Mobiles Hardware-Praktikum 2023

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

Übungsblatt 4 (24 Punkte)

Teil 1: Quartus Einführung

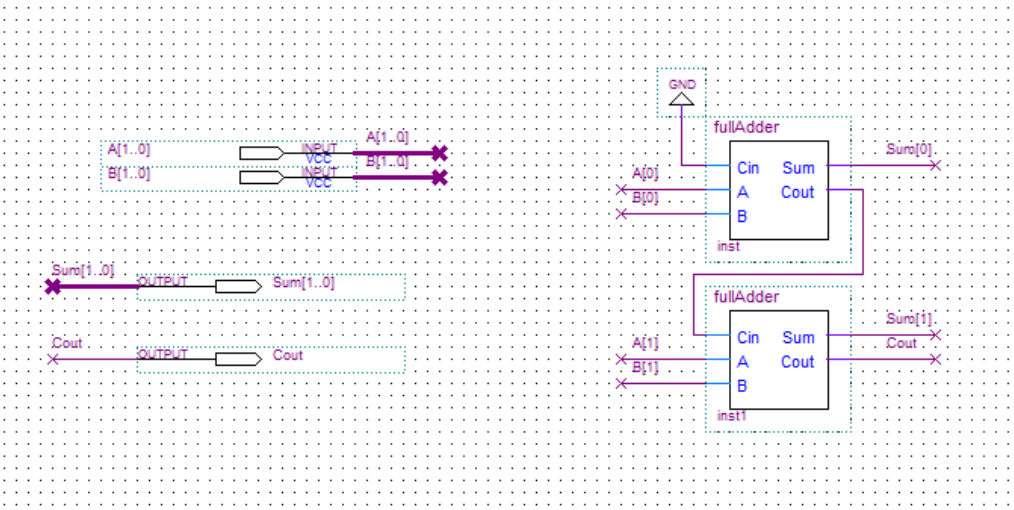
Hinweis:

Laden Sie das “Quartus Prime Tutorial” aus ILIAS herunter. Folgen Sie dem Tutorial Schritt für Schritt. Sie können den dort entwickelten Volladdierer für die weiteren Aufgaben verwenden.

Die Quartus Prime Software bietet eine sehr große Anzahl an Einstellungen und Funktionen. Daher lässt sich eine Einarbeitung und etwas trial-and-error nicht vermeiden. Achten Sie insbesondere bei Simulationen darauf die Optionen genau so wie im Tutorial beschrieben zu setzen.

In dieser Übung wird ein Carry-Ripple und Conditional-Sum Addierer entwickelt. Informieren Sie sich im Internet über diese, falls Ihnen deren Funktionsweise nicht bekannt ist.

Aufgabe 1 (6 Punkte):



Erstellen Sie auf Basis des Volladdierers aus dem Quartus Prime Tutorial einen 2-Bit Carry-Ripple-Addierer. Der resultierende Schaltkreis sollte aus zwei Volladdierern, zwei 2-Bit Input-Pins (A[1..0], B[1..0]), einem 2-Bit Output-Pin (Sum[1..0]) und einem 1-Bit Output-Pin (Cout) bestehen. Simulieren Sie Ihren 2-Bit Carry-Ripple-Addierer für alle möglichen Werte von A und B (A=0..3, B=0..3). Fügen Sie Ihrer Abgabe ein Bild der Simulation bei.

Aufgabe 2 (6 Punkte):

Erstellen Sie einen 2-Bit-Multiplexer. Der resultierende Schaltkreis sollte aus Grundgattern (AND, OR, NOT), zwei 2-Bit Input-Pins (S0[1..0], S1[1..0]), einem 1-Bit Input-Pin (*select*), sowie einem 2-Bit Output-Pin (result[1..0]) bestehen. Hat *select* den Wert 0 soll A mit dem Ausgang verbunden

werden, ansonsten B. Erstellen Sie, wie im Tutorial beschrieben, ein Block Symbol MUX aus dem Schaltkreis.

Aufgabe 3 (6 Punkte):

Verwenden Sie den Volladdierer aus dem Quartus Prime Tutorial und den Multiplexer aus Aufgabe 2, um einen 2-Bit Conditional-Sum-Addierer zu erstellen. Der resultierende Schaltkreis sollte aus drei Volladdierern, einem Multiplexer, zwei 2-Bit Input-Pins (A[1..0], B[1..0]), einem 2-Bit Output-Pin (Sum[1..0]) und einem 1-Bit Output-Pin (Cout) bestehen. Simulieren Sie Ihren 2-Bit Conditional-Sum-Addierer für alle möglichen Werte von A und B (A=0..3, B=0..3). Fügen Sie Ihrer Abgabe ein Bild der Simulation bei.

