Aufgabe 4 (10 + 40 Punkte)

Lernziele: Im Rahmen dieser Aufgabe sollen Sie Ihre Kenntnisse aus den vorherigen Aufgaben vertiefen und Ihr Wissen um neue Aspekte erweitern, vor allem aus dem Bereich der fließbandorganisierter Schaltwerke, indem Sie sich mit folgenden Punkten befassen:

a) Entwurf fließbandorganisierter und durchsatzoptimierter Schaltwerke mit selbst definierten Handshake-Protokollen und Schnittstellen,
b) synthesegerechte Verhaltens- und Strukturbeschreibung komplexerer Schaltwerke mit Hilfe von PROCESS-, IF- und CASE-Anweisungen,
c) Strukturbeschreibung im hierarchischen Entwurf durch Instanzierung von bereitgestellten, selbst entwickelten und primitiven Komponenten, wie Multiplizierer und Dual-Port-Speicher (RAM- und ROM-Blöcke).

Aus der 1. Aufgabe übernehmen Sie die Komponente zur Darstellung von Zahlen auf der 4-stelligen 7-Segmentanzeige, und aus der 2. Aufgabe verwenden Sie die Schaltung zur Entprellung von Tasten und ggf. den Universalzähler. Die 3. Aufgabe stellt Ihnen ein Schaltwerk zur Berechnung des Skalarproduktes bereit. Dieses Schaltwerk kann beim Bedarf modifiziert oder erweitert werden.

Hinweise:

1. Vorgegebene Schnittstellen oder bereitgestellte Komponenten dürfen weder geändert noch durch andere ersetzt werden.
2. Der Takt in synchronen Systemen ist eine „unantastbare“ Größe und darf auf keinen Fall über Gattern mit anderen Signalen verknüpft werden. So etwas ist ein schlechter Programmierstil und Hinweis auf mangelnde Kenntnisse der Digitaltechnik. Der Takt darf nur direkt an die Clock-Eingänge von Flipflops angeschlossen werden.

Aufgabenstellung. Im Rahmen dieser Aufgabe ist eine *durchsatzoptimierte* und *fließbandorganisierte* Schaltung in VHDL zu beschreiben und für den FPGA-Baustein XC3S200 auf dem Entwicklungs-Board zu synthetisieren. Diese Schaltung soll den Algorithmus zur Matrixmultiplikation als kooperierende Schaltwerke, bestehend aus einem fließbandorganisierten Rechenwerk und einem Steuerwerk, realisieren. Dieser Algorithmus, der als Programmausschnitt in Bild 1 zu sehen ist, beruht auf der iterativen Berechnung von Skalarprodukten.

Zur Lösung der Aufgabe sind zwei 16x16-Matrizen mit vorzeichenbehafteten, ganzzahligen 16-Bit-Werten im 2er-Komplementformat vorgegeben. Diese Werte sind, so wie das in der Programmiertechnik vom Prinzip her üblich ist, zeilenweise in einem eindimensionalen und linearen Speicher abgelegt. Dieses Prinzip ist am Beispiel einer 4x4-Matrix in Bild 2 veranschaulicht.

```
process
   constant N: natural := 16;
   type matrix is array(0 to N-1, 0 to N-1) of integer;
   constant A: matrix := ( ... );
   constant B: matrix := ( ... );
   variable S: integer;
   variable C: matrix := (others => (others => 0));
begin
   for i in 0 to N-1 loop
      for j in 0 to N-1 loop
         S := 0;
         for k in 0 to N-1 loop
            S := S + A(i, k) * B(k, j);
         end loop;
         C(i, j) := S;
      end loop;
   end loop;
   wait;
end process;
```

Bild 1: Algorithmus zur Matrixmultiplikation in VHDL. Die Initialisierung der beiden Eingangsmatrizen A und B ist hier der Übersichtigkeit halber ausgeblendet.

