Praktische Übungen zu Computertechnik 2

Versuchsprotokoll

Versuch:

A2 – Befehlszähler eines RISC-Prozessors

Versuchsdatum und -zeit:

Donnerstag, 06. Mai 2010, 10-13 Uhr

Betreuer:

Volker Dörsing

Name, Studiengang, Mat.-Nr.:

Ralf Wondratschek, B.Sc. Informatik, 112626

Email:

ralf.wondratschek@uni-jena.de

Name, Studiengang, Mat.-Nr.:

Kerstin Gößner, B. Sc. Informatik, 114656

Email:

kerstin.goessner@uni-jena.de

Vom Betreuer auszufüllen:

Vorbereitung/Kolloquium:

Durchführung:

Protokoll:

Gesamtbewertung:

Gliederung

1. Vorbereitung

Für diesen Versuch ist ein DLXJ-Prozessor (eine RISC-Prozessorarchitektur von Hennessy und Patterson abgewandelt in der Jenaer Version von Dr. Reinsch) gegeben. In diesem Projekt speziell werden nur wenige Funktionseinheiten aus dem Datenpfad des Prozessorkerns benutzt.

Ziel ist es den Befehlszähler genauer zu untersuchen. Die Signalfolgen an den Funktionseinheiten mit den entsprechenden Funktionen sollen erprobt und verstanden werden.

Um dies zu erreichen stehen die bekannten Werkzeuge der CAE-Entwicklung zur Verfügung. Zuerst erfolgt die funktionale Simulation, in der der vorgegebene VHDL-Entwurf analysiert wird. Danach muss dieser Entwurf synthetisiert und auf einem FPGA (Field Programmable Gate Array) implementiert werden. Bei der anschließenden zeitbehafteten Simulation sollen die Unterschiede zwischen theoretischem und realem Verhalten erkannt werden. Die letzte Aufgabe besteht in der Erprobung. Mit Hilfe von Testprogrammen sollen Signale an den FPGA gelegt werden und die Ergebnisse an einer Digitalanzeige und 8 LEDs abgelesen werden.

2. Vorgehensweise

2.1 Funktionale Simulation

Für den Versuch sind einige VHDL-Dateien zur Beschreibung des Datenpfades und seiner Komponenten vorgegeben, so zum Beispiel datapath.vhd, alu_core.vhd und viele mehr (siehe Versuchsvorbereitung). Bei der funktionalen Simulation werden mit Hilfe des Skriptes fsim.sh all diese VHDL-Dateien analysiert, ein Simulationsmodell wird erstellt, welches auch ausgeführt wird, und zum Schluss öffnet sich ein Visualisierungsprogramm mit den graphisch dargestellten Signalverläufen (siehe Seite A1). Die standartmäßige Testbench tb.vhd wird natürlich noch um weitere Funktionen erweitert um die Funktionen des Befehlszählers besser untersuchen zu können.

2.2 Synthese und Implementierung

Bei diesem Schritt steht erneut das Skript imp.sh zur Verfügung. Es analysiert wie in der funktionalen Simulation zunächst die VHDL-Dateien und synthetisiert sie anschließend, das heißt die abstrakte VHDL-Beschreibung wird in eine logische Funktion überführt. Es entsteht eine Netzliste, die beim nächsten Punkt, der Implementierung, gebraucht wird.

Bei der Implementierung wird die Netzliste auf die Komponenten des FPGA abgebildet. Es werden Logikblöcke bestimmt, welche zur Verwendung nötig sind, und die nötigen Verbindungen zwischen ihnen werden hergestellt. Es entsteht eine Konfigurationsdatei fpga.bit, die auf der Speicherebene des FPGA gespeichert wird. Da die Länge der Verbindungen bekannt ist, können die Verzögerungszeiten berechnet werden. Diese werden in der Datei fpga.sdf gespeichert. Als drittes entsteht noch die Datei fpga.vhd.

Zum Schluss öffnet sich der FPGA-Editor mit einer graphischen Übersicht aller Komponenten

2.3 Zeitbehaftete Simulation

Auch hier steht im Vergleich zum Versuch A1 wieder das Skript tsim.sh zur Verfügung. Es analysiert die fpga.vhd Datei und bezieht dabei die Zeitinformationen aus der fpga.sdf Datei mit ein.

