Mobiles Hardware-Praktikum 2023

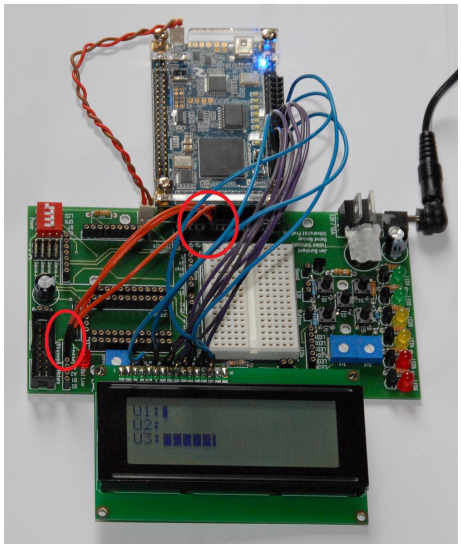
Dr. Philipp Scholl, Marc Pfeifer, Professur für Rechnerarchitektur,
Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau

Übungsblatt 6 (24 Punkte)

Ziel:

Die drei Ultraschall-Sensoren des Roboters werden der Reihe nach ausgelesen, um mögliche Hindernisse vor dem Roboter zu erkennen. Im nächsten Schritt wird eine Motor-Steuerung entworfen mit der die Geschwindigkeit der Motoren reguliert werden kann. Im letzten Schritt wird, basierend auf den Daten von den Ultraschall-Sensoren, eine einfache Steuerung für den Roboter entworfen.

Teil 1: Ultraschall-Sensoren



Importieren Sie das archivierte Projekt “ultrasonic_controller” in Quartus.
Es enthält ein Blockdiagramm mit dem Modul “ultrasonic_controller” zum Auslesen eines Ultraschall-Sensors. Die Entfernung zu erkannten Hindernissen wird auf dem LCD angezeigt (1 volles Kästchen entspricht dabei ca. 10 cm).

Verbinden Sie die drei Ultraschall-Sensoren mit dem DE0-Nano (Pins G15, G16, F14). Verbinden Sie außerdem das LCD wie folgt:

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

Aufgabe 1 (10 Punkte):

Um den Ultraschall-Sensor vom FPGA anzusprechen gehen, Sie genau so vor wie auf Blatt 2 (mit entsprechenden Anpassungen für das FPGA). Hier nochmal eine kurze Anleitung:

Um den Sensor zu aktivieren muss zunächst der Sensor-Pin als Ausgang definieren und einen Trigger Impuls gesendet werden (LOW, dann mindestens 10 μs HIGH, dann wieder LOW). Danach muss der Pin als Eingang definiert werden. Der Sensor hält jetzt die Leitung für eine bestimmte Zeit auf LOW und sendet dann ebenfalls einen HIGH-Puls. Messen Sie die Länge dieses Pulses. Wenn Sie nach 30 ms (= 30000 μs) keinen Puls vom Sender erhalten haben wurde keine Objekt detektiert. Ansonsten können Sie die Entfernung zum detektierten Objekt (in cm) berechnen indem Sie die gemessene Zeit durch 58 dividieren.

Das Modul “ultrasound_controller” steuert **einen** Ultraschall-Sensor (an Pin G15) an. Implementieren Sie zunächst dieses Modul mit Hilfe einer Finite State Machine (FSM).

Die FSM soll sich in einem Initialzustand befinden während das “enable”-Signal den Wert ‘0’ hat. Ändert “enable” seinen Wert auf ‘1’ soll der Durchlauf der Zustände starten. In den ersten Zuständen soll der Trigger Puls erzeugt werden. Im darauffolgenden Zustand wird auf eine Antwort des Sensor gewartet. Bestimmen Sie die Länge des Antwort-Pulses in einem weiteren Zustand und berechnen Sie aus diesem die Entfernung zum erkannten Hindernis in cm. Geben Sie die Entfernung am Ausgang “distance” aus. Die FSM sollte sich nach einem Durchlauf wieder im Initialzustand befinden. Vergessen Sie nicht auch auf einen Timeout angemessen zu reagieren (durch Rückkehr in den Initialzustand).
Das Signal “finished” soll genau dann den Wert ‘1’ haben wenn sich die FSM im Initialzustand befindet.

Wenn der Sensor das “trigger”-Signal steuert darf das FPGA diese Leitung nicht kontrollieren. Daher **muss** in den entsprechenden Zuständen das Signal auf ‘Z’ (hohe Impedanz) gesetzt werden.

