

Versuch 3: Addierer

Einführung

Im Versuch 3 wird wieder mit der PLD-Entwicklungssoftware Altera MAX+plusII und dem PLD-Board SB2 gearbeitet. Der Versuch besteht aus zwei voneinander abhängigen Teilen die in der angegebenen Reihenfolge gelöst werden müssen:

1: 4-Bit Addierer. 2: 2er Komplement Darstellung.

Es werden die im Versuch 2 erworbenen Kenntnisse über die Bedienung von MAX+ vorausgesetzt. Wer sich unsicher fühlt, konsultiere für die schrittweise Beschreibung des Vorgehens die Anleitung zum Versuch 2 Teil 1.

Das Lösungsblatt zum Versuch erhalten Sie am Praktikumsnachmittag.

Vorbereitung am Praktikumsnachmittag:

Am Netzgerät eine Ausgangsspannung von ca. 6 bis 7 Volt einstellen (nach Skala hinter dem Einstellknopf genügt). Strombegrenzungsregler auf ca 50% einstellen. Das PLD-Board mit dem Speisekabel am Netzgerät anschliessen. Netzgerät einschalten. PLD-Board mit dem 25poligen Verbindungskabel an den Parallelport (LPT1) des PC anschliessen.

Im Praktikumsverzeichnis DigiPrakt ein neues Arbeitsverzeichnis V3 anlegen. In dieses den 7-Segment Decoder von V2 laden (Ihre Files oder die bereitgestellte Lösung von der DigiPrakt Seite).

1: 4-Bit Addierer

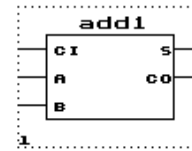
Einführung

Der Addierer ist die **Mutter aller Rechenschaltungen**. Alle anderen Mathematischen Operationen können mehr oder weniger aufwändig darauf zurückgeführt werden. Während sehr langer Zeit war der Addierer denn auch die einzige Rechenschaltung in Rechnern und Computern. Heute noch verfügen einfache Mikrocomputer und Mikrocontroller immer noch oft nur über einen Addierer. Alle komplizierteren mathematischen Operationen (Multiplikation, Division) werden mit sehr ausgereiften Algorithmen auf der Addition aufgebaut. Wir bauen in diesem Versuch einen einfachen **4-Bit ripple-carry Addierer** auf. Diese elementare Architektur (die Mutter aller Addierer) benötigt den kleinsten Aufwand, ist jedoch auch die langsamste. Das Problem bei der Addition besteht darin, dass die einzelnen Stellen nicht unabhängig voneinander bearbeitet werden können: von jeder Stelle zur Nächsthöheren kann ein Übertrag stattfinden.

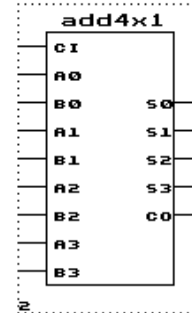
Aufgaben

1. Legen Sie ein neues Grafik Editor File add1.gdf an (dieses und alle weiteren im Verzeichnis V3 speichern). Zeichnen Sie das Schaltschema eines 1-Bit (Voll-)

Addierers mit den Eingängen a, b, ci (carry input) und den Ausgängen s und co (carry output). Erzeugen Sie davon ein Symbol für die weitere Verwendung (siehe nebenstehendes Bild).



- Setzen Sie wieder in einem neuen Grafikfile (add4x1.gdf) einen 4-Bit Addierer (ripple Carry) aus 4 1-Bit Addierstufen (add1) zusammen. Erzeugen Sie davon ein Symbol für die weitere Verwendung (siehe nebenstehendes Bild).
- Um den Addierer austesten zu können, wollen wir die beiden Summanden mit den beiden 4-Bit Schaltern (SA und SB) auf dem Board eingeben und die Summe auf die 7-Segment Anzeige ausgeben.



Ein neues Grafik File (add4_sab-db.gdf) für die Eingabe des Schaltschemas öffnen. Addierer (add4x1) und 7-Segment Decoder (7seg) als Symbole in die Zeichnungsebene holen.

Als 7-Segment Decoder können Sie Ihre Lösung aus dem zweiten Teil von Versuch 2 verwenden oder den ganzen Decoder von der DigiPrakt Seite herunterladen.

