



Ausgabe: 27. November 2017

Abgaben		Theorie	03. Dezember 2017
		Praxis	10. Dezember 2017
		Rücksprache	11/12. Dezember 2017

Dies ist das **1. Aufgabenblatt mit Theorie!**

Beachten Sie die **Abgabetermine** im Kopf dieser Seite.

Laden Sie Ihre gelösten Theorieaufgaben entsprechend der Aufgabenblatt-Nr. *X* und der Aufgaben-Nr. *Y* in die vorgegebene Datei `blattX/theorie/AufgabeY.txt` ins *GitLab* hoch.

Für die Definition von unterschiedlichen Typen und *Generics* wird im Praktikum die Datei `proc_config.vhd` verwendet, welche beim Simulieren ihrer Quelltexte automatisch durch das Makefile eingebunden wird. Da Sie einige der Typen, welche in der Datei definiert sind, verwenden werden, steht Ihnen die Datei `proc_config.txt` in den Vorgaben zur Verfügung, welche die Definitionen enthält. **Diese Datei dient nur als Informationsquelle und muss weder verändert noch zum Simulieren verwendet werden.**

Aufgabe 1: VHDL-Attribute (2 Punkte)

Mithilfe von Attributen kann in VHDL auf bestimmte Eigenschaften von Signalen zugegriffen werden. Informieren Sie sich zum Beispiel in [1] oder [5] über Attribute in VHDL und beantworten Sie die folgenden Fragen:

1. Wie greifen Sie auf das Attribut eines Signals zu? Geben Sie die Syntax anhand eines Beispiels an.
2. Nennen Sie drei der im Standard vordefinierten Attribute und erklären Sie kurz, was sie repräsentieren.
3. Über welche Attribute erhält man den Bereich für Signale des Typs `std_logic_vector`?
4. Wie kann man in VHDL mithilfe von Attributen eine positive bzw. negative Flanke eines Signals detektieren? Geben Sie die entsprechenden VHDL-Anweisungen an.

Aufgabe 2: Schaltwerke (4 Punkte)

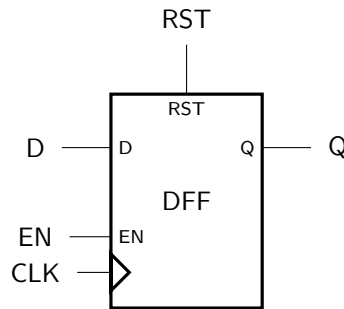


Abbildung 1: D-Flip-Flop inkl. rst und en

Name	Typ	Art	Beschreibung
D	std_logic	in	Datum, welches zum Speichern anliegt
CLK	std_logic	in	Takt, bei dessen steigender Flanke gespeichert werden soll
EN	std_logic	in	Flag, bei '1' soll gespeichert werden
RST	std_logic	in	Zurücksetzen des Flip-Flops bei '1'
Q	std_logic	out	Datum, welches gespeichert ist

Schaltwerke sind digitale Funktionsgruppen, deren Ausgangssignale nicht nur von aktuellen Eingangssignalen, sondern auch von den in der Vergangenheit aufgetretenen Zuständen abhängen. Taktflankengesteuerte D-Flip-Flops stellen die Grundlage der meisten Schaltwerke dar. Das Verständnis der Modellierung der D-FFs ist daher unverzichtbar. In dieser Übung soll dieses Verständnis erworben bzw. vertieft werden.

1. (2 Punkte) Implementieren Sie für die vorgegebene Entity-Spezifikation `dff` die Architektur `behavioral` in der Datei `dff.vhd`, welche das Verhalten eines flankengesteuerten D-FFs nachbildet.

Die **synchrone** Reaktion des FF-Ausgangs auf eine steigende Flanke wird innerhalb einer *if*-Anweisung beschrieben. Erinnern Sie sich dazu an die entsprechende Theorieaufgabe.

Testen Sie ihre Implementierung mit der Testbench `dff_tb`.

2. (2 Punkte) Taktflankengesteuerte Register lassen sich nach denselben Regeln entwerfen wie taktflankengesteuerte FFs. Die Datenein- und Ausgangssignale haben jedoch den Typ `std_logic_vector`. Implementieren Sie für die gegebene Entity-Spezifikation `reg` eines flankengesteuerten Registers mit generischer Breite die Architektur `behavioral` in der Datei `reg.vhd`. Das Register soll außerdem um ein **synchrones** Rücksetzsignal `rst` und um ein **synchrones** Aktivierungssignal `en` erweitert werden. Beide Signale sind high-aktiv, das heißt, ihre Funktion ist aktiviert, wenn an dem entsprechenden Eingang eine logische 1 anliegt. Wenn das Rücksetzsignal gesetzt ist, sollen alle Bits des Registers auf 0 gesetzt werden. Bei gesetztem Aktivierungssignal werden die Daten vom Eingang des Registers in die einzelnen D-Flip-Flops übernommen.