*Wichtig: Verwenden Sie zur Berechnung der Distanz **keine Division**. Auf dem FPGA kann eine Division nur realisiert werden, wenn dafür ein (aufwendiger) Hardware-Divider aufgebaut wird. “‘Generieren” Sie hingegen die Distanz direkt während Sie den Antwort-Pulse des Sensors empfangen.*

Simulieren Sie das Modul (vergessen Sie nicht es zunächst als Top-Level Entity zu setzen) mit den folgenden Stimuli mit Hilfe einer RTL-Simulation und fügen Sie Ihrer Abgabe ein Bild der Simulation hinzu. Sie können die Simulationszeit auf 10 ms begrenzen. Die gemessene Distanz sollte ca. 86 cm (5000 μs Trigger-Länge geteilt durch 58) betragen.

```
-- clock generation process
process
begin
   clk_50 <= '1';  wait for 10 ns;
   clk_50 <= '0';  wait for 10 ns;
end process;
```

```
-- stimulation process:
process
begin
   enable <= '0';
   trigger <= 'Z';
   wait for 1 ms;

   -- start the meassurement
   enable <= '1';

   -- wait for the trigger signal to be over
   while trigger /= '1' loop wait for 1 us; end loop;
   while trigger = '1' loop wait for 1 us; end loop;

   -- stop measuring after the first sample
   enable <= '0';

   -- now send response after a short delay
   wait for 10 us;
   trigger <= '1';
   wait for 5 ms;
   trigger <= '0';
   wait for 10 us;
   trigger <= 'Z';

   wait;
end process;
```

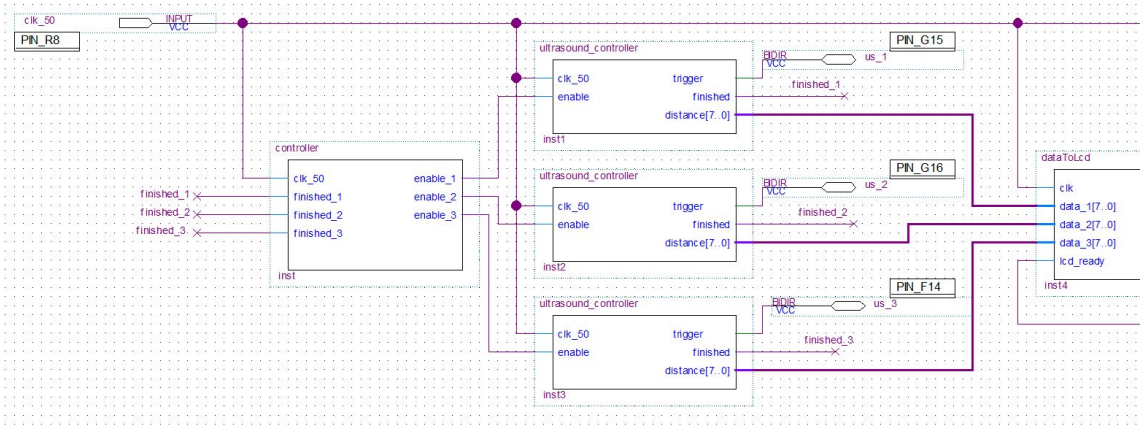
Aufgabe 2

Setzen Sie jetzt die Datei “ultrasound_reader.bdf” als Top-Level Entity, kompilieren Sie ihr Projekt und übertragen Sie es auf den FPGA. Sie sollten auf dem LCD die gemessene Entfernung des Ultraschall-Sensors graphisch angezeigt bekommen.

Aufgabe 3 (4 Punkte):

Das Modul “controller” steuert bisher einen “ultrasound_controller” an. Erweitern Sie das Modul, so dass alle drei Ultraschall-Sensoren angesprochen werden. Fügen Sie dazu zunächst zwei weitere “ultrasound_controller” mit entsprechenden **Bidir**-Pins in das Blockdiagramm ein.
Setzen Sie jeweils das “enable”-Signal eines Controllers auf ‘1’ um eine Messung durchzuführen. Die Messung ist abgeschlossen, wenn das “finished”-Signal wieder den Wert ‘1’ hat.
Beachten Sie, dass Sie die Ultraschall-Sensoren nacheinander ansprechen müssen, da diese sich ansonsten gegenseitig stören. Vergessen sie nicht, die neuen Pins im Pin-Planer mit den entsprechenden FPGA-Pins zu verbinden.

Tipp: Verwenden Sie auch hier eine kleine FSM. Sie können zum nächsten Zustand wechseln, wenn das “finished”-Signal des aktuell aktiven Sensors auf logisch ’1’ wechselt.



Teil 2: Motor-Steuerung

Achtung: Bocken Sie den Roboter auf, bevor Sie die Motoren ansteuern. Achten Sie immer darauf, dass die Räder während der Testphase den Boden nicht berühren können.

