Arithmetische und Logische Einheit

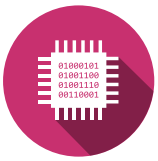
Labor Digitales Design

Inhalt

1 Ziel

Dieses Labor dient dazu, den Entwurf logischer Schaltungen mit Hilfe von Multiplexern zu üben. Es vermittelt eine Methode zur Umsetzung von arithmetischen und logischen Einheiten (Logical Unit (LU) und Arithmetic Unit (AU)) für Mikroprozessoren.

In einer ersten Laborsitzung werden die LU und AU realisiert. In einer zweiten Sitzung wird daraus eine vollständige Arithmetic and Logical Unit (ALU) aufgebaut, welche die beiden Einheiten kombiniert und durch geeignete Steuersignale das gewünschte Ergebnis auswählt.



2 Logische Einheit LU

Die Abbildung 1 zeigt die Schaltung einer Logischen Einheit (LU) eines Mikroprozessors. Die logische Operationen werden Bit für Bit durchgeführt. Acht dieser Blöcke bilden eine 8-bit LU .

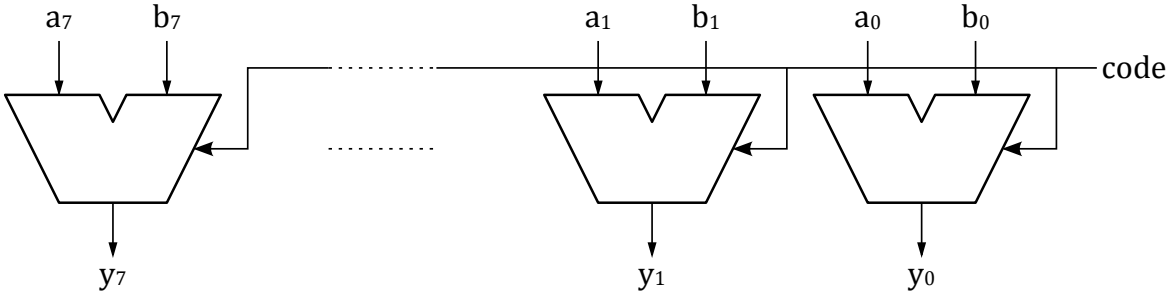


Abbildung 1 - Logische Einheit LU

Die logische Schaltung der einzelnen Iterativblöcke werden mit Multiplexern erstellt, welche eine Wahrheittabelle abbilden, wobei die Steuereingänge $\text{sel}_0 = a_i$, $\text{sel}_1 = b_i$ und $(\text{sel}_3, \text{sel}_2) = \text{code}[1:0]$ zur Bestimmung der zu erzeugenden Funktion dienen.

Erstellen Sie die Wahrheitstabelle der Logikfunktion, welche folgende Operationen im programmierbaren Logikblock erzeugt:

- $y_i = b_i$ für **`code = "00"`** Laden von der Wertes b
- $y_i = a_i * b_i$ für **`code = "01"`** UND Funktion zwischen a und b
- $y_i = a_i + b_i$ für **`code = "10"`** ODER Funktion zwischen a und b
- $y_i = a_i \oplus b_i$ für **`code = "11"`** exklusiv-ODER Funktion zwischen a und b

2.1 Implementierung und Simulation

Implementieren Sie die Schaltung der LU von Abbildung 1. Einige Schaltungselemente sind bereits im Block **`ALU/LU1`** vorhanden. Vervollständigen Sie die fehlende Eingangsverbindung des Multiplexers, die entweder mit einem logischen **`"0"`** oder einem logischen **`"1"`** verbunden sein soll. Diese Werte können durch die Elemente **`gates/logic0`** bzw. **`gates/logic1`** erzeugt werden.

Der Testbench **`ALU_test/LU8_tb`** ist bereits vorhanden, testet aber nicht alle Fälle. Der Testbench muss die Funktionalität der gesamten LU überprüfen.

Vervollständigen Sie die Schaltung des iterativen Blocks des **`ALU/LU1`**, der die 4 angegebenen Operationen ausführt.

Vervollständigen Sie die Teststimuli **`ALU_test/LU8_tester`** und überprüfen Sie die Funktion der gesamten **`ALU_test/LU8_tb`** mit der Simulationsdatei **`$SIMULATION_DIR/ALU1.do`**.

3 Arithmetische Einheit AU

Die Abbildung 2 zeigt die iterative Schaltung einer Arithmetischen Einheit (AU). Auch diese Schaltung besteht aus acht 1-Bit-Einheiten, die zu einer 8-Bit AU verbunden sind. Die logische Schaltung wird mit Multiplexern realisiert, welche eine Wahrheitstabelle darstellen.

