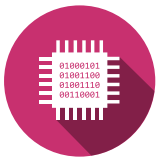


Binäre Addierer

Labor Digitales Design

Inhalt

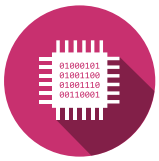
1 Ziel 1
2 Addierer mit Übertragsfortpflanzung 2
2.1 Schaltung 2
2.2 Ausführung 2
2.3 Simulation 2
3 Addierer mit Übertragsvorausschau 4
3.1 Schaltung 4
3.2 Ausführung 4
3.3 Erstellung 5
3.4 Simulation 5
4 Vergleich 6
5 Checkout 7

1 Ziel

Dieses Labor vermittelt ein tieferes Verständnis für die binäre Darstellung von Zahlen und deren Verarbeitung in digitalen Schaltungen. Dabei werden verschiedene Addierer-Architekturen untersucht, um ihre Funktionsweise, Vor- und Nachteile sowie deren Einfluss auf die Berechnungsgeschwindigkeit zu analysieren.

Im Fokus stehen zwei Varianten von Addierern:

- Addierer mit Übertragsfortpflanzung: Eine grundlegende Methode, bei der sich der Übertrag schrittweise durch die Schaltung propagiert.
- Addierer mit Übertragsvorausschau: Eine optimierte Version, die den Übertrag vorab berechnet, um die Verzögerung zu reduzieren.



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:

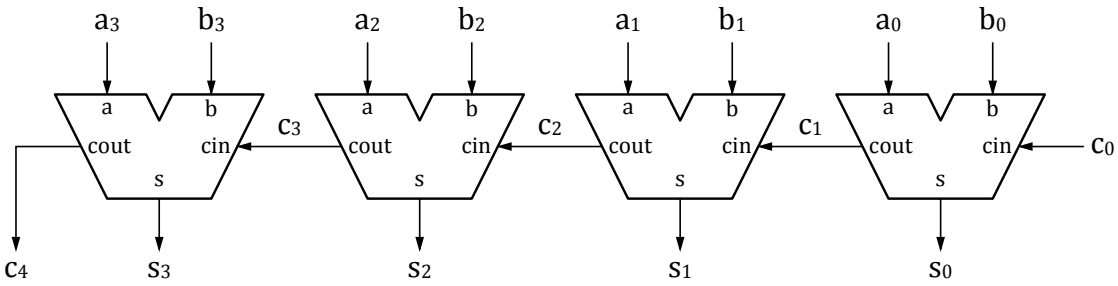


Abbildung 1 - Addierer mit Übertragsfortpflanzung

2.2 Ausführung

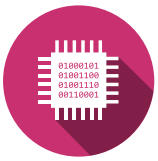
Beginnen Sie mit der Erstellung der Wahrheitstabelle, leiten Sie die Boolesche Gleichung für das Summenbit s_i und den Übertrag c_{i+1} ab. Entwickeln Sie schliesslich die kombinatorische Schaltung unter Verwendung von INV, AND, OR, XOR Gattern.

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.

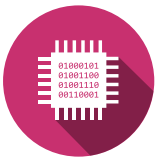
```
--------------------------------------------------------------------------------
-- Test 1:    0 + 0 + 0 = 0
--
a   <="0000";
b   <="0000";
cin <='0';
WAIT FOR clockPeriod;
assert (cout = '0') AND (s="0000")
  report "test 1 wrong"
  severity note;
```

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 Addierer mit Übertragsvorausschau

Dieser Addierer ist optimiert um die Verzögerung zu reduzieren, indem der Übertrag vorab berechnet wird.

3.1 Schaltung

Abbildung 2 zeigt die Schaltung eines Addierers mit Übertragsvorausschau. Sie ist dem Addierer mit Übertragsfortpflanzung ähnlich. Die Übertragskette wurde aber mit 4 neuen Blöcken durchtrennt. Diese Blöcke müssen die jeweiligen Überträge c_1, c_2, c_3 und c_4 mit weniger sequentiellen Gattern berechnen. Dabei wird die Kaskade/Kette von logischen Gattern reduziert, um die Verzögerung der Ausbreitung zu begrenzen.

