



Ausgabe: 7. Dezember 2015

Abgaben {  Theorie 13. Dezember 2015
 Praxis 11. Januar 2016
Rücksprache 12/13. Januar 2016

Aufgabe 1: Carry-Select Addierer (2 Punkte)

Erklären Sie das Prinzip und die Funktionsweise eines *Carry-Select*-Addierers.
Nennen Sie jeweils ein 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 im Assemblerbefehl einkodiert)
operation	std_logic_vector (3 downto 0)	out	Ansteuerungsausgang zur ALU

Tabelle 1: Entity-Ports

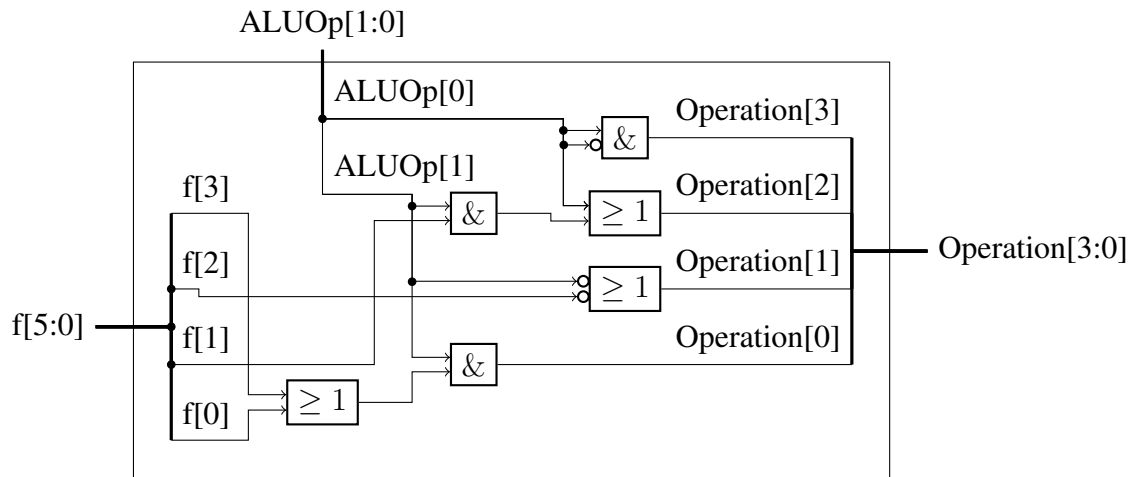


Abbildung 1: Schaltung für die Steuersignale der ALU aus [2]

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, insofern 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 aluCtrl_tb` 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 behavioral` und testen Sie die Implementierung mit der Testbench `alu_1bit_tb`.

Name	Typ	Art	Beschreibung
operation	std_logic_vector (1 downto 0)	in	durchzuführende Operation (bzw. Auswahl der richtigen Operation)
a	std_logic	in	1. Dateneingang
aInvert	std_logic	in	Flag, ob der 1. Dateneingang invertiert werden soll
b	std_logic	in	2. Dateneingang
bInvert	std_logic	in	Flag, ob der 2. 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 2: Entity-Ports