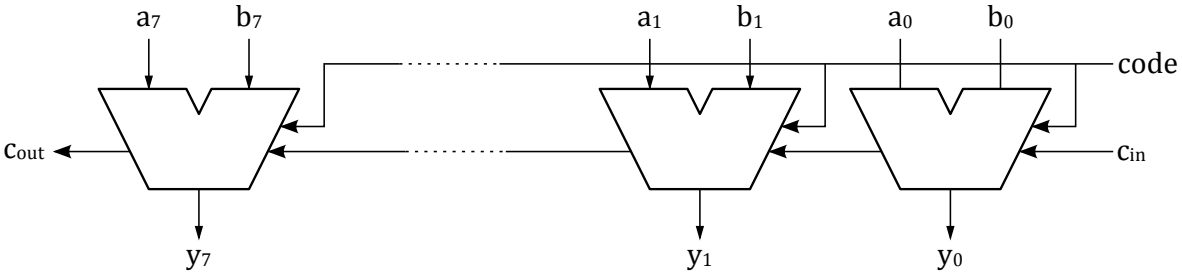


Abbildung 2 - Iterative Arithmetische Einheit

Die logische Schaltung jedes Iterationsblocks wird mit Multiplexern realisiert, welche eine Wahrheitstabelle abbilden. Die Steuereingänge $\text{sel}_0 = a_i$, $\text{sel}_1 = b_i$, $\text{sel}_2 = c_{\text{in}}$ und $\text{sel}_3 = \text{code}[0]$ dienen zur Bestimmung der zu erzeugenden Funktion.

Erstellen Sie die Wahrheitstabelle des iterativen Logikblocks **`ALU/AU1`** von Abbildung 2 für die folgenden Operationen auf Ganzzahlen:

- $y = a + b + c_{\text{in}}$ für **`code[0] = '0'`** ⇨ Addition
- $y = a - b - c_{\text{in}}$ für **`code[0] = '1'`** ⇨ Subtraktion

Unter der Annahme, dass eine Linksverschiebung einer Multiplikation mit zwei entspricht, schlagen Sie eine Erweiterung des iterativen Schaltkreises vor, um die Linksverschiebungsfunktion zu realisieren:

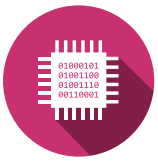
$$y = a \ll 1 = 2a = a + a \text{ für } \texttt{code[1:0] = '10'}$$

Erweitern Sie den Schaltkreis **`ALU/AU1`**, um die Linksverschiebungsoperation zu unterstützen, wenn der Wert von **`code[1:0] = '10'`**.

3.1 Implementierung und Simulation

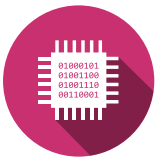
Implementieren Sie die Schaltung der AU von Abbildung 2, um alle zuvor genannten Funktionalitäten abzudecken. Einige Schaltungselemente sind bereits im Block **`ALU/AU8`** vorhanden.

Der Testbench **`ALU_test/AU8_tb`** ist bereits vorhanden, testet aber nicht alle Fälle. Der Testbench muss die Funktionalität der gesamten AU überprüfen.



Vervollständigen Sie die Schaltung des iterativen Blocks der **`ALU/AU1`**, welcher die 3 spezifizierten Operationen ausführt.

Vervollständigen Sie die Teststimuli **`ALU_test/AU8_tester`** und überprüfen Sie die Funktion des gesamten **`ALU_test/AU8_tb`** mit der Simulationsdatei **`$SIMULATION_DIR/ALU2.do`**.



4 Checkout Part 1

Dies ist das Ende des ersten Teils des Labors, Sie haben erfolgreich eine Logical Unit und eine Arithmetic Unit gebaut. Bevor Sie das Labor verlassen, stellen Sie sicher, dass Sie die folgenden Aufgaben erledigt haben:

- [ ] Schaltungsentwurf
  - [ ] Überprüfen Sie, ob die Blöcke **`ALU/LU8`** und **`ALU/AU8`** mit den in Abschnitt 2 und Abschnitt 3 genannten Funktionen entworfen und getestet wurden.
- [ ] Simulationen
  - [ ] Die spezifischen Tests der jeweiligen Testbänke (**`ALU_test/AU8_tb`** und **`ALU_test/LU8_tb`**) wurden an die Schaltung angepasst und gewährleisten einen vollständigen Test.
  - [ ] Die Schaltungen wurden erfolgreich mit den jeweiligen Testbänken **`ALU_test/AU8_tb`** und **`ALU_test/LU8_tb`** getestet.
- [ ] Dokumentation und Projektdateien
  - [ ] Stellen Sie sicher, dass alle Schritte (Entwurf, Konvertierungen, 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.