2.4 Erprobung der Experimentalschaltung

Am lokalen Rechner wird das FPGA-Experimentalsystem mit dem Programm konf und der Datei fpga.bit konfiguriert und anschließend können verschiedene Signalfolgen mit dem Programm prob angelegt werden. Die Ergebnisse bzw. die Ausgabe kann an den 8 LEDs und der Digitalanzeige abgelesen werden.

3. Erprobung

Da die Testbench tb.vhd bereits vorgegeben war (siehe Seite A6), konnten wir diese und die anderen Dateien für die Beschreibung des Datenpfades direkt mit dem Skript fsim.sh analysieren. Im Visualisierungsprogramm (siehe Seite A1) konnte man die Bedeutung der einzelnen Signale gut ablesen:

- Const_sel → liefert Konstante über den S1-Bus an das Eingangslatch L1
- /const_o1_en → bei logischer Null wird diese Konstante über den S1-Bus gelegt
- /const_02_en → bei logischer Null wird die Konstante 0x0 auf den S2-Bus gelegt
- PC_latch_en → bei logischer Eins wird das Signal PC_addr_out auf den Wert des PC geändert
- /PC_out_en → bei logischer Null wird der Wert des PC auf den S2_Bus gelegt
- PC_addr_out → Ausgabe des zuletzt freigegebenen (mit

PC_latch_en) Wertes vom PC, ständig aktiv

- PHI1 → entspricht Takt am L1 und L2 Latch; bei logischer Eins wird das Signal vom S1_Bus und S2_Bus an die ALU gelegt und additiv verknüpft
- PHI2 → entspricht Takt am PC; bei logischer Eins wird das Signal vom Dest_Bus in den PC geladen und gespeichert

Liegt bei /const_o1_en eine logische Null an, so gelangt der Wert von const_sel (vom Konstanten-Speicher) auf den S1_Bus. Liegt bei /const_o2_en eine logische Null an und bei /PC_out_en eine logische Eins, so gelangt die Konstante 0x0 vom Konstanten-Speicher auf den S2_Bus. Liegt ein Takt an den beiden Latches L1 und L2 an, sprich liegt an PHI1 eine logische Eins an, so gelangen die Werte in die ALU und werden zu einem gültigen Wert additiv verknüpft. Über den Dest_Bus wandert das Signal an den PC. Liegt bei PHI2 eine logische Eins an, so wird das Signal gespeichert (der PC ist gültig). Liegt bei PC_latch_en eine logische Eins an, so ändert sich der Wert vom Signal PC_addr_out auf den Inhalt des PC. Liegt bei /PC_out_en eine logische Null an, so wird der Wert vom PC auf den S2_Bus gelegt. Dabei ist zu beachten, dass bei /PC_out_en und /const_o2_en niemals gleichzeitig eine logische Null anliegt, da es sonst zu Konflikten kommt. Dieser Sachverhalt wird später bei der Erprobung der Experimentalschaltung allerdings noch genau untersucht.

Anschließend haben wir die Testbench erweitert (siehe Seite A7) und 4 Funktionen des PC simuliert (Siehe Seite A2).

	DEST_BUS	**PHI2**	**PC_LATCH_EN**	**Wert in PC**	**PC_ADDR_OUT**
Zeile 1	new_Data	0	0	old_Data	old_Data
Zeile 2	new_Data	1	0	new_Data	old_Data
Zeile 3	new_Data	0	1	old_Data	old_Data
Zeile 4	new_Data	0	1	new_Data	new_Data
Zeile 5	new_Data	1	1	new_Data	new_Data
Zeile 6					

PHI2 und PC_LATCH_EN haben jeweils zwei verschiedene logische Werte, deshalb gibt es $2^2 = 4$ Möglichkeiten. Man muss allerdings an einer Stelle beachten (Zeile 4 und 5), welcher Wert im PC gespeichert ist, für die korrekte Ausgabe (deshalb befindet sich der Wert vom PC auch in der Tabelle).

