Christian-Albrechts-Universität zu Kiel
Institut für Informatik

Lehrstuhl für Technische Informatik
Prof. Dr. Manfred Schimmler
Dipl.-Inf. Lars Wienbrandt

FPGA-Entwurf mit VHDL

Sommersemester 2011

Serie 6

Abgabe: bis Fr. 10.06. um 10 Uhr im Schrein oder per Mail an lwi@informatik.uni-kiel.de.

Hinweis: Die Übung am Fr. 03.06. fällt aus. Sie haben daher für diese Serie zwei Wochen Bearbeitungszeit.

Aufgabe 1

Das Datenblatt `xapp463` beschreibt die Nutzung von Block-RAM in Spartan3-FPGAs.

a) Wie ist die Kapazität eines Spartan3-RAM-Blocks und wieviele Blöcke besitzt der Spartan3-400 auf dem Memec FPGA-Board?

b) Wieviele Taktzyklen benötigt das Auslesen eines beliebigen Datums aus dem Speicher?

c) Wie ist die maximale Taktfrequenz des Block-RAMs in Spartan3-FPGAs?

d) Was unterscheidet die Konfigurationsmöglichkeiten `READ_FIRST`, `WRITE_FIRST` und `NO_CHANGE` für den `WRITE_MODE`?

e) Welche Konfliktsituationen können bei der Verwendung von Dual-Port-RAM auftreten?

10 Punkte

Aufgabe 2

Das Sieb des Eratosthenes ist ein einfaches Verfahren zur Bestimmung von Primzahlen. Dabei werden in einem Zahlenfeld von 2 bis zu einer Schranke S alle Vielfache von noch unmarkierten Zahlen markiert, angefangen bei der 2. Am Ende des Verfahrens sind alle in dem Feld vorkommenden Primzahlen durch die unmarkierten Zahlen bestimmt.

Informieren Sie sich über das genaue Verfahren des Sieb des Eratosthenes.

Für die Darstellung des Zahlenfeldes soll nun ein Block-RAM eines Spartan-3 der begrenzende Faktor sein.

a) Für wieviele Zahlen kann komfortabel eine Markierung in einem RAM-Block gespeichert werden? Wie muss dieser konfiguriert sein? Hier genügt der enstprechende Instanzname eines Single-Port-RAM.

b) Wie groß ist dann die Schranke S, wenn gerade Zahlen implizit ausgelassen werden sollen und die erste potentiell markierbare Zahl die 3 sein soll? Wieviele Primzahlen werden in diesem Zahlenbereich erwartet?

Hinweis: Die Anzahl der Primzahlen $\pi(S)$ können Sie natürlich direkt bestimmen, wenn Sie möchten. Die Näherung nach der Verbesserung des Primzahlsatzes durch *Adrien-Marie Legendre* trifft's aber auch sehr gut.

c) Wie ist die benötigte Wort- und Adressbreite für ein geeignetes RAM und wieviele RAM-Blöcke werden demnach benötigt um alle in dem durch b) eingeschränkten Zahlenbereich vorhandenen Primzahlen in einer Liste zu speichern? Wie ist dessen Konfiguration (Instanzname)? Alle Zahlen sollen mit der gleichen Wortbreite gespeichert werden.

10 Punkte

Aufgabe 3 – Programmieraufgabe

Programmieren Sie das in Aufgabe 2) beschriebene Sieb des Eratosthenes in VHDL. Ihr Design soll folgende Schnittstelle besitzen:

clk: Taktsignal

sync_reset: Taktsynchroner Rücksetzeingang. Bei "1" sollen alle Ausgaberegister zurückgesetzt werden. Ihr Design beginnt mit der Durchführung des Algorithmus.

ready: Zeigt an, ob ihr Algorithmus alle Primzahlen in das BRAM geschrieben hat. Damit ist auch die am Ausgang gelesene Primzahl valide.

