Übungen zu Architektur Eingebetteter Systeme

Blatt 7

03./10.07.2009

Teil 1: Grundlagen

1.1: Verhaltens- und Strukturbeschreibung

VHDL bietet die Möglichkeit sowohl eine strukturelle Beschreibung als auch eine Verhaltensbeschreibung eines Bausteins zu implementieren, um dessen Ein- und Ausgabeverhalten zu spezifizieren. Die strukturelle Beschreibung eines Bausteines gibt an, wie viele Instanzen der einzelnen Komponenten zur Realisierung dieses Bausteines benutzt werden, wie diese Komponenten untereinander verdrahtet sind und wie diese mit den Ein- und Ausgängen des Bausteines verdrahtet sind. Die Verhaltensbeschreibung dagegen beschreibt einen Baustein nicht durch seine Komponenten sondern durch gleichzeitig nebeneinander laufende Prozesse, welche die Belegung der Signale des Bausteines funktional bestimmen. Prozesse beschreiben damit wie sich Signale in Abhängigkeit anderer Signale verhalten.

Weitere Informationen finden Sie beispielweise in: Molitor und Ritter *VHDL - Eine Einführung* oder Online unter `http://tams-www.informatik.uni-hamburg.de/vhdl/`

1.2: Hardwarebeschreibung

Bei einer Hardwareimplementierung gibt es die Vorteile, einerseits eine direkte Verdrahtung von Signalen vorzunehmen und damit Abläufe zu parallelisieren. Dieses beutetet wiederum, dass eine sequentielle Beschreibung, wie sie in den meisten Software-Programmiersprachen stattfindet, nicht direkt in VHDL umgesetzt werden kann. Hierzu muss selbst eine Taktung eingeführt werden. Wie dieses im einzelnen aussieht, hängt von der Situation ab. Typische Konzepte sind die State-Machine oder die Benutzung der Clock in Zusammenhang mit einem Counter. Als Grundregel können Sie zuerst einmal davon ausgehen, dass alle innerhalb eines Prozesses beschriebenen Aktionen parallel ausgeführt werden und die Belegung der Signale sowie Ein- und Ausgänge erst am Ende des Prozesses stattfindet. Somit ist folgende Beschreibung in VHDL möglich:

```
1  ARCHITECTURE Tauschen OF Beispiel IS
2  SIGNAL a,b: STD_LOGIC;
3  BEGIN
4       PROCESS(a,b)
5       BEGIN
6            a <= b;
7            b <= a;
8       END PROCESS;
9  END Tauschen;
```

Hier findet keine sequenzielle Verarbeitung statt, sondern a wird gleichzeitig mit b vertauscht, so dass beide Signale nach dem Durchlaufen verschiedene Werte haben können.

1.3: Realisierung des CRC

Bei der Realisierung haben Sie mehrere Möglichkeiten. Es ist Ihnen freigestellt eine eigene Implementierung zu verfolgen. Sollten Sie Schwierigkeiten bei der Umsetzung haben, soll Ihnen die folgende Beschreibung die Umsetzung etwas erleichtern.

Nach der Beschreibung der Top-Level-Entity sollten Sie sich bei der Architekturbeschreibung Hilfssignale definieren, die Sie mit den Signalen des Board verbinden können. Die können Testweise auch einen der Knöpfe als Clock verwenden und die LEDs zur Ausgabe von bestimmten Signalen verwenden.

Sowohl bei der Verhaltens- als auch der Strukturbeschreibung werden Sie einen Prozess benötigen, der abhängig vom Clocksignal einen Counter hochzählt. Weiterhin sollte der Counter nur hochzählen, wenn das Enable-Signal Eins ist. Auch sollte der Counter zurück auf Null gesetzt werden können, wenn das Reset-Signal Eins ist. Außerhalb dieses Prozesses sollten Sie spezifizieren, wann das Enable Signal für den Counter Eins und wann Null ist. Dieses sollte abhängig davon sein, ob Sie Daten prüfen oder ein CRC generieren. Je nachdem müssen Sie entweder 12 Schritte oder 16 Schritte machen. Dieses können Sie wiederum vom Counter auslesen.

Grundlegend für beide Beschreibungen ist auch die theoretische Grundlage. Sie benötigen ein Schieberegister. Im ersten Schritt werden die auf den Schaltern eingestellten Daten in das Schieberegister geladen. Danach wird dieses 12 bzw. 16 mal geshiftet. Vor dem Shiften ist zu prüfen, ob im am weitesten links stehende Bit eine Eins steht. Ist dieses der Fall müssen die ersten fünf Bit des Schieberegisters durch XOR mit dem Generatorpolynom 10011 berechnet werden. Tip: da das erste Bit sofort rausgeschoben wird, ist ein XOR überflüssig. Ein XOR mit 0 hat keinen Einfluss. Daher ist es ausreichend, das vierte und fünfte Bit (Zählung von links nach rechts) zu invertieren.

