



Ausgabe: 11. Dezember 2017

Abgaben {  Theorie 17. Dezember 2017  
 Praxis 14. Januar 2018  
Rücksprache 15/16. Januar 2018

Auf diesem Aufgabenblatt können insgesamt **20 Punkte** erreicht werden. Daher beträgt die Bearbeitungszeit der Praxisaufgaben **zwei Wochen**. Die Rücksprache für das gesamte Blatt findet in dem Termin nach der Abgabe des gesamten Blattes statt.

### **Aufgabe 1: Carry-Select-Addierer (2 Punkte)**

Erklären Sie das Prinzip und die Funktionsweise eines *Carry-Select*-Addierers.  
Nennen Sie jeweils einen Vor- und Nachteil im Vergleich zu einem *Carry-Ripple*-Addierer.  
Siehe dazu unter Anderem [1] und [3].

#### **Hinweis:**

Beschränken Sie sich bei der Erklärung auf einen einfachen, nicht kaskadierten *Carry-Select*-Addierer.

### **Aufgabe 2: ALU Control (2 Punkte)**

Entwerfen Sie die Steuereinheit für die ALU, wie sie in [2, ab Seite 246] vorgestellt wird. Wie im Buch beschrieben lässt sich die Steuerfunktion durch ganz einfache Logik abbilden (siehe Abbildung 1). Implementieren Sie die Funktionalität in der Datei `aluCtrl.vhd`.

| Name      | Typ                           | Art | Beschreibung   |
|-----------|-------------------------------|-----|--|
| aluOp     | std_logic_vector (1 downto 0) | in  | Betriebsart der ALU                                      |
| f         | std_logic_vector (5 downto 0) | in  | ALU-Optionen (sind Bestandteil der Assemblerinstruktion) |
| operation | std_logic_vector (3 downto 0) | out | Ansteuerungsausgang zur ALU                              |

Tabelle 1: Entity: Ports

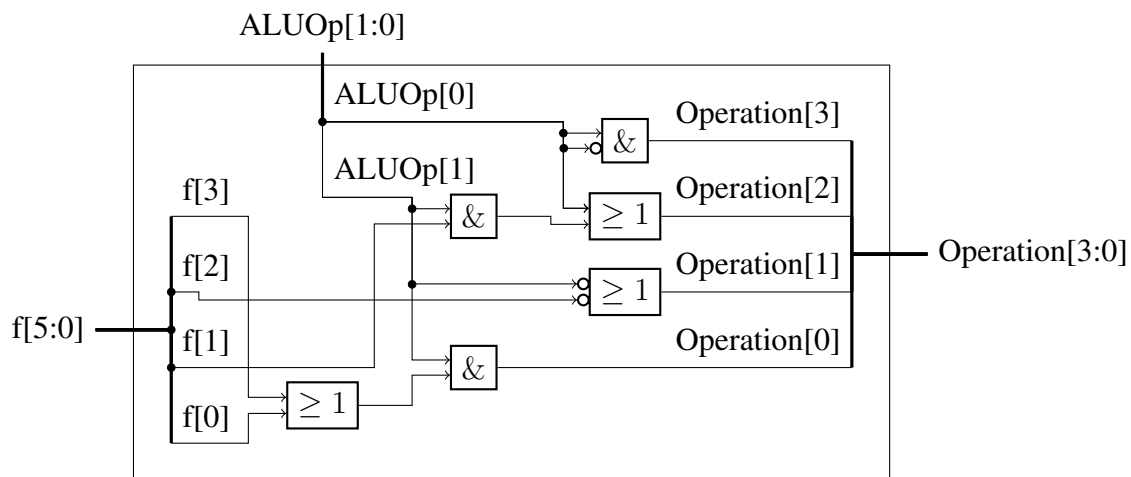


Abbildung 1: Schaltung für die Steuersignale der ALU aus [2]

### Aufgabe 3: ALU Control Testbench (5 Punkte)

Entwerfen Sie eine Testbench zur Überprüfung Ihrer Implementation der `aluCtrl`. Eine Testbench zeichnet sich durch das generelle Fehlen von Ports aus, daher wird auf diese auch nicht weiter eingegangen.

