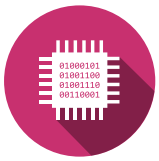


Binäre Addierer

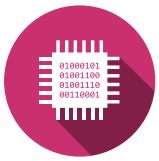
Labor Digitales Design

Inhalt

1 Ziel

Dieses Labor dient dazu, den Entwurf iterativer arithmetischer Schaltungen zu erlernen und praktisch umzusetzen. Im Mittelpunkt steht die Entwicklung eines Addierers als grundlegende arithmetische Schaltung.

Darüber hinaus wird gezeigt, wie sich die erstellten Addierer zur Realisierung eines Subtrahierers nutzen lassen. Durch gezielte Modifikationen wird das Konzept der binären Subtraktion vermittelt, wodurch ein umfassendes Verständnis für arithmetische Operationen in digitalen Schaltungen entwickelt wird.



2 Addierer mit Übertragsfortpflanzung

In digitalen Schaltungen werden Addierer verwendet, um binäre Zahlen zu addieren. Eine grundlegende Implementierung ist der Addierer mit Übertragsfortpflanzung (Carry Ripple Adder). Dieser besteht aus mehreren hintereinandergeschalteten Addierblöcken, in denen sich der Übertrag von einer Stelle zur nächsten fortpflanzt.

2.1 Schaltung

Ein Addierer mit Übertragsfortpflanzung besteht aus mehreren iterativen Blöcken, die jeweils zwei gleichwertige Bits (a_i, b_i) zusammen mit einem eingehenden Übertrag addieren (c_i)). Dabei erzeugt jeder Block:

- Ein Summenbit s_i, dass das Ergebnis der Addition darstellt.
- Einen Ausgangsübertrag c_{i+1}, der an den nächsten Block weitergegeben wird.

Die Abbildung 1 zeigt das Schaltungsprinzip eines solchen Addierers mit Übertragsfortpflanzung:

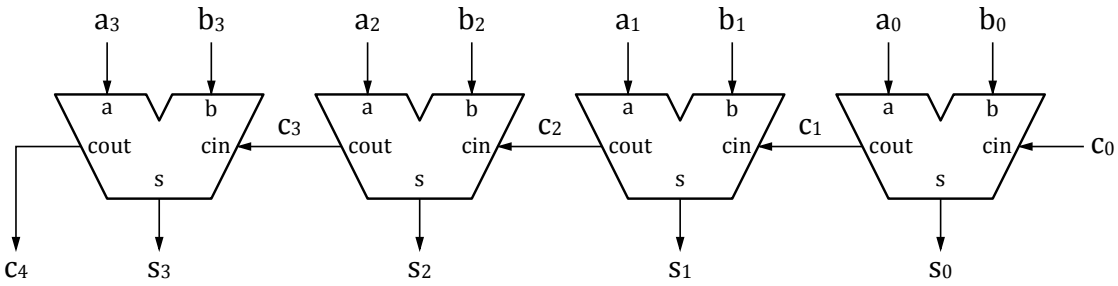


Abbildung 1 - Addierer mit Übertragsfortpflanzung

2.2 Ausführung

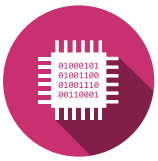
Beginnen Sie mit der Erstellung der Wahrheitstabelle, leiten Sie die Boolesche Gleichung für das Summenbit s_i und den Übertrag c_{i+1} ab. Entwickeln Sie schliesslich die kombinatorische Schaltung unter Verwendung von INV, AND, OR, XOR Gattern.

Implementieren Sie dann das iterative Schema eines 4-Bit Addierers mit Übertragsfortpflanzung im Projekt **`ADD/add4`**.

2.3 Simulation

Jede implementierte Schaltung muss korrekt getestet werden. Die Testbench **`ADD_test/add4_tester`** muss die **vollständige** Funktion der Schaltung überprüfen. Beantworten Sie hierzu die folgenden Fragen:

1. Wieviele Testfälle sind theoretisch notwendig, um die korrekte Funktionsweise des Addierers zu überprüfen?
2. Welche praktischen Testfälle sind notwendig, um die korrekte Funktionsweise des Addierers zu überprüfen?
3. Wie lange dauert die Simulation mit den praktischen Testfällen?