read_prime: Gibt an, die wievielte Primzahl gelesen werden soll. Informatiker fangen natürlich bei Null an zu zählen... ;-)

prime: Zeigt die aus dem BRAM gelesene Primzahl an. Die Ausgabe erfolgt um einen Takt verzögert zu `read_prime`.

```
entity eratosthenes is
port (
  clk : in std_logic;
  sync_reset : in std_logic;
  ready : out std_logic;
  read_prime : in std_logic_vector(XXXX downto 0);
  prime : out std_logic_vector(YYYY downto 0)
);
end eratosthenes;
```

a)

Verwenden Sie für den RAM-Block, der die Markierungen enthalten soll, ein Xilinx-Primitive. Der RAM-Block soll als Single-Port instanziiert werden.

b)

Für die Liste der Primzahlen generieren Sie den erforderlichen Speicher mit Hilfe des Core-Generators. Hier dürfen Sie ein "Simple Dual-Port-RAM" verwenden.

c)

Synthetisieren und Implementieren Sie Ihr Design. Für die Implementierung wird ein Timing-Constraint benötigt. Binden Sie daher die UCF-Datei auf der Homepage mit in Ihr Projekt ein. Diese enthält ein sehr loses Consraint. Achten Sie darauf, dass diese UCF-Datei in der Design-Hierarchie direkt unter der Top-Level-Entity erscheint.

Wie ist die maximale Frequenz Ihres Designs nach der Synthese und nach PAR?

d)

Simulieren Sie Ihr Design. Wieviele Primzahlen findet Ihr Verfahren? Was ist die letzte gefundene Primzahl? Wieviel Zeit benötigt Ihr Design, wenn eine Taktperiode $10ns$ beträgt? (Sie können diesen Wert aus der Simulation entnehmen.)

Hinweis 1: Da gerade Zahlen nicht im Zahlenfeld vorgesehen sind, können natürlich auch nur ungerade Vielfache einer Primzahl markiert werden. Machen Sie sich vorher Gedanken über ein korrektes Mapping zwischen der zu markierenden Zahl und der entsprechenden Adresse im Zahlenfeld.

Hinweis 2: Allgemein genügt es mit dem Markieren der Vielfachen einer Zahl mit dessen Quadrat zu beginnen. Auf einem FPGA benötigt die Berechnung des Quadrates aber bereits zusätzliche Ressourcen, und für dieses Beispiel wird es nicht einmal einen nennenswerten Performance-Vorteil erzeugen. Demnach beginnen Sie mit den Markierungen der Vielfachen bitte beim Dreifachen.

Hinweis 3: Das Block-RAM kann nicht durch ein Reset-Signal zurückgesetzt werden, d.h. nach einem synchronen Reset sind die Inhalte noch immer vorhanden. Achten Sie in Ihrem Verfahren also darauf im Zahlenfeld nicht auch Ihre Primzahlen zu markieren!

Hinweis 4: Auch wenn alle geraden Zahlen im Vornherein nicht behandelt werden sollen, soll die 2 als erste Primzahl dennoch der Vollständigkeit halber beim Lesen der 0-ten Primzahl ausgegeben werden. Sie müssen diese aber nicht im BRAM ablegen.

50 Punkte

Aufgabe 4 – Programmieraufgabe

Untersuchen Sie verschiedene Addierertypen auf dem Spartan3-400-FPGA. Alle zu implementierenden Addierer sollen eine generische Wortbreite besitzen und die Ausgabe soll getaktet aus einem Register erfolgen, d.h. die Ausgabe erfolgt um einen Takt versetzt nach der Eingabe. Ein auftretendes Carry-Out darf vernachlässigt werden.

```
entity rcadd is
generic (
  WIDTH : positive := 64
);
port (
  clk : in std_logic;
  A : in std_logic_vector(WIDTH-1 downto 0);
  B : in std_logic_vector(WIDTH-1 downto 0);
  S : out std_logic_vector(WIDTH-1 downto 0)
);
end rcadd;
```

a) Wie ist die Zeit- und Flächenkomplexität eines Ripple-Carry-Addierers und eines Carry-Look-Ahead-Addierers?

b.1) Implementieren Sie einen Ripple-Carry-Addierer mit generischer Wortbreite mit Hilfe des Operators `+`.

b.2) Implementieren Sie einen Ripple-Carry-Addierer mit generischer Wortbreite durch Instanziierung mehrerer Volladdierer-Einheiten ohne den Operator `+`.