Nach der funktionalen Simulation führten wir die Synthese und Implementierung mit Hilfe des Skriptes imp.sh aus. Aus dem FPGA-Editor konnte man die einzelnen verwendeten Komponenten des Schaltkreises ablesen:

- BUFGMUX → 2x
- IBUF → 8x
- SLICEL → 9x

- IOB → 16x

Anschließend begannen wir mit der Zeitsimulation der erweiterten Testbench aus der zeitlichen Simulation mit Unterstützung des Skriptes tsim.sh. Es fiel deutlich auf, dass die drei Busse S1_Bus, S2_Bus und Dest_Bus fehlten (siehe Seite A3). Das liegt unter anderem daran, dass die Werte der inneren Schaltung, wozu die Busse gehören, nicht nach außen sichtbar sind. Für die zeitliche Simulation und spätere Erprobung ist es daher extrem wichtig zu verstehen, was jedes Signal und jede Signaländerung bewirkt. Liegt zum Beispiel bei PC_latch_en eine logische Null an, so muss man sich auch merken, welcher Wert tatsächlich im PC steht, wenn ein neuer Wert von der ALU kommt. Bei dieser muss man auch mitrechnen, da sonst möglicherweise auch unverhoffte Ergebnisse im PC stehen.

Im Vergleich zur funktionalen unterscheidet sich die zeitliche Simulation in einigen Punkten (Vergleich Seite A2 und A3). Am deutlichsten fällt auf, dass in den ersten 100ns nichts außer der Null bei der Zeitsimulation auf PC_addr_out ausgegeben wird. Das liegt an der Initialisierungszeit. Folglich entstehen auch später andere Ergebnisse im Vergleich zur funktionalen Simulation, wenn der Wert im PC mit dem Wert const_sel additiv verknüpft wird. Um dem entgegen zu wirken, haben wir die Signale const_sel, /const_01_en, /const_o2_en, /PC_out_en und PC_latch_en um 120ns nach hinten verschoben (siehe Seite A4). Wir konnten nicht 100ns wählen, weil die Signale sonst nicht mehr mit den Takten übereingestimmt hätten. Ein Takt (z. B. PHI1) wiederholt sich alle 40ns und da die Initialisierungszeit 100ns beträgt, liegt 120ns am nächsten. Des Weiteren sticht ein weiterer Unterschied in der Zeitsimulation ins Auge: Obwohl bei 140ns PHI2 und PC_latch_en auf logisch Eins gesetzt werden, ändert sich PC_addr_out erst nach rund weiteren 10ns (siehe Seite A3), was auf die Signallaufzeiten zurückzuführen ist.

Um neue Funktionen des Befehlszählers zu erhalten, haben wir die Testbench erneut geändert (siehe Seite A8). Zum einen haben wir wie oben besprochen die Signale um 120ns nach hinten verschoben und zum anderen haben wir die Eingaben an die gewünschten Funktionen angepasst. Auf Seite A4 sieht man das dokumentierte Ergebnis.

P_1, P_3 und P_5 unterscheiden sich nur bei dem Signal const_sel und der Ausgabe PC_addr_out. PC_latch_en spielt bei diesen Punkten keine Rolle, da es hier um die beiden Latches L1 und L2 und um die ALU geht. /PC_out_en und /const_o1_en sind beide auf logisch Null gesetzt, weil zum einen die Konstante von const_sel am S1_Bus und zum anderen der Inhalt vom PC an S2 liegt. Der Takt PHI1 ist auf logisch Eins gesetzt und somit werden die beiden Werte über die Latches an die ALU weitergeleitet und additiv verknüpft. /const_02_en liegt auf logisch Eins, damit es keine Konflikte am S2_Bus gibt. Die Werte für const_sel wurden nach der Aufgabenstellung

gewählt, sprich der PC wird zuerst um 4, dann um 1 und anschließend um 0 erhöht.

P_7 unterscheidet sich von obig erklärten Punkten, weil es hier gefordert war, den PC auf 0 zu setzen. Deshalb wurde /const_02_en auf logisch Null und /PC_out_en auf logisch 1 gesetzt. Daraus folgt dass nicht der Inhalt des PC mit const_sel verknüpft wird, sondern der Wert auf dem S2_Bus (in dem Fall die Konstante 0x0). Da bei const_sel eine 0 anliegt, wird 0+0 berechnet und der PC wird nach weiteren 20ns wieder auf 0 gesetzt.