Die Schaltersignale ($nSA[1..4]$ und $nSB[1..4]$, 1=LSB, 4=MSB) mit Eingangssymbolen einführen (Inversion nicht vergessen!), zusätzlich die Drucktaste KB zur Erzeugung des Eingangscarrys.

Alle Segmente der beiden Anzeigen ($nDA[1-7]$ und $nDB[1-7]$) mit Ausgangssymbolen einführen.

Schaltung verdrahten: Summen-Ausgang des Addierers über den 7-Segment Decoder auf Anzeige B. Invertierte Schaltereingänge auf die Summandeneingänge des Addierers führen. Den Carry Eingang an die Drucktaste KB anschliessen (nicht auf GND legen, da der Compiler damit ein Problem hat!). Den Carry Ausgang nur mit 0 oder 1 (wie in der V2-Aufgabe Simple-Logic) auf der Anzeige A anzeigen.

Fertige Schaltung kompilieren und ins PLD programmieren (Pin-Locking und Schaltung zum Projekt machen nicht vergessen!).

- Die Funktion des Addierers auf dem Board austesten. Das Schema und die funktionierende Schaltung von einem Assistenten auf dem Lösungsblatt visieren lassen. ➔ **B**

2: 2er Komplement Darstellung

Einführung

Da Sie den Addierer bereits fertig realisiert haben, bleibt uns noch genügend Zeit die 2er Komplement Darstellung für negative Zahlen zu erfinden. Bereits seit Jahrzehnten werden Integerzahlen (Ganzzahlen) in praktisch jedem Rechner und Prozessor im 2er Komplement dargestellt. Selbstverständlich wird diese Darstellungsart (und andere) in der Vorlesung behandelt werden. Hier wollen wir uns bereits praktisch damit bekannt machen:

Durch Bildung des 2er-Komplements einer positiven Zahl erhält man die entsprechende negative Zahl und umgekehrt!

Aufgaben

Voraussetzung zur Bearbeitung der folgenden Aufgaben ist die Lösung des ersten Teils "4-Bit Addierer" des Versuchs.

Achtung: Wir beschränken uns jetzt auf reine 4-Bit Werte und ignorieren absichtlich allfällige Ausgangscarrys (co des Addierers).

5. Unser Addierer addiert zwei 4-Bit Zahlen im Binärkode. Wir können dabei feststellen, dass bei vielen Bitkombinationen des Addenden (zweiter Summand) das Resultat (die Summe) kleiner wird als der Augmend (erster Summand,- Carry ignoriert!). Wir können deshalb der entsprechende Bitkombination des Augmendenden eine negative Zahl zuordnen. Wir wollen jetzt experimentell ein paar dieser Bitkombinationen bestimmen.

Als Summanden A an SA den Wert 3 (Bitkombination 0011) einstellen (Schalter SA1 on, SA2 on, SA3 off, SA4 off). Für den Summanden B an SB denjenigen Wert suchen, der zum Resultat 2 führt. Nach der Beziehung $3 + X = 2$ muss X und damit die an SB eingestellte Bitkombination dem Wert -1 entsprechen. Diese Bitkombination im Lösungsblatt eintragen und auf die gleiche Art die Bitkombinationen für -2 und -3 bestimmen und eintragen ➔ **B**

6. Wenn wir jetzt als Addenden -4 einstellen, so muss als Summe die für -1 gefundene Bitkombination erscheinen. Stimmt es?
7. Finden sie durch Vergleich der Bitkombinationen für $+1$ und -1 , $+2$ und -2 , $+3$ mit -3 die einfache Regel zur Bildung des 2er Komplements: **-1 ist das 2er Komplement von $+1$ und umgekehrt.** ➔ **B**

Tip: Alle Bits zu invertieren und eine kleine Korrektur anfügen.

8. Das höchstwertige Bit enthält die Information über das Vorzeichen. Tragen Sie die richtige Zuordnung im Lösungsblatt ein. ➔ **B**

Jetzt sollen Sie unter Verwendung des im ersten Teil entwickelten 4-Bit Addierers eine Schaltung zur Umwandlung von 4-Bit 2er Komplementzahlen in die gewohntere Vorzeichen-Betrag Darstellung entwickeln und in den PLD programmieren.