5 Arithmetisch und Logische Einheit ALU

Die Arithmetic and Logical Unit (ALU) wird durch die Kombination der bisher in Abschnitt 2 und Abschnitt 3 entwickelten Logical Unit und Arithmetic Unit realisiert, sie enthält auch eine zusätzliche *right shift* Operation siehe Abbildung 3. Sie wird den Assembler-Code des Xilinx PicoBlaze Befehlssatzes unterstützen Abbildung 3.

Damit der ALU die unterschiedlichen Operationen ausführt, müssen die Signale der Steuereinheit entsprechend gesetzt werden. Es handelt sich um $\mathrm{LU}_{\mathrm{Code}}[1:0]$, $\mathrm{AU}_{\mathrm{Code}}[1:0]$, $\mathrm{select}_{\mathrm{AU}}$, $\mathrm{select}_{\mathrm{SR}}$, $c_{\mathrm{in\_AU}}$. Diese Signale sind in Abbildung 3 sowie Tabelle 2 aufgeführt.

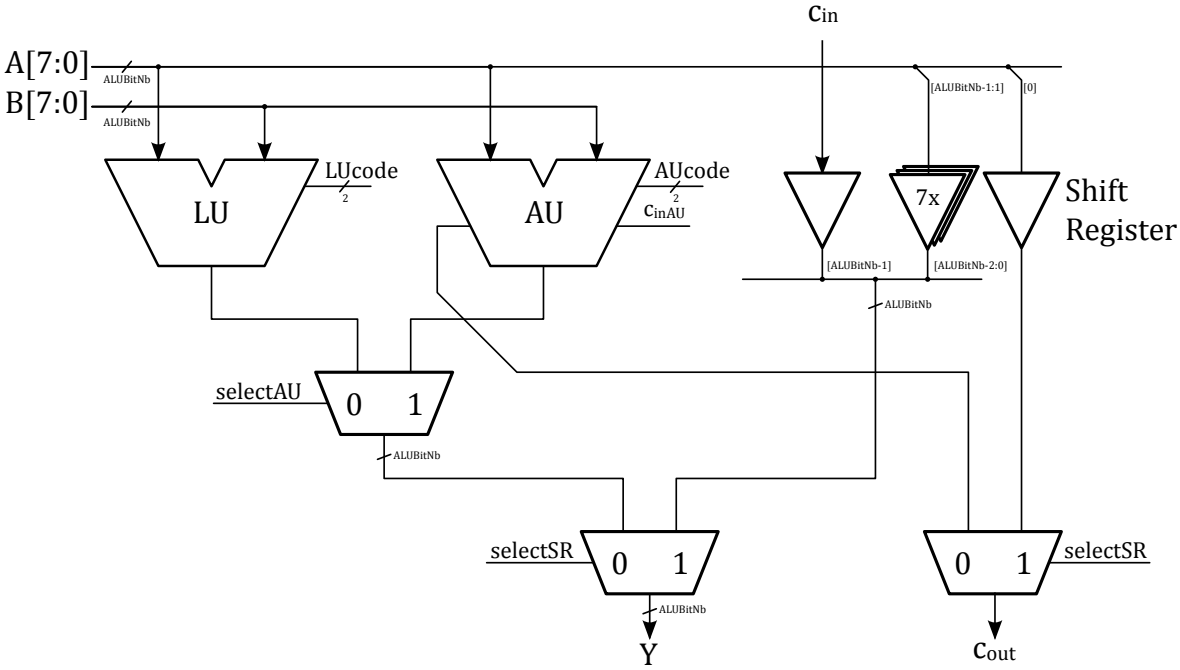
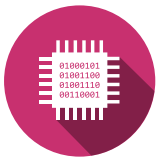


Abbildung 3 - Interner Aufbau ALU

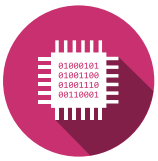
Untersuchen Sie, wie das Schieberegister in Abbildung 3 arbeitet und für welche Instruktionen es verwendet wird.



Die verschiedenen Operationen sind in der Tabelle 1 aufgelistet. Sie definiert, welcher OpCode welche Operation ausführt. Der OpCode ist ein 5-Bit Code, der in den Bits $I_{17} : I_{13}$ des Instruktionswortes liegt.