Im **add4_tester** sind bereits einige *Beispiel* Testfälle implementiert, diese können Sie zur Inspiration nehmen und müssen Sie anpassen. Im Listing 1 sehen Sie einen solchen Testfall, dieser überprüft ob die Addition $s = a + b + c_{\text{in}} = 0 + 0 + 0 = 0$ ergibt. Falls nicht wird in Modelsim Terminal die Meldung **test 1 wrong** ausgegeben.

```
--------------------------------------------------------------------------------
-- Test 1:    0 + 0 + 0 = 0
--
a   <="0000";
b   <="0000";
cin <='0';
WAIT FOR clockPeriod;
assert (cout = '0') AND (s="0000")
  report "test 1 wrong"
  severity note;
```

Listing 1 - Beispiel Testfall 1

Vervollständigen Sie den **ADD_test/add4_tester** mit den notwendigen Tests zur vollständigen Überprüfung der Funktionsweise des Addierers.

Simulieren Sie die Testbench **ADD_test/add4_tb** mit der Simulationsdatei **$SIMULATION_DIR/ADD1.do**.

Untersuchen Sie den längsten Pfad im Addierer und finden Sie dessen Länge heraus.

3 Subtrahierer

Ein Subtrahierer ist ein Schaltkreis, der die Subtraktion zweier Zahlen durchführt. Wie wir im Kurs gelernt haben kann man das Vorzeichen einer Zahl im Zweierkomplement invertieren. Dieses bildet die Basis eines Subtrahierers.

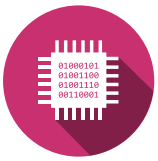
3.1 Schaltung

Die Schaltung eines Subtrahierers kann auf der Grundlage des zuvor entwickelten Addierers erfolgen. Um diese Subtraktion durchzuführen, kann man das Gegenzeichen der zu subtrahierenden Zahl addieren:

$$a - b = a + (-b) \tag{1}$$

Im Zweierkomplement erhält man das Gegenzeichen einer Zahl, indem man alle Bits der Zahl invertiert und zu diesem Zwischenergebnis 1 addiert. Die Inversion aller Bits erfolgt mit einem Inverter für jedes Bit. Die Addition von 1 kann erfolgen, indem man auf den allerersten Übertrag des Addierers **c_0** einwirkt .

Im Block **ADD_Test/sub8_tb** befinden sich 2 Addierer mit jeweils 4 Bits, die so verkettet sind, dass sie einen Addierer mit 8 Bits ergeben. Es gibt auch einen Block, um die Bits der Zahl b zu invertieren.



3.2 Erstellung

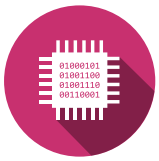
Verändern Sie die Schaltung im einen 8-Bit Subtrahierer zu erstellen.

Zeichne das Schema des Blocks **`ADD/forSubtraction`**, der die Bits der Zahl b invertiert. Erklären Sie wiso das Signal c_{in} invertiert wird.

3.3 Simulation

Jede implemetierte Schaltung muss korrekt getestet werden.

Simulieren Sie die Testbench **`ADD_Test/sub8_tb`** mit der Simulationsdatei **`$SIMULATION_DIR/ADD3.do`**.



4 Checkout

Bevor Sie das Labor verlassen, stellen Sie sicher, dass Sie die folgenden Aufgaben erledigt haben:

- [ ] Schaltungsentwurf
  - [ ] Der Addierer mit Übertragsfortpflanzung **`ADD/add4`** wurde entworfen und getestet.
  - [ ] Der Subtrahierer **`ADD_test/sub8_tb`** wurde entworfen und getestet.
- [ ] Simulationen
  - [ ] Die spezifischen Tests der jeweilgen Testbänke (**`ADD_test/add4_tb`** und **`ADD_test/sub8_tb`**) wurden an die Schaltung angepasst.
  - [ ] Die Simulationen erhalten keine Fehler und alle Berechnungen sind korrekt.
- [ ] Dokumentation und Projektdateien
  - [ ] Stellen Sie sicher, dass alle Schritte (Entwurf, Schaltung, Simulationen) in Ihrem Laborbericht gut dokumentiert sind.
  - [ ] Speichern Sie das Projekt auf einem USB-Stick oder dem gemeinsamen Netzlaufwerk (**`\\filer01.hevs.ch`**).
  - [ ] Teilen Sie Dateien mit Ihrem Laborpartner, um die Arbeitskontinuität sicherzustellen.