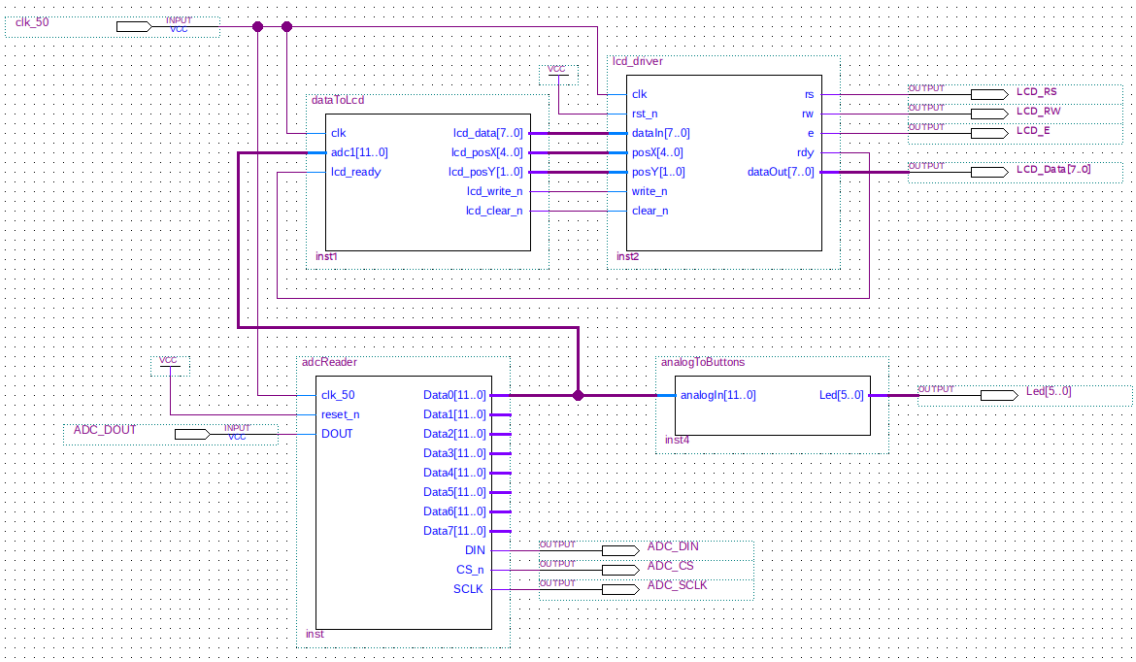
Teil 2: Buttons

Hinweis:

Ziel des zweiten Teils des Übungsblatts ist es eine Button-Erkennung in VHDL zu programmieren. Diese soll den aktuell gedrückten Button durch eine leuchtende LED anzeigen. Da der FPGA keinen internen Analog-Digital Converter (ADC) besitzt, wird stattdessen ein externer Chip auf dem DE0-Nano Board für diese Aufgabe verwendet. Um Ihnen bei der Entwicklung zu helfen wird außerdem das LC Display mit dem FPGA verbunden um den aktuellen Wert des ADC anzuzeigen.
Verbinden Sie den Anschluss **Buttons** mit dem Analog Pin A0 des FPGA (an der Stirnseite des DE0-Nano). Verbinden Sie außerdem die LEDs 1-6 mit den Pins E9, F9, F8, E8, D8, und E7 des FPGA. Die Verbindungen für das LCD entnehmen Sie bitte der folgenden Tabelle:

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 4 (6 Punkte):



Öffnen Sie das Template “ex4buttonsToLEDs.qar”. Darin enthalten ist das oben gezeigtes Block-Diagramm “ex4buttonsToLEDs.bdf”. Dieses besteht aus vier Teilen: Das Modul “adcReader” liest alle 8 Kanäle des ADC aus. Dieses Modul ist bereits vollständig implementiert und benötigt keine Änderungen. Ebenfalls vollständig implementiert sind die Module zum Ansteuern des LCD. Das Modul “analogToButtons” konvertiert einen Wert des ADC in einen gedrückten Button.
Öffnen Sie dieses Modul. Es hat einen 12-Bit Eingang “analogIn” und einen 6-Bit Ausgang. Implementieren Sie die Funktionalität des Moduls so dass LED i nur dann leuchtet (‘1’ ist), wenn Button i gedrückt wird. Wird keine Taste gedrückt, soll LED 6 leuchten.

Beachten Sie, dass der ADC auf dem DE0-Nano mit 12 Bit Auflösung arbeitet, die gemessenen Werte liegen also zwischen 0 und $2^{12} - 1$.

Bei allen VHDL Aufgaben das VHDL best practice Dokument zu beachten (zu finden im ILIAS). Auf diesem Blatt ist insbesondere Punkt 4 relevant!

Abgabe

Archivieren Sie Ihre Implementierung von Aufgabe 1-3 (inklusive der Simulationen als Bild) und Aufgabe 4 und fügen Sie beide Quartus Prime Archiv-Dateien zu einer ZIP-Datei hinzu. Laden Sie diese im Übungsportal hoch. Überprüfen Sie die Archiv-Datei auf Vollständigkeit.