P_2, P_4, P_6 und P_8 sind bis auf die Signale const_sel und PC_addr_out gleich. const_sel, /PC_out_en, /const_o1_en und /const_02_en spielen bei diesen Punkten keine Rolle, da die beiden Latches gesperrt sind (PHI1 auf logisch Null) und somit der vor rund 20ns berechnete Wert am PC anliegt. Wichtig ist zum einen PHI2, was dem Takt entspricht und zum anderen PC_latch_en. PHI2 liegt auf logisch 1, damit der Wert von der ALU im PC abgelegt wird. PC_latch_en bewirkt, dass automatisch der neue Wert vom PC an das Signal PC_addr_out weitergeleitet wird und somit auch ausgegeben wird. Wie in der Aufgabenstellung gefordert, wird zunächst 0 (Startwert), dann 4 (um 4 erhöhen), dann 5 (um 1 erhöhen), wieder 5 (um 0 erhöhen) und zuletzt 0 (auf 0 zurücksetzen) ausgegeben.

Wie bereits weiter oben erwähnt, haben sich die Änderungszeitpunkte von PC_addr_out um 10ns im Vergleich zur funktionalen Simulation nach hinten verschoben, was auf die Signallaufzeiten zurückzuführen ist. Im Einzelnen ergaben sich bei 3 verschiedenen PC-Werten (siehe Seite A4):

- 9,822ns (Zeitpunkt ca. 160ns)
- 9,846ns (Zeitpunkt ca. 219ns)
- 9,982ns (Zeitpunkt ca. 300ns)

Die Signallaufzeiten sind von vielen Faktoren abhängig, zum Beispiel der Temperatur, welche zu den 3 Zeitpunkten nicht gleich ist, deshalb ergeben sich auch kleine Unterschiede bei den Messungen. Auf Seite A5 findet man eine Messung im Überblick.

Die Zeitsimulation war damit abgeschlossen und wir widmeten uns der Erprobung der Experimentalschaltung. Wir konfigurierten das FPGA mit dem Programm und der Konfigurationsdatei fpga.bit am lokalen Computer. Anschließend legten wir mit dem Programm prob die folgenden Signalwerte an unseren Schaltkreis. Die Werte, die wir änderten, sind fett markiert.

/PC_out_en	PC_latch_en	/const_o2_en	/const_o1_en	const_sel	PHI1	PHI2	PC_addr_out
0	0	0	**1**	0	0	1	0
0	**1**	0	1	0	0	1	0
0	1	**1**	1	0	0	1	0
0	1	1	**0**	0	0	1	0
0	1	1	0	**2**	0	1	0
0	1	1	0	2	**1**	**0**	0
0	1	1	0	2	**0**	**1**	**4**
0	1	1	0	**1**	0	1	4
0	1	1	0	1	**1**	**0**	4
0	1	1	0	1	**0**	**1**	**5**
0	1	1	0	**0**	0	1	5
0	1	1	0	0	**1**	**0**	5
0	1	1	0	0	**0**	**1**	5
1	1	1	0	0	0	1	5
1	1	**0**	0	0	0	1	5
1	1	0	0	0	**1**	**0**	5
1	1	0	0	0	**0**	**1**	**0**

Diese Werte entsprechen den 4 Funktionen, die wir in der Zeitsimulation bereits diskutierten. Die Bedeutung der Werte wurden bereits erklärt und es traten auch keine Unterschiede auf, alles verlief wie vermutet.