1. (1 Punkt) Instanziierung und Verdrahtung des zu testenden Moduls
2. (1 Punkt) Anlegen eines Arrays von (mind. 4) Testvektoren und Durchlaufen des Arrays in einer Schleife
3. (1 Punkt) Erkennen von Fehlern durch Vergleich mit einem Erwartungswert
4. (1 Punkt) Ausgabe einer Fehlermeldung, wenn das Ergebnis vom Erwartungswert abweicht
5. (1 Punkt) Auswertung am Ende der Testbench (inkl. eindeutiger Aussage, ob das Modul korrekt funktioniert)

Implementieren Sie ihre Testbench in der Datei `aluCtrl_tb.vhd` und führen Sie die Testbench durch einen Aufruf von `make clean all` aus.

### Aufgabe 4: 1 Bit ALU (3 Punkte)

Die ALU ist das Herzstück der MIPS-CPU, sie führt fast alle in der CPU auftretenden Berechnungen durch. Zuerst soll eine 1-Bit-ALU in der Datei `alu_1bit.vhd` implementiert werden, jene ist eine leicht abgewandelte Variante der in [2] vorgestellten ALU. Implementieren Sie die Funktionalität in der `architecture structural` und testen Sie die Implementierung mit der Testbench `alu_1bit_tb`. Setzen Sie in der Testbenchdatei `alu_1bit_tb.vhd` den generischen Parameter `carrySelect` auf `false`.

| Name                     | Typ                  | Art                  | Beschreibung  |
|--------------------------|----------------------|----------------------|---|
| <code>carrySelect</code> | <code>boolean</code> | <code>generic</code> | gibt an, ob ein Carry-Select-Addierer verwendet werden soll |

Tabelle 2: Entity: Generics

| Name      | Typ                           | Art | Beschreibung   |
|-----------|-------------------------------|-----|--|
| operation | std_logic_vector (1 downto 0) | in  | durchzuführende Operation (bzw. Auswahl der richtigen Operation) |
| a         | std_logic                     | in  | erster Dateneingang  |
| aInvert   | std_logic                     | in  | Flag, ob der erste Dateneingang invertiert werden soll           |
| b         | std_logic                     | in  | zweiter Dateneingang   |
| bInvert   | std_logic                     | in  | Flag, ob der zweite Dateneingang invertiert werden soll          |
| carryIn   | std_logic                     | in  | Carry-In für den Addierer  |
| less      | std_logic                     | in  | Eingang für das Ergebnis der <i>slt</i> -Operation               |
| result    | std_logic                     | out | Ergebnis der Operation   |
| carryOut  | std_logic                     | out | Carry-Out des Addierers  |
| set       | std_logic                     | out | Ausgang für das Ergebnis der <i>slt</i> -Operation               |