OpCode $I_{17} : I_{13}$	Assembler Instruction	ALU code Operation	Operation
00000	LOAD	LOAD B	$y = b$
00001	*ununsed*	-	-
00010	INPUT	LOAD B	$y = b$
00011	FETCH	LOAD B	$y = b$
00100	*ununsed*	-	-
00101	AND	AND	$y = a$ AND b
00110	OR	OR	$y = a$ OR b
00111	XOR	XOR	$y = a$ XOR b
01000	*ununsed*	-	-
01001	TEST	AND	$y = a$ AND b
01010	COMPARE	SUB	$y = a - b$
01011	*ununsed*	-	-
01100	ADD	ADD	$y = a + b$
01101	ADDCY	ADDCY	$y = a + b + c_{\text{in}}$
01110	SUB	SUB	$y = a - b$
01111	SUBCY	SUBCY	$y = a - b - c_{\text{in}}$
10000	SH / ROT	SHR	$a \gg 1$
10001	SH / ROT	SHL	$a \ll 1$
10010	*ununsed*	-	-
10011	*ununsed*	-	-
10100	*non-ALU*	-	-
...	...	...	...
11111	*non -LU*	-	-

Tabelle 1 - OpCode-Dekodierung in Bezug auf die ALU Operationen



5.1 Implementierung und Simulation

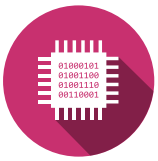
Für jede Instruktion sind alle Steuersignale im Schaltkreis **`ALU/ALU8`** wie die Multiplexer und die Befehle der LU und AU in der Tabelle 2 angegeben. Die Wahrheitstabelle basiert auf dem OpCode von Tabelle 1.

$\text{code}[4:0]$	$\text{LU}_{\text{code}}[1:0]$	$\text{AU}_{\text{code}}[1:0]$	$\text{select}_{\text{AU}}$	$\text{select}_{\text{SR}}$	$c_{\text{in\_AU}}$
00000					
00001					
00010					
00011					
00100					
00101					
00110					
00111					
01000					
01001					
01010					
01011					
01100					
01101					
01110					
01111					
10000					
10001					
10010					
10011					
10100					
...					
11111					

Tabelle 2 - ALU-Steuerungen

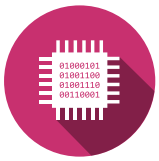
Vervollständigen Sie die Tabelle 2, die die Werte aller Steuersignale basierend auf der ALU Operation angibt, die durch das Signal $\text{code}[4:0]$ angezeigt wird.

Leiten Sie für jedes Steuersignal die boolesche Gleichung ab und implementieren Sie diese im Schaltkreis **`ALU/ALU8`**.



Überprüfen und falls mötig vervollständigen Sie die Teststimuli **`ALU_test/ALU8_tester`**. Führen Sie die Testbank **`ALU_test/ALU8_tb`** mit der Simulationsdatei **`$SIMULATION_DIR/ALU3.do`** aus und überprüfen Sie die Funktionalität der ALU.

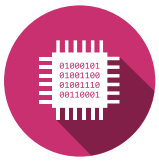
Die einzelnen Operationen wie **`ADD`**, **`OR`**, **`AND`**, ... wurden bereits im vorherigen Labor überprüft. Und müssen nicht erneut vollständig verifiziert werden.



6 Checkout Teil 2

Dies ist das Ende des zweiten Teils des Labors, Sie haben erfolgreich eine Arithmetic and Logical Unit des Xilinx PicoBlaze implementiert. Bevor Sie das Labor verlassen, stellen Sie sicher, dass Sie die folgenden Aufgaben abgeschlossen haben:

- [ ] Verständnis
  - [ ] Sie verstehen, wie die einzenen ALU Kompenenten funktionieren im speziellen die neue Schaltung des Shift Registers.
- [ ] Schaltungsentwurf
  - [ ] Überprüfen Sie, ob der Block **`ALU/ALU8`** mit den in Tabelle 1 genannten Funktionen entworfen und getestet wurde.
- [ ] Simulationen
  - [ ] Die spezifischen Tests der Testbank **`ALU_test/ALU8_tb`** wurden an die Schaltung angepasst und gewährleisten einen vollständigen Test.
- [ ] Dokumentation und Projektdateien
  - [ ] Stellen Sie sicher, dass alle Schritte (Entwurf, Konvertierungen, 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.



Glossar

***ALU* – Arithmetic and Logical Unit** 1, 6, 7, 8, 8, 9, 10

***AU* – Arithmetic Unit** 1, 1, 3, 3, 3, 3, 8

***LU* – Logical Unit** 1, 1, 2, 2, 2, 2, 8

PicoBlaze: PicoBlaze is a small, 8-bit microcontroller designed by Xilinx for use in FPGAs. It is often used in educational settings to teach basic microcontroller concepts. 6, 10