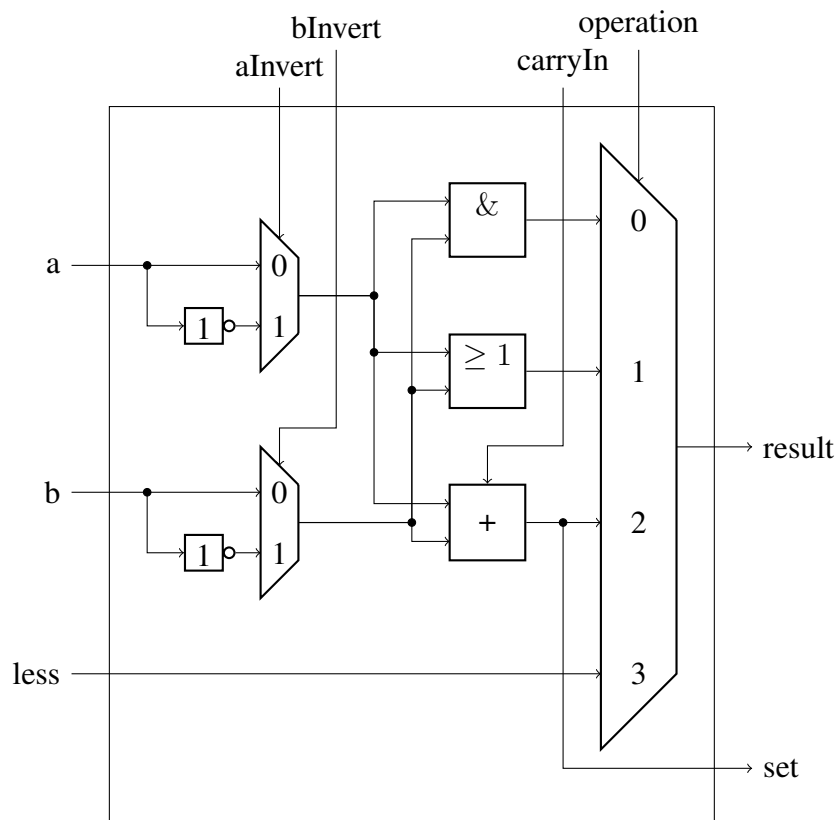


Abbildung 2: Aufbau der 1-Bit ALU

Aufgabe 5: 1 Bit CS ALU (2 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 die `architecture carrySelect` in der Datei `alu_1bit.vhd` mit einer Beschreibung einer 1-Bit ALU unter Verwendung einer CS Addierers um eine 1-Bit CS ALU zu realisieren. Sie können zum Testen die Testbench der 1-Bit aus der vorherigen Aufgabe ALU nutzen, dabei müssen Sie jedoch die Instanziierung der ALU in der Testbench anpassen.

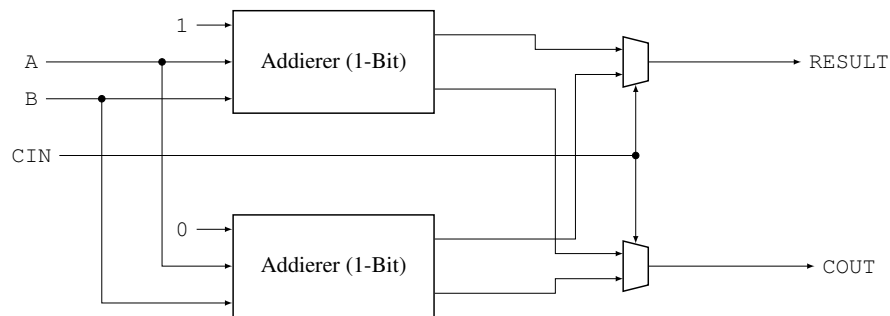


Abbildung 3: Aufbau eines 1-Bit CS Addierers

Aufgabe 6: MIPS CS ALU (6 Punkte)

Damit wir auch n -breite Vektoren verarbeiten können, müssen wir mithilfe der 1-Bit ALUs eine Schaltung bauen. Desweiteren brauchen wir einige Ports nicht mehr, da wir die gebaute Schaltungen neu “verpacken”. Neu hinzu kommen jedoch weitere Aussagen (Flags) über die letzte Operation. Die Funktionalität der n -bit ALU soll in der Datei `csAlu.vhd` implementiert werden.

Name	Typ	Art	Beschreibung
ctrl	std_logic_vector(3 downto 0)	in	durchzuführende Operation
a	std_logic_vector(WIDTH-1 downto 0)	in	1. Dateneingang
b	std_logic_vector(WIDTH-1 downto 0)	in	2. Dateneingang
result	std_logic_vector(WIDTH-1 downto 0)	out	Ergebnis der Operation
overflow	std_logic	out	Flag, ob es einen Überlauf gab
zero	std_logic	out	Flag, ob <i>result</i> = 0

Tabelle 3: Entity-Ports

Name	Typ	Art	Beschreibung
WIDTH	integer	generic	Breite der Eingangsvektoren

Tabelle 4: Entity-Generics

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

Tabelle 5: Aufschlüsselung des ALU ctrl Ports

1. Implementieren Sie eine generische Schaltkette 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's mit dem less-Port des LSB verbunden. Alle anderen less-Ports werden mit einer "0" gespeißt, die entsprechenden set-Ports werden auf "open" gesetzt. ("open" erklärt einen Ausgangsport als 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 eine zusätzliche Logik, welche das zero-Flag ($result = 0$) entsprechend setzt.
4. 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. zuletzt abgerufen am 12.10.2014.