Validieren Sie das Verhalten des Registers mit der vorgegebenen Testbench `reg_tb`.

Aufgabe 3: RAM (4 Punkte)

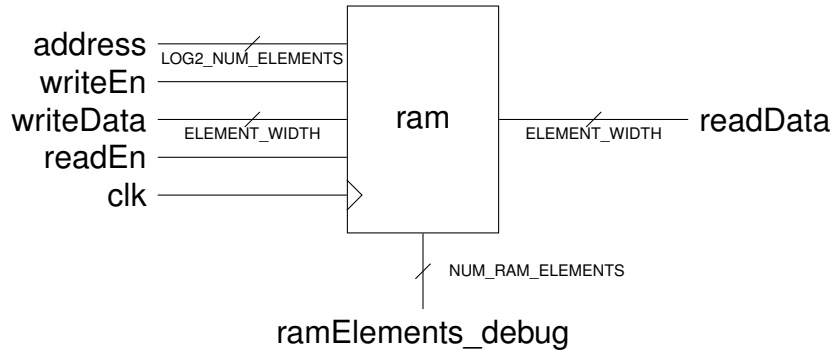


Abbildung 2: Entity ram

Name	Typ	Art	Beschreibung
clk	std_logic	in	Taktsignal zum synchronen Betrieb des RAMs
address	std_logic_vector (LOG2_NUM_ELEMENTS - 1 downto 0)	in	Adresse, an die geschrieben/gelesen werden soll
writeEn	std_logic	in	Flag, bei '1' soll geschrieben werden
writeData	std_logic_vector (ELEMENT_WIDTH - 1 downto 0)	in	Speicherelement, das geschrieben werden soll
readEn	std_logic	in	Flag, bei '1' soll gelesen werden
readData	std_logic_vector (ELEMENT_WIDTH - 1 downto 0)	out	Speicherelement, das gelesen wurde

Tabelle 1: Entity-Ports

Name	Typ	Art	Beschreibung
NUM_ELEMENTS	integer	generic	Anzahl der Speicherelemente des RAMs
LOG2_NUM_ELEMENTS	integer	generic	Zweier-Logarithmus des oben genannten Parameters
ELEMENT_WIDTH	integer	generic	Breite eines einzelnen Speicherelements

Tabelle 2: Entity-Generics

Name	Typ	Art	Beschreibung
ramElements_debug	ram_elements_type	out	Debug-Port für den Zugriff auf den RAM-Inhalt

Tabelle 3: Entity-Debug

Für sehr viele Anwendungen wird Speicher zum Beispiel in Form von externen RAM-Bausteinen verwendet, da die Kapazität der RAM-Blöcke innerhalb eines FPGA begrenzt ist. Um während der Designphase einen derartigen Baustein simulieren zu können, ist ein entsprechendes Modell zu implementieren.

Eine prinzipielle Speicherkomponente mit den notwendigsten Signalen ist der Abbildung 2 zu entnehmen. Soll ein Datum gelesen werden, so muss das `readEn`-Signal gesetzt werden. Beim Schreiben ist dementsprechend die `writeEn`-Leitung zu aktivieren. Bei beiden Aktionen wird die Adresse durch die Daten auf den `address`-Leitung definiert. Gleichzeitiges Schreiben und Lesen ist möglich. In diesem Fall soll der Schreibvorgang zuerst erfolgen, damit die aktualisierte Speicherzelle noch in demselben Takt gelesen werden kann. Die Breite der zu speichernden Daten wird generisch über den Parameter `ELEMENT_WIDTH` gesteuert. Die Anzahl der Speicherelemente ist dem Parameter `NUM_ELEMENTS` zu entnehmen. Die notwendige Adressbreite wird über `LOG2_NUM_ELEMENTS` angegeben. Das Schreiben soll synchron erfolgen. Der `ramElements_debug`-Port ermöglicht der Testbench das Auslesen des RAM-Inhalts, ohne dass der Leseport funktionieren muss. Dieser Debug-Port muss in ihrer Beschreibung nicht weiter berücksichtigt werden.

1. Implementieren Sie das Speichermodell in der Architektur `behavioral` für die vorgegebene Entity-Spezifikation `ram` in der Datei `ram.vhd` und testen Sie Ihr Design mit der zur Verfügung gestellten Testbench `ram_tb`.

Literatur

- [1] Gunther Lehmann, Bernhard Wunder, and Manfred Selz. *Schaltungsdesign mit VHDL*, 1994.
- [2] Mentor Graphics Corporation. *ModelSim SE Reference Manual*, 6.4a edition.
- [3] Mentor Graphics Corporation. *ModelSim SE Tutorial*, 6.4a edition.
- [4] Mentor Graphics Corporation. *ModelSim SE User's Manual*, 6.4a edition.
- [5] A. Mäder. *VHDL Kompakt*.