Tabelle 3: Entity: Ports

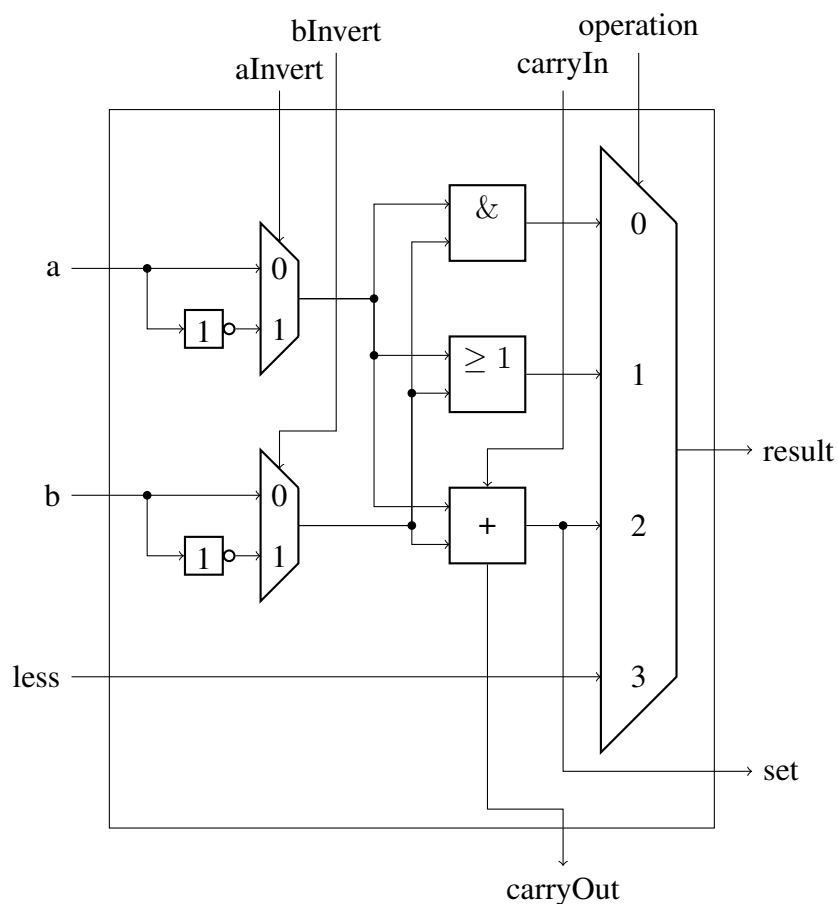


Abbildung 2: Aufbau der 1-Bit-ALU

---

### Aufgabe 5: 1 Bit CS ALU (3 Punkte)

Aus zwei Instanzen des 1-Bit-Volladdierers und zwei Multiplexern soll nun ein 1-Bit-*Carry-Select* Addierer aufgebaut werden (siehe Abbildung 3) und in die vorhandene ALU integriert werden. Implementieren Sie zuerst eine `architecture carrySelect` des Addierers in der Datei `adder_1bit.vhd`. Erweitern Sie anschließend die bestehende `architecture structural` der Datei `alu_1bit.vhd` so, dass in Abhängigkeit des generischen Parameters `carrySelect` entweder die `architecture behavioral` oder `carrySelect` des 1-Bit-Addierers instanziiert wird. Erinnern Sie sich an die bedingte Instanziierung, wie sie in der Theorieaufgabe des sechsten Aufgabenblattes verwendet werden sollte. Sie können zum Testen die Testbench der 1-Bit-ALU aus der vorherigen Aufgabe nutzen, dabei müssen Sie jedoch die Instanziierung der ALU in der Testbench anpassen und den generischen Parameter `carrySelect` auf `true` setzen.

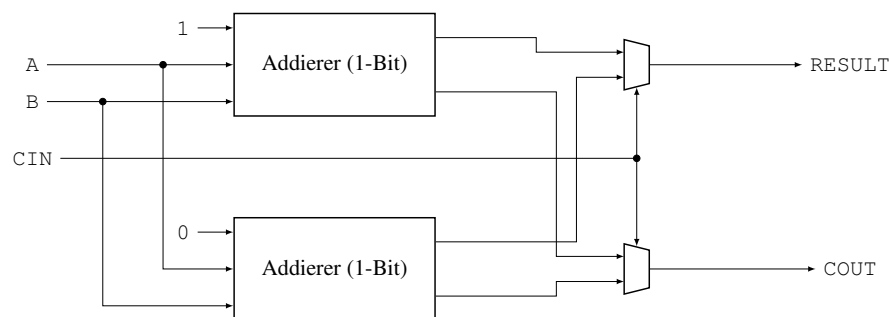


Abbildung 3: Aufbau eines 1-Bit-CS-Addierers

---

### Aufgabe 6: MIPS CS ALU (5 Punkte)

Damit auch  $n$  Bit breite Vektoren verarbeitet werden können, muss aus mehreren 1-Bit-ALUs eine Schaltung gebaut werden. Desweiteren werden einige Ports nicht mehr benötigt, da die gebaute Schaltungen neu “verpackt” wird. Neu hinzu kommen jedoch weitere Aussagen (Flags) über die ausgeführte Operation. Die Funktionalität der  $n$ -Bit-ALU soll in der `architecture behavioral` der Datei `csAlu.vhd` implementiert werden.