Bei der Verhaltensbeschreibung können Sie nun innerhalb eines Prozesses das Laden ihres Schieberegisters als auch die Schiebeaktionen abhängig von dem Clocksignal durchführen. Das Schieberegister können Sie als Standard-Logik-Vektor realisieren. Bei der Strukturbeschreibung sollten Sie zuerst die Architektur einer Speicherzelle (als Latch/Flip-Flop) beschreiben, welche die Möglichkeit hat, ein Signal mit der Breite 1 zu Speichern. Weiterhin sollte es geladen und resettet werden können. Das Laden kann entweder durch entsprechende Signale (Eingänge) der Speicherzelle realisiert werden oder durch einen Multiplexer. Wie jedes flankengetriggertes, getaktetes Latch sollte die Speicherung, Ausgabe, Laden etc. von dem Clocksignal abhängig sein. Mit Hilfe dieses Latches können Sie nun Ihr Schieberegister realisieren, indem Sie Instanzen der Komponente Aufrufen und die Signale der einzelnen Komponenten miteinander verbinden so dass ein Schieberegister entsteht.

1.4: Hilfreiche Befehle

- Bedingungen

```
1  ARCHITECTURE Beispiel1 OF Bedingung IS
2  SIGNAL a: STD_LOGIC_VECTOR (1 DOWNTO 0);
3  SIGNAL b: STD_LOGIC_VECTOR (2 DOWNTO 0);
4  BEGIN
5       PROCESS(a,b)
6       BEGIN
7            IF (a="01") THEN
```

```
 8              b<="001";
 9          ELSIF (a="01") THEN
10              b<="000";
11          ELSE
12              b<="111"';
13          END IF;
14      END PROCESS;
15  END Beispiel1;
```

- Fallunterscheidung

```
 1  ARCHITECTURE Beispiel2 OF Fallunterscheidung IS
 2  SIGNAL a,b: STD_LOGIC_VECTOR (2 DOWNTO 0);
 3  BEGIN
 4      PROCESS(a,b)
 5      BEGIN
 6          CASE a IS
 7              WHEN "001" => b <= "000";
 8              WHEN "011" | "110" => b <= "100";
 9              WHEN OTHERS => b <= "111";
10          END CASE;
11      END PROCESS;
12  END Beispiel12;
```

- Bedingung für Signale außerhalb eines Prozesses

```
1  ARCHITECTURE Beispiel3 OF Signalbedingung IS
2  SIGNAL a,b,c,d: STD_LOGIC;
3  BEGIN
4      a<= NOT a WHEN b = '1' ELSE (c XOR d);
5  END Beispiel13;
```

- Clock-Signale

```
 1  ARCHITECTURE Beispiel4 OF Clocksignal IS
 2  SIGNAL clk: STD_LOGIC;
 3  SIGNAL count: STD_LOGIC_VECTOR(4 DOWNTO 0);
 4
 5  BEGIN
 6      clk <= clock_50;
 7      PROCESS(clk)
 8          IF (clk='1' AND clk'event) THEN -- wenn clk high und fallende Flanke
 9              count <= count + 1;
10          ELSE
11              count <= count;
12          END IF;
13      END PROCESS;
14  END Beispiel14;
```

- Variablen und Schleifen

```
 1  ARCHITECTURE Beispiel5 OF Schleife IS
 2  SIGNAL clk,reset: STD_LOGIC;
 3  SIGNAL register: STD_LOGIC_VECTOR(6 DOWNTO 0);
 4
 5  BEGIN
 6      clk <= clock_50;
 7      reset <= NOT KEY(0);
 8
 9      PROCESS(clk)
10          VARIABLE i : INTEGER := 0;
11          IF reset ='1' THEN
12              FOR i IN 0 TO 5 LOOP
13                  IF i = 0 OR i = 1 THEN
14                      register(i) <= NOT register(i+1);
15                  ELSE
16                      register(i) <= register(i+1);
17                  END IF;
18              END LOOP;
19          END IF;
20      END PROCESS;
21  END Beispiel15;
```

Teil 2: Aufgaben

Aufgabe 1: CRC in VHDL als Verhaltensbeschreibung

Implementieren Sie unter VHDL einen CRC Prüfsummengenerator und Prüfer, wie im vorherigen Übungsplatt beschrieben als Verhaltensbeschreibung (Behaviour). Als Orientierung können Sie sich die im Theorieteil dieses Blattes beschriebene Vorgehensweise zu nutze machen. Auf Blatt 5 finden Sie die benötigten Grundlagen für die VHDL Implementierung. Auf Blatt 4 finden Sie weitere Informationen zum CRC.

Aufgabe 2: CRC in VHDL als Strukturbeschreibung

Beschreiben Sie nun unter VHDL einen CRC Prüfsummengenerator und Prüfer als Strukturbeschreibung. Implementieren Sie hierzu Ihre eigene Speicherzelle, welche als Komponente für Ihr Schieberegister verwendet werden soll.