



Ausgabe: 30. November 2015

Abgaben	{		Theorie	06. Dezember 2015
			Praxis	13. Dezember 2015
			Rücksprache	14/15. Dezember 2015

### Aufgabe 1: Iterative Instanziierung (3 Punkte)

Sehr häufig wird eine gewisse Anzahl von Komponenten ähnlich mit anderen Bestandteilen verbunden. Um derartige strukturelle Beschreibungen zu vereinfachen, bietet VHDL die iterative Instanziierung (siehe [4]).

Exemplarisch soll ein n-Bit Addierer aus 1-Bit Volladdierern mit Hilfe des Statements `generate` beschrieben werden. Die `entity fulladd` des zu verwendenden Volladdierers ist vorgegeben. Schreiben Sie eine entsprechende `architecture behavioral` für die `entity adder` unter Verwendung des 1-Bit Volladdierers.

Die Abgabe muss als `adder.vhd` hochgeladen werden und soll erfolgreich kompilieren.

### Aufgabe 2: Adressdekoder (2 Punkte)

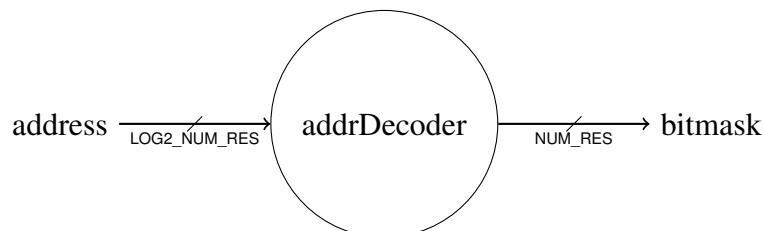


Abbildung 1: Entity `addrDecoder`

Für den folgenden Registerspeicher wird ein Dekoder benötigt. Dieser bekommt ein `LOG2_NUM_REGS` Bit breites Eingangssignal und liefert ein `NUM_REGS` Bit breites Ausgangssignal. Für jede mögliche Kombination soll jeweils nur ein einziges Bit des Ausgangssignals gesetzt werden (1-aus-*n*-Code). Implementieren Sie den vorgegebenen Dekoder in der Datei `addrDecoder.vhd` und validieren Sie Ihr Design mit Hilfe der vorgegebenen Testbench `addrDecoder_tb`. Weitere Informationen finden Sie unter anderem in [1].

Name	Typ	Art	Beschreibung
address	std_logic_vector (LOG2_NUM_REGS - 1 downto 0 )	in	Adresse, die umgewandelt werden soll.
bitmask	std_logic_vector (NUM_REGS - 1 downto 0 )	out	zugeordnete 1 aus $n$ Bitmaske

Tabelle 1: Entity-Ports

Name	Typ	Art	Beschreibung
NUM_REGS	integer	generic	Anzahl der Register
LOG2_NUM_REGS	integer	generic	Zweier-Logarithmus des oben genannten Parameters

Tabelle 2: Entity-Generics

### Aufgabe 3: Registerspeicher (5 Punkte)

Mit Hilfe des Dekoders aus der vorherigen Aufgabe sowie dem Register aus dem vorherigen Aufgabenblatt soll ein Registerspeicher für den MIPS-Prozessor (siehe [2]) zusammengesaltet werden.

Der Registerspeicher umfasst  $n$  Register (bzw. NUM\_REGS Register), welche mit Hilfe des Dekoders adressiert werden können. Beim Schreiben erfolgt die Auswahl des Registers über den Dekoder (siehe Abbildung 2). Für die zwei erforderlichen Leseports erfolgt die Auswahl durch zwei Multiplexer. Implementieren Sie den Registerspeicher in Form einer Netzliste unter Berücksichtigung der vorgegebenen Schnittstelle. Verwenden Sie die vorgegebene Datei `regFile.vhd` für die Implementierung und testen Sie ihre Implementierung mit der Testbench `regFile_tb`.

Die Bedeutung der einzelnen Signale ist in Tabelle 3 beschrieben. Beachten Sie dabei die folgenden Hinweise.

- Instanzieren Sie  $n$  Register (bzw. NUM\_REGS Register) (aus dem vorherigen Aufgabenblatt) iterativ. Legen Sie an jeden Registereingang die Eingangsdaten an.
- Nutzen Sie Ihre Kenntnisse über Arrays aus dem letzten Aufgabenblatt.
- Realisieren Sie die zwei Ausgangsports mit Hilfe von zwei Multiplexern (siehe Abbildung 3). Überlegen Sie sich eine geeignete (vor allem kurze) Realisierung der Multiplexer. Dazu dürfen Sie auch vom Konzept einer Netzliste abweichen.
- Um die ENABLE Eingänge der Register zu steuern, soll ebenfalls die iterative Instanziierung (siehe [3] und [4]) verwendet werden (siehe Abbildung 2). Zu beachten ist hierbei, dass lediglich bei aktiver REG\_WRITE Leitung des ausgewählten Registers, dieses zu überschreiben ist. Dies wird durch eine UND-Verknüpfung der REG\_WRITE Leitung mit der entsprechenden Leitung vom Dekoder erreicht.