Die beiden 16x16-Matrizen sind in einem ROM-Block abgelegt, der als nur lesbarer 16-Bit-Dual-Port-Speicher mit 1024 Speicherzellen konfiguriert ist, und als Datei *rom_block.vhd* zur Verfügung steht. Jede Matrix umfaßt 256 vorzeichenbehaftete 16-Bit-Dualzahlen im 2er-Komplementformat. Beide Matrizen sind im ROM-Block zweilenweise so angeordnet, daß die erste Matrix die ersten 256 Speicherzellen ab der Adresse 0x0000 bis 0x00FF belegt, und die zweite Matrix die darauffolgenden 256 Speicherzellen ab der Adresse 0x0100 bis 0x01FF. Weil der ROM-Block als Dual-Port-Speicher konfiguriert ist, ist es auch möglich, und im Rahmen dieser Aufgabe durchaus wünschenswert, parallel und gleichzeitig auf die Elemente beider Matrizen lesend zuzugreifen.

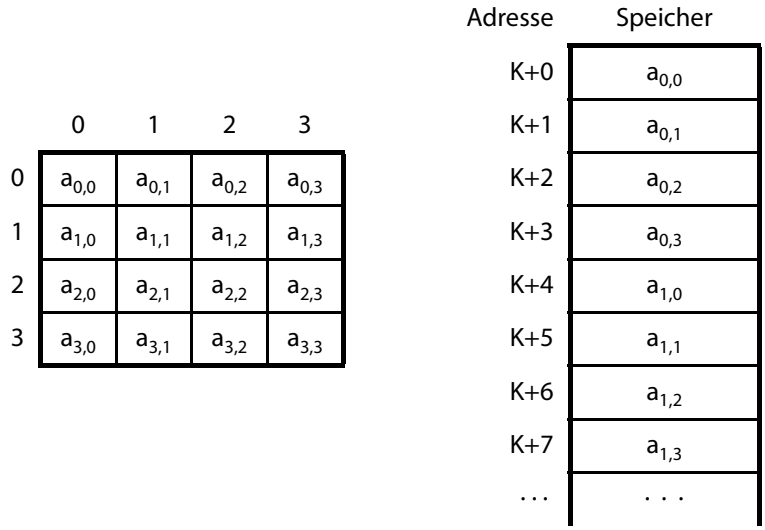

Bild 2: Eine 4x4-Matrix und der Auszug aus einem Speicher. K ist die Basis-/Anfangsadresse des Speicherbereiches, in dem die Matrix abgelegt ist. Der Offset in der Adresse bezieht sich auf einen wortweise organisierten Speichert, also keine Byte-Adressierung.

Die Werte in den beiden 16x16-Matrizen wurden so gewählt, daß das Endergebnis aus der Berechnung jeden Skalarproduktes immer als vorzeichenbehafteter, ganzzahliger 16-Bit-Wert im 2er-Komplementformat darstellbar ist, so daß man die Ergebnisse in einem 16-Bit-Speicher ablegen kann. Die Ergebnisse aus der Matrixmultiplikation sollen in einer dritten Matrix, der sog. Ergebnismatrix, in einem RAM-Block abgelegt werden, und zwar ebenfalls zeilenweise, beginnend mit der Adresse 0x0000. Dieser RAM-Block wurde als beschreibbarer Dual-Port-Speicher konfiguriert und steht als Datei *ram_block.vhd* zur Verfügung.

Die Schnittstelle der Schaltung, die die Matrixmultiplikation realisieren soll, ist als die Datei *core.vhd* vorgegeben und in Bild 3 zu sehen. Neben den Standard-Signalen rst, clk und swrst sind in der Schnittstelle auch zwei Handshake-Signale strt und rdy vorhanden. Mit dem Signal strt wird der Algorithmus gestartet. Mit dem Signal rdy wird das Ende der Berechnung angezeigt. Das Signal strt wird auf dem Entwicklungsboard über die Entprellschaltung von der Taste btn_0 abgeleitet. Mit dem Niederdrücken dieser Taste soll der Berechnungsalgorithmus gestartet werden. Das Signal rdy wird auf dem Entwicklungsboard an die Leuchtdiode ld_0 angeschlossen. Sie zeigt also mit ihrem Leuchten das Ende des Berechnungsalgorithmus an. In der Reset-Phase, beim Start und während der Ausführung des Berechnungsalgorithmus wird sie bzw. bleibt sie ausgeschaltet.

```
entity core is
   generic(RSTDEF: std_logic := '0');
   port(rst:   in  std_logic;                      -- reset,          RSTDEF active
        clk:   in  std_logic;                      -- clock,          rising edge
        swrst: in  std_logic;                      -- software reset, RSTDEF active
        strt:  in  std_logic;                      -- start,          high active
        rdy:   out std_logic;                      -- ready,          high active
        sw:    in  std_logic_vector( 7 downto 0);  -- address input
        dout:  out std_logic_vector(15 downto 0)); -- result output
end core;
```

Bild 3: Schnittstelle der Schaltung zur Matrixmultiplikation.

