



Ausgabe: 04. Dezember 2017

Abgaben {		Theorie	11. Dezember 2017
		Praxis	17. Dezember 2017
		Rücksprache	18/19. Dezember 2017

Ab diesem Aufgabenblatt werden entwickelte Komponenten wiederverwendet. Sollte bei der Implementierung einer benötigten Komponente nicht die volle Punktzahl erreicht worden sein, **kann – aber soll auch nur dann** – die entsprechende Musterlösung aus der ROrgPrSimLib genutzt werden. Ändern Sie dazu bei der Instanziierung den Namen der Bibliothek (ROrgPrSimLib statt work).

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 mithilfe 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. Um dies zu testen, kann der Befehl `make all` im entsprechenden Verzeichnis ausgeführt werden.

Aufgabe 2: Adressdekoder (2 Punkte)

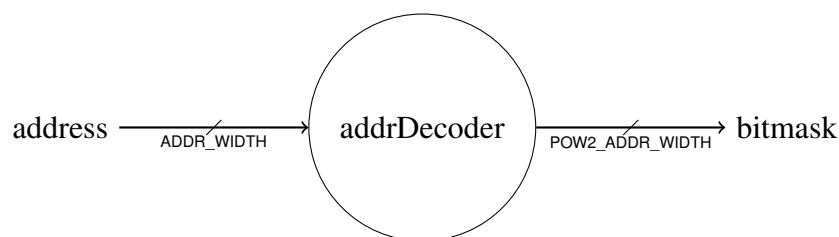


Abbildung 1: Entity `addrDecoder`

Für Registerspeicher der folgenden Aufgabe wird ein Dekoder benötigt. Dieser erhält ein `ADDR_WIDTH` Bit breites Eingangssignal und liefert ein `POW2_ADDR_WIDTH` Bit breites Ausgangssignal. Für jede

mögliche Kombination soll jeweils nur ein einziges Bit des Ausgangssignals gesetzt sein (1-aus- n -Code). Dabei handelt es sich genau um das m -te Bit handeln, wobei m die von `address` binär dargestellte Zahl ist. Implementieren Sie den Dekoder in der Architektur `behavioral` der Datei `addrDecoder.vhd` und validieren Sie Ihr Design mithilfe der vorgegebenen Testbench `addrDecoder_tb`. Weitere Informationen finden Sie unter Anderem in [1].

Name	Typ	Art	Beschreibung
address	std_logic_vector (ADDR_WIDTH - 1 downto 0)	in	Adresse, die umgewandelt werden soll.
bitmask	std_logic_vector (POW2_ADDR_WIDTH - 1 downto 0)	out	zugeordnete 1-aus- n -Bitmaske

Tabelle 1: Entity-Ports

Name	Typ	Art	Beschreibung
POW2_ADDR_WIDTH	integer	generic	Anzahl der möglichen Adressen, entspricht $2^{\text{ADDR_WIDTH}}$
ADDR_WIDTH	integer	generic	Anzahl der Bits des Adresssignals

Tabelle 2: Entity-Generics

Aufgabe 3: Registerspeicher (5 Punkte)

Mithilfe des Dekoders aus der vorherigen Aufgabe sowie dem Register aus dem vorherigen Aufgabenblatt soll ein Registerspeicher für den MIPS-Prozessor (siehe [2]) zusammengeschaltet 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 Architektur `structural` der vorgegebenen 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 beiden Ausgangsports mithilfe zweier Multiplexer. (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 iterative Instanziierung (siehe [3] und [4]) verwendet werden (siehe Abbildung 2). Zu beachten ist hierbei, dass lediglich

bei aktiver `writeEn`-Leitung das ausgewählte Registers zu überschreiben ist. Dies wird durch eine *UND*-Verknüpfung der `writeEn`-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	erstes zu lesendes Register
readData1	std_logic_vector (REG_WIDTH - 1 downto 0)	out	Daten des Registers <code>readAddr1</code>
readAddr2	std_logic_vector (LOG2_NUM_REGS - 1 downto 0)	in	zweites zu lesendes Register
readData2	std_logic_vector (REG_WIDTH - 1 downto 0)	out	Daten des Registers <code>readAddr2</code>
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: Debugports

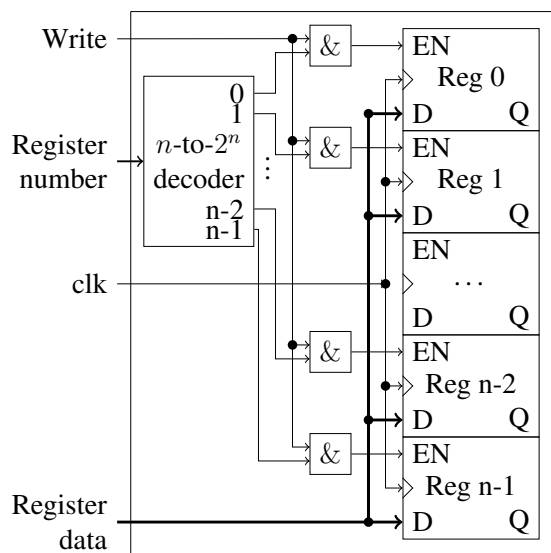


Abbildung 2: Die Realisierung des Schreibports des Registerspeichers (aus [2])

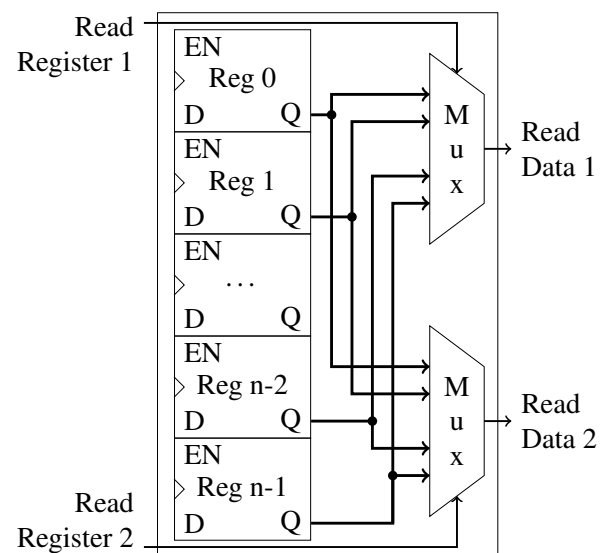


Abbildung 3: Die Realisierung der zwei Lesports des Registerspeichers (aus [2])

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.