Name	Typ	Art	Beschreibung
clk	std_logic	in	Takt für alle Register
rst	std_logic	in	Reset für alle Register
readAddr1	std_logic_vector (LOG2_NUM_REGS - 1 downto 0)	in	1. zu lesendes Register
readData1	std_logic_vector (REG_WIDTH - 1 downto 0)	out	Daten des Registers readAddr1
readAddr2	std_logic_vector (LOG2_NUM_REGS - 1 downto 0)	in	2. zu lesendes Register
readData2	std_logic_vector (REG_WIDTH - 1 downto 0)	out	Daten des Registers readAddr2
writeEn	std_logic	in	writeEnable für alle Register
writeAddr	std_logic_vector (LOG2_NUM_REGS - 1 downto 0)	in	Adresse des zu überschreibenden Registers
writeData	std_logic_vector (REG_WIDTH - 1 downto 0)	in	Zu schreibenden Daten

Tabelle 3: Entity-Ports

Name	Typ	Art	Beschreibung
NUM_REGS	generic	integer	Anzahl der Register
LOG2_NUM_REGS	generic	integer	Logarithmus zur Basis 2 von NUM_REGS
REG_WIDTH	generic	integer	Breite der Register

Tabelle 4: Entity-Generics

Name	Typ	Art	Beschreibung
reg_vect_debug	reg_vector_type	out	Debugport für die Testbench

Tabelle 5: Entity-Debug

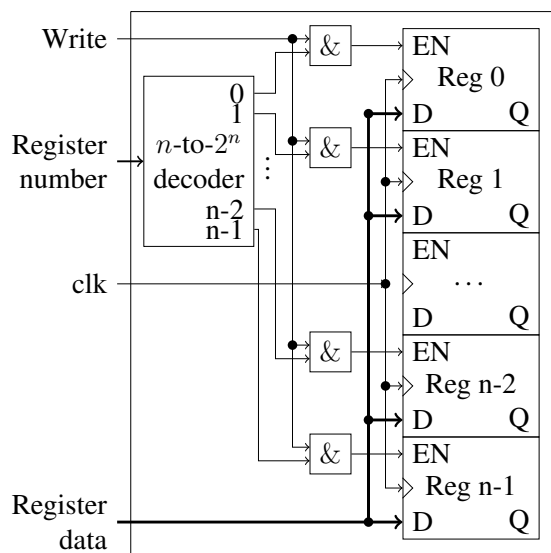


Abbildung 2: Die Realisierung des Schreibports des Registerspeichers (aus [2])

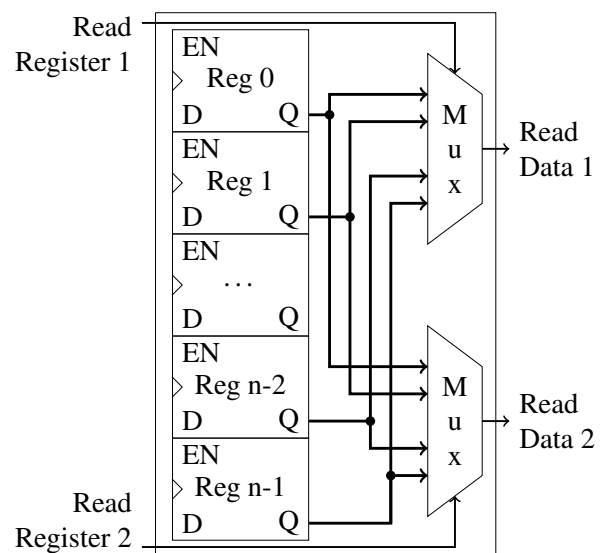


Abbildung 3: Die Realisierung der zwei Leseports des Registerspeichers (aus [2])

## Literatur

- [1] Peter J. Ashenden. *The Designer's Guide to VHDL, Volume 3, Third Edition*. Morgan Kaufmann, 3. edition, May 2008.
- [2] David A. Patterson and John L. Hennessy. *Rechnerorganisation und-entwurf*. Spektrum Akademischer Verlag, September 2005.
- [3] Hans-Ulrich Post. *Technische Grundlagen der Informatik 1*, 2009.
- [4] Jürgen Reichardt and Bernd Schwarz. *VHDL-Synthese: Entwurf digitaler Schaltungen und Systeme*. Oldenbourg, 4., überarbeitete auflage. edition, October 2007.