c) Die zwei Implementierungen aus b) sollen nun miteinander verglichen werden. Schreiben Sie je eine Top-Level-Entity, die ihre Addierer-Einheit umschließt. Um dabei vom Routing der Ein- und Ausgabepins unbeeinflusst zu bleiben benutzen Sie bitte die Einheiten `Serializer` und `Deserializer` von der Lehrstuhl-Homepage. Hierbei handelt es sich um leicht optimierte Versionen gegenüber der Musterlösung zur Aufgabenserie 3. Schließen Sie diese Einheiten vor und hinter die Addiereinheit an, sodass die Ein- und Ausgaben bitseriell erfolgen. Der Addierer addiert folglich die Ausgaben aus zwei Deserializern und gibt das Ergebnis an einen Serializer weiter.

Für die genaue Bestimmung der Timings wird wieder ein "Timing Constraint" benötigt. Binden Sie hierzu die auf der Lehrstuhl-Homepage angegebene UCF-Datei in Ihr Projekt mit ein. Achten Sie darauf, dass die UCF-Datei in der Hierarchie immer direkt unter der Top-Level-Entity ist.

Hinweis: Die Signale `D_new_in` der beiden Deserializer-Einheiten können der Einfachheit halber konstant auf `1` gesetzt werden, das Signal `D_new_out` des Serializers darf offen bleiben.

```
entity rcadd_top is
generic (
  WIDTH : positive := 64
);
port (
  clk : in std_logic;
  A : in std_logic;
  B : in std_logic;
  S : out std_logic
);
end rcadd_top;
```

Implementieren Sie Ihre Designs (einschließlich PAR ("Place & Route")) mit den Wortbreiten 64*bit*, 128*bit* und 256*bit*. Notieren Sie in einer Tabelle die Ergebnisse für die maximale Taktfrequenz (nach der Synthese und nach PAR) und die Anzahl der verwendeten Flipflops, LUTs und Slices.

d) Stimmen die Ergebnisse aus c) mit den Erwartungen aus a) überein? Begründen Sie.

e) Implementieren Sie Ihr Design b.1) wie in c) mit einer Wortbreite von 129*bit*. Notieren Sie sich die Taktfrequenz nach der Synthese und nach PAR. Was fällt Ihnen auf, wenn Sie das Ergebnis mit der Implementierung mit 128*bit* Wortbreite vergleichen? Begründen Sie.

30 Punkte

Zusatzaufgabe 5 – Programmieraufgabe

Fügen Sie Ihrer Untersuchung aus Aufgabe 4 einen Carry-Look-Ahead-Addierer hinzu.

a) Wie ist die Zeit- und Flächenkomplexität eines Carry-Look-Ahead-Addierers?

b) Versuchen Sie einen Carry-Look-Ahead-Addierer mit generischer Wortbreite zu implementieren, d.h. Sie dürfen folgende Formel zur Vorberechnung der Überträge verwenden:

$$c_{i+1} = g_i + p_i c_i$$

Warum führt dies nicht zu einem "echtem" Carry-Look-Ahead-Addierer?

c) Schreiben Sie eine Top-Level-Entity genau wie in 4c) und fügen Sie Ihre Ergebnisse ebenfalls für die Wortbreiten 64*bit*, 128*bit* und 256*bit* Ihrer Tabelle hinzu.

d) Was fällt Ihnen nun hinsichtlich des Aufgabenteils a) und im Vergleich zu den Ergebnissen aus Aufgabe 4 auf?

20 Zusatzpunkte

Die Abgabe der Programmieraufgaben soll bitte folgendermaßen erfolgen:

1. **Verschieben Sie oder benennen Sie eine evtl. generierte Konfigurationsdatei zunächst um, damit sie beim Cleanup nicht gelöscht wird!**
2. Dann führen Sie in ISE "`Project` → `Cleanup Project Files`" aus.
3. Verpacken Sie den Projektordner in ein gepacktes Archiv (`.zip`, `.tar.gz` o.ä.).
4. Senden Sie das Archiv per Mail an `lwi@informatik.uni-kiel.de`