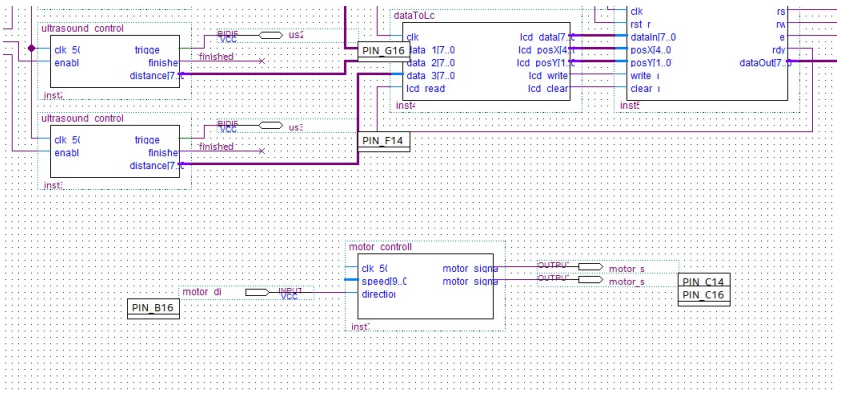
Aufgabe 4 (3 Punkte):

Erstellen Sie eine neue VHDL-Datei “motor_controller.vhd”. In dieser definieren Sie die Entity “motor_controller” mit drei Eingängen (“clk_50”, “speed : std_logic_vector(9 downto 0)”, “direction”) und zwei Ausgängen (“motor_signal1”, “motor_signal2”). Implementieren Sie zunächst die Steuerung für die Drehrichtung der Motoren wie folgt:

direction	motor_signal1	motor_signal2
1	1	0
0	0	1

Verifizieren Sie das Verhalten des Moduls durch Simulation mittels einer Simulation in Modelsim und fügen Sie Ihrer Abgabe ein Bild der Simulation hinzu. Hierzu müssen Sie eine Testbench generieren und darin das Signal “direction” auf logisch ’1’ und nach einer Verzögerung auf logisch ’0’ setzen. Die beiden Ausgänge “motor_signal1” und “motor_signal2” sollten sich wie gewünscht verhalten.
Hinweis: Setzen Sie das Modul “motor_controller” als Top-Level Entity und starten Sie die Synthese, bevor Sie den Test-Bench Template Writer starten.

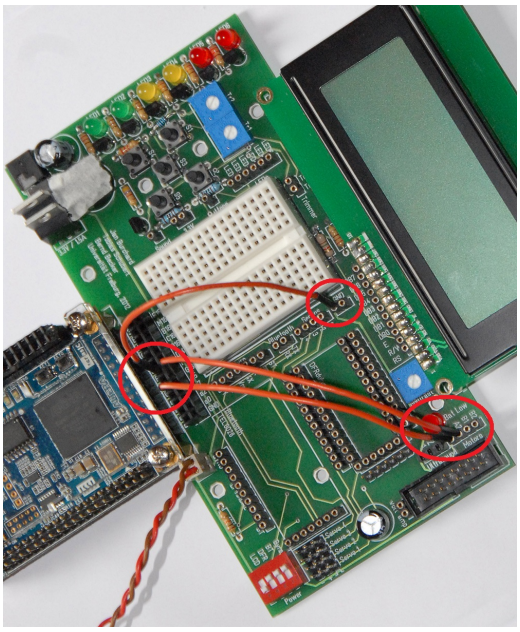
Aufgabe 5:



Erstellen Sie ein Symbol-File zu Ihrer VHDL-Datei und verbinden Sie dieses im Blockdiagramm mit neuen Ein- und Ausgangspins. Die Eingänge “clk_50” und “speed” müssen Sie dabei zunächst nicht verbinden.
Weisen Sie den Pins im Pin-Planer drei GPIO Pins des DE0-Nano zu (die Motorsignale sollten dabei mit den Pins C14 und C16 verbunden werden). Verbinden Sie dann den Pin “motor_signal1” mit

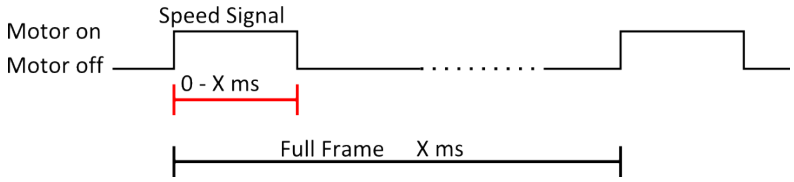
dem Motor-Pin 'A1' und den Pin "motor_signal2" mit 'A2' auf dem Experimentierboard. Den Pin "direction" verbinden Sie zunächst mit dem GND Socket.

Laden Sie ihr Design auf das FPGA. Der Motor sollte sich nun mit voller Geschwindigkeit drehen. Wenn Sie den "direction"-Pin mit dem 3.3V Pin verbinden sollte er sich in die andere Richtung drehen.