Außerdem enthält diese Schnittstelle zwei Signalvektoren sw und dout. Der Signalvektor sw ist mit den Schiebeschaltern $sw_7..sw_0$ auf dem Entwicklungsboard zu verbinden. Mit der Hilfe der Schiebeschalter ist es möglich, beliebige 8-Bit-Werte einzustellen. Diese Werte werden als Adressen aus dem Bereich 0x00 bis 0xFF interpretiert, und dienen dazu, über einen der beiden Eingangsports des RAM-Blocks den Inhalt von Speicherzellen auszulesen. Der Inhalt derjenigen Speicherzelle, die durch eine solche Adresse angesprochen wird, wird über den Signalvektor dout zurückgegeben. Dieser Wert soll anschließend mit Hilfe der Komponente hex4x7seg aus der 1. Aufgabe auf der 4-stelligen 7-Segmentanzeige dargestellt werden.

In Bild 4 ist die vereinfachte Struktur eines modular aufgebauten Rechenwerks zur Matrixmultiplikation dargestellt. Das Rechenwerk setzt sich aus einem Adressengenerator, einem Dual-Port-ROM-Block mit den beiden 16x16-Matrizen, einem vorzeichenbehafteten 18x18-Bit-Multiplizierer, einem vorzeichenbehafteten 44-Bit-

Addierer mit einem Akkumulator und einem Dual-Port-RAM-Block für die Ergebnismatrix zusammen. Der Multiplizierer und der Addierer mit dem Akkumulator bilden eine sog. MAC-Einheit, mit deren Hilfe das Skalarprodukt zweier Vektoren berechnet wird.

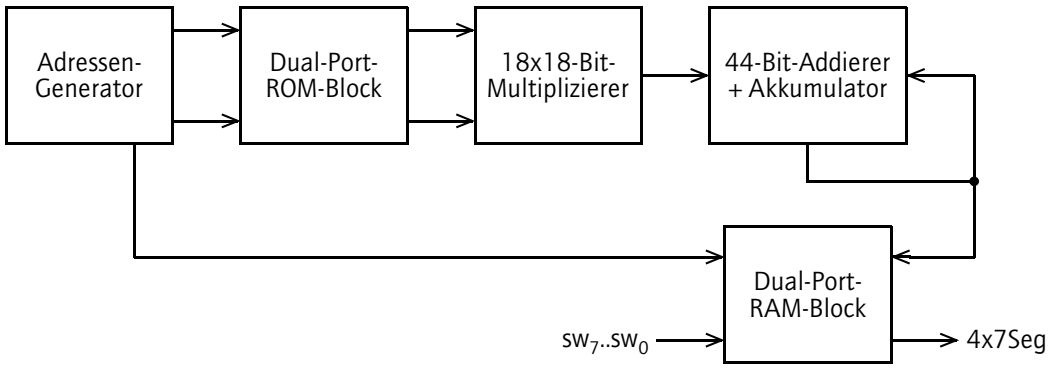

Bild 4: Schematischer Aufbau des fließbandorganisierten Rechenwerks zur Matrixmultiplikation.

Neben der Schaltungskomplexität und der minimalen Periodendauer wird zur Bewertung der Lösung auch der durch diese Schaltung erreichte Durchsatz herangezogen. Da die Messung des Durchsatzes in der Schaltung umständlich ist, wird der Durchsatz in der Simulationsumgebung ermittelt. Zu diesem Zweck steht die Testumgebung mit der Datei *core_tb.vhd* zur Verfügung. Sowohl in der Simulation mit der Messung des Durchsatzes als auch in der Synthese sind dieselben VHDL-Quelltextdateien zu verwenden.