Es gab allerdings noch 3 weitere Schaltungen, wo wir nicht wussten, was passieren wird, weshalb wir sie auch erprobten. Bei der ersten wollten wir erfahren, was passiert, wenn sowohl /const_o2_en als auch /PC_out_en auf logisch Null liegen. Dazu setzten wir die obige Tabelle fort:

/PC_out_en	PC_latch_en	/const_o2_en	/const_o1_en	const_sel	PHI1	PHI2	PC_addr_out
0	1	0	0	0	0	1	0
0	1	0	0	0	**1**	**0**	0
0	1	0	0	0	**0**	**1**	0

Es passierte nichts Außergewöhnliches. Wir vermuteten eine Fehlermeldung, doch wir konnten ohne Probleme mit dem FPGA weiter arbeiten. Bei der 2. Schaltung wollten wir erfahren, was bei einem vorhandenen Wert im PC und einen anliegenden Wert am S1_Bus passiert. Wir setzten wieder die obige Tabelle fort:

/PC_out_en	PC_latch_en	/const_o2_en	/const_o1_en	const_sel	PHI1	PHI2	PC_addr_out
1	1	0	0	0	0	1	0
1	1	0	0	**2**	0	1	0
1	1	0	0	2	**1**	**0**	0
1	1	0	0	2	**0**	**1**	**4**
0	1	0	0	2	0	1	4
0	1	0	0	2	**1**	**0**	4
0	1	0	0	2	**0**	**1**	**8**

Wie man erkennt, legten wir den Wert 4 in den PC. Anschließend setzten wir wieder /PC_out_en und /const_o2_en gleichzeitig auf logisch Null. Danach erfolgte der Takt zum berechnen (PHI1 auf logisch Eins) und der Takt um den PC gültig zu machen. Zu unserer Verwunderung kam es wieder zu keinem Fehler, sondern const_sel und der Wert aus dem PC wurden wieder addiert

ohne einen Konflikt. In der 3. interessanten Schaltung wollten wir wissen, was passiert, wenn /PC_out_en, /const_o1_en und /const_o2_en auf logisch Eins liegen und der Takt zum Berechnen erfolgt (PHI1 auf logisch Eins). Dazu setzten wir wieder die obige Tabelle fort:

/PC_out_en	PC_latch_en	/const_o2_en	/const_o1_en	const_sel	PHI1	PHI2	PC_addr_out
0	1	0	**1**	2	0	1	8
0	1	**1**	1	2	0	1	8
1	1	1	1	2	0	1	8
1	1	1	1	2	**1**	**0**	8
1	1	1	1	2	**0**	**1**	**0**

Man erkennt deutlich in den letzten beiden Zeilen, dass ein Reset erfolgte und im PC eine 0 steht.

4. Schlussfolgerungen

Bei genauer Studie der Vorbereitungsmaterialien konnte man sich bereits ein gutes Bild des Befehlszählers des DLXJ-Prozessors machen. Der prinzipielle Aufbau war verstanden und die Wirkung der Signale verinnerlicht. Deshalb kam es bei der funktionellen Simulation auch zu keinen Überraschungen.

Bei der Zeitsimulation erwarteten wir wieder Verzögerungszeiten bei den Signalen, was mit dem Ausgabesignal PC_addr_out bestätigt wurde. Die Zeiten waren mit rund 10ns auch nicht überraschend groß oder klein. Es war wieder kein großes Problem bestimmte Werte in den PC zu laden mit den richtigen Signalen, da diese verinnerlicht waren. Verwunderlich war anfangs die Tatsache, dass die Busse im Visualisierungsprogramm fehlten, allerdings wurde es in der Nachbearbeitung klar.

Da die Zeitsimulation problemlos funktionierte, waren wir guter Dinge, dass die Erprobung der Experimentalschaltung ohne große Komplikationen verläuft. Dies war dann tatsächlich auch der Fall. Mit denselben Werten aus der zeitlichen Simulation erhielten wir in der Erprobung auch die gleichen Ergebnisse. Dagegen waren wir bei dem verbotenen Zustand umso erstaunter. Wir erwarteten einen Konflikt bei /const_o2_en und /PC_out_en gleichzeitig auf logisch Null. Dies trat nicht ein. Im Gegenteil: der Wert aus dem PC wurde problemlos mit dem Wert vom S1_Bus addiert. Unsere Vermutung zum Reset, wenn sowohl /PC_out_en, /const_o1_en als auch /const_o2_en auf logisch Eins liegen wurden bestätigt.

Zusammenfassend haben wir den Aufbau und die Funktionsweise des Befehlszählers des DLXJ-Prozessors kennengelernt und verstanden. Der Umgang mit den CAE-Werkzeugen wurde tiefer verinnerlicht.