| Name     | Typ   | Art | Beschreibung                     |
|----------|---|-----|----------------------------------|
| ctrl     | <code>std_logic_vector(3 downto 0)</code>       | in  | durchzuführende Operation        |
| a        | <code>std_logic_vector(WIDTH-1 downto 0)</code> | in  | erster Dateneingang              |
| b        | <code>std_logic_vector(WIDTH-1 downto 0)</code> | in  | zweiter Dateneingang             |
| result   | <code>std_logic_vector(WIDTH-1 downto 0)</code> | out | Ergebnis der Operation           |
| overflow | <code>std_logic</code>                          | out | Flag, ob es einen Überlauf gab   |
| zero     | <code>std_logic</code>                          | out | Flag, ob <code>result = 0</code> |

Tabelle 4: Entity: Ports

| Name  | Typ     | Art     | Beschreibung                |
|-------|---------|---------|-----------------------------|
| WIDTH | integer | generic | Breite der Eingangsvektoren |

Tabelle 5: Entity: Generics

| ctrl(X)              | Steuerleitung  |
|----------------------|--|
| alu_ctrl(3)          | Ainvert  |
| alu_ctrl(2)          | Binvert <b>Hinweis:</b> Sie können davon ausgehen, dass mit <i>Binvert = high</i> immer eine Subtraktion gemeint ist |
| alu_ctrl(1 downto 0) | AluOp  |

Tabelle 6: Aufschlüsselung des ALU ctrl Ports

1. Implementieren Sie eine generische Schaltungskette aus den 1-Bit-CS-ALUs um eine  $n$ -Bit-CS-ALU zu realisieren.  
**Hinweis:**  
*Für die slt-Operation wird der set-Port des MSB mit dem less-Port des LSB verbunden. Alle anderen less-Ports werden mit einer "0" gespeist, die entsprechenden set-Ports werden auf "open" gesetzt ("open" erklärt einen Ausgangsport für nicht verbunden).*
2. Implementieren Sie eine overflow-Detektion.  
**Hinweis:**  
*Wie funktioniert die Erkennung eines Überlaufs? Wie kann man dies formal beschreiben? Es gibt zwei Möglichkeiten für die Realisierung der Overflow-Erkennung!*
3. Schreiben Sie zusätzliche Logik, welche das zero-Flag ( $result = 0$ ) entsprechend setzt.
4. Bei der overflow-Detektion und dem zero-Flag soll keine Fehlerfortpflanzung ('U' bzw. 'X') erfolgen, sodass hier nur reguläre Signalwerte entstehen ('0' oder '1').
5. Validieren Sie das Verhalten Ihrer Implementation mithilfe der vorgegebenen Testbench csAlu\_tb.

## Literatur

- [1] Hans Liebig and Stefan Thome. *Logischer Entwurf digitaler Systeme*. Springer Berlin, 3., vollst. neubearb. aufl. edition, July 1996.
- [2] David A. Patterson and John L. Hennessy. *Rechnerorganisation und -entwurf*. Spektrum Akademischer Verlag, September 2005.
- [3] Universität Hamburg (TAMS). *Carry-select adder (8 bit)*. [http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/20-arithmetic/20-carryselect/adder\\_carryselect.html](http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/20-arithmetic/20-carryselect/adder_carryselect.html). zuletzt abgerufen am 12.10.2014.