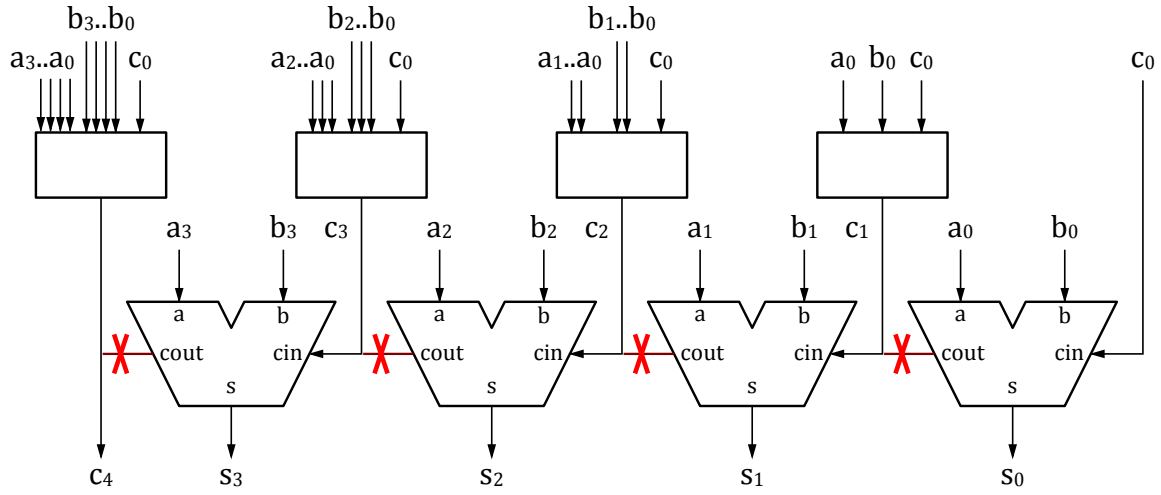


Abbildung 2 - Addierer mit Übertragsvorausschau

3.2 Ausführung

Analysieren wir zuerst den Addieren der Abbildung 2.

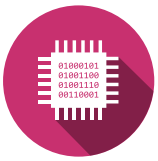
Schreiben Sie die Polynomialsche Gleichung von c_1 und c_2 aus. Wieveiele Terme besitzten diese?

Schätzen Sie die Anzahl Terme des Polynomialsche Gleichung von c_3 und c_4 ein.

Um die Erstellung der Eingangsüberträge zu vereinfachen, führt man zwei Transformation ein, g_i (generate) und p_i (propagte):

$$\begin{aligned} g_i &= a_i * b_i \\ p_i &= a_i + b_i \end{aligned} \tag{1}$$

Zeigen Sie, dass der Ausgangsübertrag und der Bit der Summe wie folgt geschrieben werden können:



$$c_{\text{out}} = g + p * c_{\text{in}} \qquad (2) \qquad c_{i+1} = g_i + p_i * c_i \qquad (3)$$
$$s = (\overline{g} * p) \oplus c_{\text{in}} \qquad\qquad s_i = (\overline{g_i} * p_i) \oplus c_i$$

Mithilfe dieser Umwandlung Gleichung 3, schreiben Sie die Polynomialform von c_1 bis c_4 als Funktion von g_i, p_i sowie c_0 auf.

Um wieviele Terme handelt es sich in diesem Fall?

3.3 Erstellung

Mit Hilfe von INV-, UND-, ODER- und XOR-Gattern, zeichnen Sie das Schema des Blocks, welches die Überträge berechnet (**`ADD/lookAheadCarry`**).

Innerhalb des bereitgestellten 4-Bit-Übertragsvorausschauaddierers **`ADD/addLookAhead4`** vervollständigen Sie die fehlenden Blöcke **`ADD/lookAheadGp`** und **`ADD/lookAheadCarry`**.

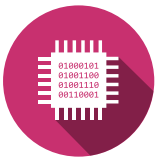
Implementieren Sie den Schaltkreis **`ADD/lookAheadGp`**, welcher die Signale p_i und g_i generiert.

Implementieren Sie den Schaltkreis **`ADD/lookAheadCarry`**, welcher die Überträge c_i generiert.

3.4 Simulation

Überlegen Sie sich auch in diesem Fall wieviele Testfälle notwendig sind, um die korrekte Funktionsweise des Addierers zu überprüfen.

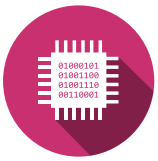
Simulieren Sie die Testbench **`ADD_test/addLookAHead4_tb`** mit der Simulationsdatei **`$SIMULATION_DIR/ADD2.do`**.



4 Vergleich

Berechnen und vergleichen Sie die maximale Verzögerung der beiden erstellen Addierer mit Übertragungsfortpflanzung Abbildung 1 und mit Übertragungsvorschau Abbildung 2.

Welche Vorteile bietet der Addierer mit Übertragungsvorschau **`ADD/addLookAhead4`** gegenüber dem Addierer mit Übertragungsfortpflanzung **`ADD/add4`**? Wie verhalten Sie sich bezüglich Geschwindigkeit und Grösse der Schaltung?



5 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 Addierer mit Übertragsvorausschau **`ADD/addLookAhead4`** wurde entworfen und getestet.
- [ ] Simulationen
  - [ ] Die spezifischen Tests der jeweilgen Testbänke (**`ADD_test/add4_tb`** und **`ADD_test/LooAhead4_tb`**) wurden an die Schaltung angepasst.
  - [ ] Die beiden Addierer wurden hinsichlich Ihrer Leistung und Resourcennutzung verglichen?
  - [ ] 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.