9. Entwickeln Sie nach der unter 7. gefundenen Regel eine Schaltung zur Konversion von 4-Bit 2er Komplement Zahlen in Betrag und Vorzeichen. Der Betrag soll über den 7-Segment Decoder auf der Anzeige B, das negative Vorzeichen in der Anzeige A (Segment 7) angezeigt werden. Die Eingabe der zu konvertierenden Werte soll über den Schalter B erfolgen. ➔ **B**

Tip: Die Schaltung des 4-Bit Addierers (Add4.gdf) modifizieren und unter neuem Namen (2sC-SM.gdf) speichern. Nicht vergessen den Projekt- und Chipnamen, entsprechend anzupassen! (Die Pinlocking-Informationen sollte der Compiler automatisch ins neue File 2sC-SM.acf übernommen haben).

Achtung: Sie können sich die Mühe ersparen den Addierer entsprechend der benötigten reduzierten Funktionalität zu vereinfachen, der Compiler würde sich darüber nur krank lachen, er macht dies automatisch (siehe Aufgaben 11, 12).

Die Schaltung compilieren, ins PLD programmieren und auf dem Board austesten. Schaltschema und Demonstration auf dem Board durch einen Assistenten auf dem Lösungsblatt visieren lassen. . ➔ **B**

10. Bestimmen Sie mit Hilfe des gebauten Konverters experimentell den Wertebereich (negativste, positivste Zahl) für 4-Bit 2er Komplementzahlen. ➔ **B**

Fakultative Aufgaben, geben Bounuspunkte

Zum Schluss wollen wir noch den Aufwand an Chipressourcen zur Implementation der beiden Schaltungen (4-Bit Addierer und Sing-Magnitude Konverter) miteinander vergleichen. Die beiden Schaltschemas erscheinen etwa gleich komplex, jedes enthält einen 4-Bit Addierer und einen 7-Segment Decoder. Beim Konverter fehlt der Schalter SA mit 4 Invertern, dafür sollte es 3 XOR-Tore zur bedingten Bitinversion enthalten. Die gesuchten Informationen finden wir in den vom Compiler automatisch generierten Report-Text-Files (.rpt).

11. Öffnen Sie mit dem Notepad (oder einem anderen Texeditor) die Reportfiles Add4.rpt. und 2sC-SM.rpt. Am Anfang, direkt nach der Copyright-Predigt finden Sie unter Device Summary / Utilization die Ausnützung des PLDs in %. Tragen Sie die entsprechenden Werte auf dem Lösungsblatt ein. ➔ **B**

12. Wie erklären Sie sich den unerwarteten Unterschied in der Belegung der Chip-Ressourcen?

Im hinteren Teil des Reportfiles finden Sie im Abschnitt Equations die minimierten logischen Gleichungen, die der Compiler zur Beschreibung der Funktion nach der Schaltung generiert hat. Hier zeigt sich nun noch eindrücklicher, wie viel einfacher die Funktion des Konverters gegenüber dem Addierer realisiert wird!

13. Berechnen Sie von Hand das 2er Komplement der Zahl -8 (4 Bit). Erklären Sie das wohl nicht erwartete Resultat. . ➔ **B**

Schluss

Den Addierer werden wir im Versuch 5 wieder verwenden. Deshalb sollten Sie Ihre heutige Arbeit sicherstellen. Dazu gehen Sie wie folgt vor:

File Add4x1.gdf öffnen (z.B. durch Doppelklick in Symbol add4x1).

Add4x1 als aktuelles Projekt definieren: File > Project > SetProjectToCurrentFile.

Projekt archivieren: File > Project > Archive

Jetzt sind alle für die weitere Verwendung wichtigen Files ins Archivverzeichnis kopiert. Dieses können Sie als Ganzes in Sicherheit bringen (siehe Hinweise Schluss Versuch 2). Wenn Sie dies nicht machen, so können Sie für die weiteren Versuche die auf der Praktikumsseite bereitgestellten Lösungsfiles herunterladen.

Bevor Sie den PC herunterfahren und alles ausschalten unbedingt sämtliche Files und Directories des Versuches löschen sowie auch noch den Papierkorb leeren.

Last but not least, danke für Ihr Interesse und einen recht schönen Abend!