARTreceiver”. Erstellen Sie dazu eine Testbench die neben clk_50 auch ein Serielles Signal erzeugt. Überprüfen Sie insbesondere ob die Reihenfolge der ausgegebenen Bits korrekt ist. Das zu erst übertragene Bit (lsb) sollte ganz rechts stehen. Fügen Sie Ihrer Abgabe ein Bild Ihrer Simulation und Ihre Testbench bei.
Eine Übertragung von “01101001” könnte z.B. wie folgt aussehen (beachten Sie, dass dataReady nur nach dem erfolgreichen Empfangen einmal kurz auf ‘1’ wechselt):



Aufgabe 8 (1 Punkt):

Erweitern Sie das Microcontroller Projekt, so dass die aktuelle "heading" regelmäßig über die Serielle Schnittstelle übertragen wird. Da Sie nur 8 Bit senden können (aber “heading” einen Wert zwischen 0 und 359 haben kann) können Sie “heading” nicht ohne Modifikation übertragen. Senden Sie daher den Wert von “heading”/2 (dieser ist immer kleiner $2^8 - 1$).

Aufgabe 9 (2 Punkte):

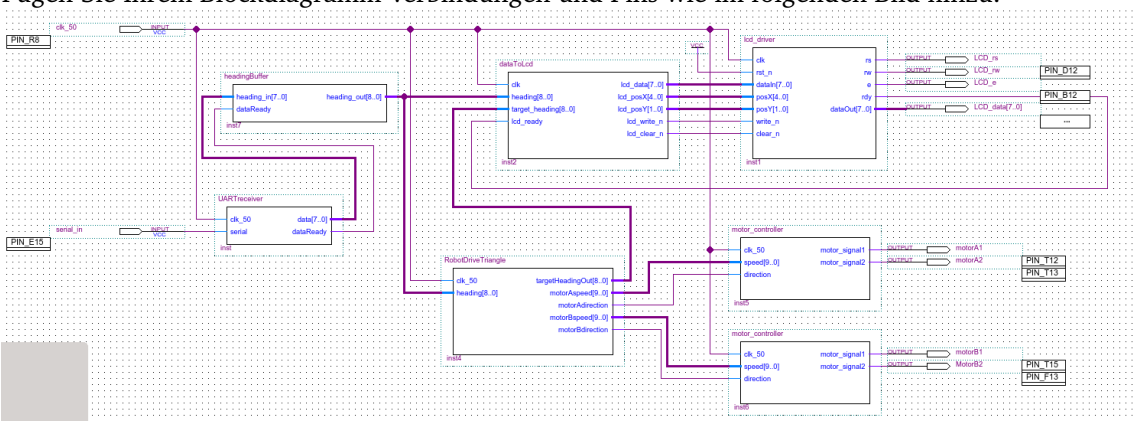
Fügen Sie die vier Dateien aus dem “vhdl” Unterordner zu Ihrem Quartus Projekt hinzu. Diese enthalten die benötigten Dateien um das LCD und die Motoren anzusprechen. Erstellen Sie Symbol-Files für die Dateien “dataToLcd”, “lcd_driver” und “motor_controller” und fügen Sie diese ihr Blockdiagramm ein.

Erstellen Sie zusätzlich eine neue VHDL-Datei “headingBuffer”. Diese soll als Eingaben einen 8 Bit Wert “heading_in” und ein Signal “dataReady” erhalten und eine mit zwei multiplizierte 9 Bit “heading_out” ausgeben. Die Umrechnung soll dabei durch das “dataReady”-Signal gestartet werden (verwenden Sie einen Prozess und Prüfen Sie auf steigende Flanken an “dataReady”).

Erstellen Sie außerdem eine neue VHDL-Datei “RobotDriveTriangle”. Diese soll als Eingabe neben der 50 MHz Clock eine 9 Bit “heading” einlesen und die beiden Motoren über die mitgegebenen

“motor_controller” ansteuern. Sie können außerdem die Zielrichtung (siehe Aufgabe 11) ausgeben und über das “dataToLcd”-Modul auf dem Display anzeigen lassen.

Fügen Sie Ihrem Blockdiagramm Verbindungen und Pins wie im folgenden Bild hinzu.



Verbinden Sie das LCD, die Motoren und den Microcontroller wie folgt mit dem DE0-Nano.

LCD:

LCD	DE0-nano
RS	032 - D12
R/W	nicht verbunden
E	033 - B12
DB0	030 - A12
DB1	031 - D11
DB2	028 - C11
DB3	029 - B11
DB4	026 - E11
DB5	027 - E10
DB6	024 - C9
DB7	025 - D9

Motoren:

Motor	DE0-nano
A1	T12
A2	T13
B1	T15
B2	F13

Datenübertragung:

Micrcontroller	DE0-nano
1 (TX)	E15

Aufgabe 10:

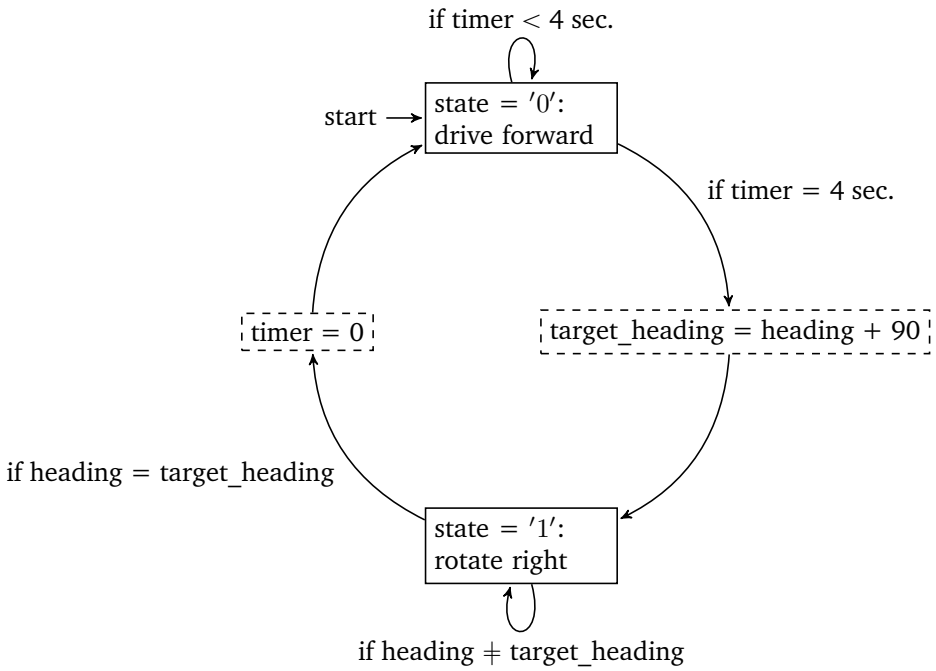
Implementieren Sie den “headingBuffer” wie oben beschrieben. Übertragen Sie das Quartus II-Projekt auf den DE0-Nano. Das LCD sollte nun in der ersten Zeile die aktuelle Fahrtrichtung "heading: XXX" anzeigen. Falls die Kommunikation zwischen Microcontroller und DE0-Nano nicht funktionieren sollte, überprüfen Sie die Kabelverbindungen und drücken Sie den "reset"-Button des Microcontrollers.
Falls Ihre Simulation erfolgreich ist, aber die Daten nicht empfangen werden, können Sie SignalTap II zur genaueren Analyse verwenden. Ein Tutorial dazu finden Sie auf der Website der Veranstaltung.

Teil 3

Im letzten Schritt muss nun eine Steuerlogik auf dem FPGA implementiert werden, die den Roboter im Quadrat fahren lässt.

Aufgabe 11 (4 Punkte):

Implementieren Sie die Funktionalität des Moduls "RobotDriveTriangle": Der Roboter soll immer 4 Sekunden langsam geradeaus fahren, dann um 90 Grad drehen. Verwenden Sie ein neues Signal "state", um einen einfachen Zustands-Automaten wie folgt zu implementieren:



Wenn der Zustand '0' ist, fährt der Roboter vorwärts. Das Signal "timer" zählt dabei die Zeit. Nachdem der Roboter 4 Sekunden geradeaus gefahren ist, wird die Ziel-Richtung (target_heading) berechnet und gespeichert. Nun dreht sich der Roboter so lange, bis die aktuelle Richtung der Ziel-Richtung entspricht, dann wird der Timer auf 0 zurückgesetzt und der Roboter beginnt wieder geradeaus zu fahren.

Hinweise:

- 4 Sekunden entsprechen $200.000.000$ Clock-Zyklen (bei einer Taktfrequenz von 50 Mhz).
- "heading" wird nur für die Drehung, nicht aber für das eigentliche Fahren genutzt. Dadurch kann der Einfluss des Drifts[1] des Drehraten-Sensors minimiert werden.
- "heading" ist ein Wert zwischen 0 und 359 Grad. Die "target_heading" sollte also auch in diesem Bereich liegen (es gilt also z.B.: "350+90Grad = 80 Grad").
- Wenn Sie "heading" mit "target_heading" vergleichen können Sie einen kleinen ϵ-Bereich um "target_heading" legen. Dies stellt sicher, dass sich der Roboter nicht zu weit dreht.

Abgabe:

Archivieren Sie das Quartus II Projekt, archivieren Sie es zusammen mit dem Programm in einer ZIP-Datei und laden Sie diese im Übungsportal hoch. Bewertet werden Ihre Implementationen von Aufgabe 1, 2, 3, 4, 5, 6, 7, 8, 9 und 11 und die Simulation. Überprüfen Sie die Archiv-Datei auf Vollständigkeit. Überprüfen Sie insbesondere auch die im Übungsportal hochgeladene Datei.

[1] Unter Drift versteht man den Fehler, der beim Auslesen des Drehraten-Sensors entsteht: Leichte Schwankungen im Drehraten-Signal sorgen für eine wahrgenommene Drehung, obwohl der Sensor sich nicht bewegt. Bei dem verwendeten Gyro liegt der Drift meistens bei unter 45 Grad pro Minute. Falls Sie einen hohen Drift beobachten (der Roboter steht still, aber "heading" verändert sich), starten Sie den Microcontroller mittels "reset"-Button neu.