Aufgabe 6 (4 Punkte):

Erweitern Sie nun das Modul "motor_controller" um eine Geschwindigkeits-Regulierung mittels Pulsweiten-Modulation (PWM). Diese soll etwa 1500 mal pro Sekunde zwischen "Motor an" und "Motor aus" (motor_signal1 = '0' und motor_signal2 = '0') hin- und her schalten. Die Geschwindigkeit des Motors kann über das Verhältnis von "Motor an" zu "Motor aus" geregelt werden:

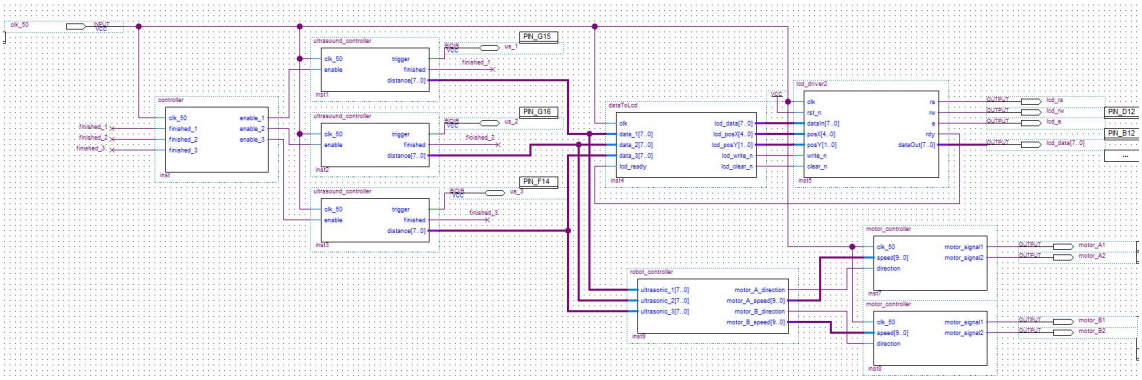


Die Geschwindigkeit wird dabei durch das "speed"-Signal bestimmt und soll zwischen "Motor immer aus" und "Motor immer an" reguliert werden können.

Tipp: Da Multiplizieren und Dividieren in Hardware sehr teuer ist, versuchen Sie am besten ohne diese Operationen auszukommen. Verwenden Sie stattdessen Shift-Operationen (diese entsprechen einer Multiplikation/Division mit 2^X).
Testen Sie Ihre Implementierung durch Simulation. Beachten Sie dabei vor allem die Randfälle "immer aus" und "immer an" und fügen Sie Ihrer Abgabe ein Bild der Simulation hinzu. Achten Sie drauf, die Simulationszeit lang genug zu wählen, da andernfalls keine vollständigen PWM-Perioden zu erkennen sind. Sie können Ihre Simulation von Aufgabe 4 erweitern.

Teil 3: Roboter Steuerung

Aufgabe 7 (3 Punkte):



Erstellen sie eine neue VHDL-Datei “robot_controller.vhd”, in der die Entity “robot_controller” definiert wird. Diese hat drei 8-Bit Eingänge für die Ultraschall-Sensoren und vier Ausgänge: “motor_A_speed” (10-Bit), “motor_A_direction”, “motor_B_speed” (10-Bit) und “motor_B_direction”. Erstellen Sie wie in Aufgabe 5 ein Symbol-File und fügen Sie das Symbol in Ihr Blockdiagramm ein. Fügen Sie außerdem einen zweiten “motor_controller” hinzu. Verbinden Sie den “robot_controller” mit den zwei “motor_controller” und den drei “ultrasound_controller”. Fügen Sie die noch fehlenden Pins in das Blockdiagramm ein und verbinden Sie diese im Pin-Planer mit passenden FPGA-Pins (C15 und D16). Verbinden Sie nun auf dem Experimentierboard die gewählten FPGA-Pins mit den Motor-Pins “B1” und “B2”.

Implementieren Sie das Modul “robot_controller”, so dass der Roboter langsam vorwärts fährt. Sobald ein Ultraschall-Sensor ein Hindernis erkennt soll der Roboter reagieren (stehen bleiben, ausweichen, rückwärts fahren, ...). Die Steuerung muss dabei weder komplex sein noch alle drei Ultraschall-Sensoren auslesen. Erklären Sie kurz das Verhalten Ihres Roboters in einem Kommentar.

Abgabe

Archivieren Sie Ihre Implementierung von Teil 1-3. Laden Sie das Archiv im Übungsportal hoch. Sie können alle Aufgaben gemeinsam in einem Projekt abgeben. Überprüfen Sie die Archiv-Datei auf Vollständigkeit. Vergessen Sie die Bilder der Simulationen nicht.