High-Level-Synthese einer ATM-Switch-Steuerung mit dem Behavioral Compiler™ von Synopsys™

Walter Lange Wolfgang Rosenstiel

WSI-99-1

20. Januar 1999

Wilhelm-Schickard-Institut
Universität Tübingen
D-72076 Tübingen, Germany
e-mail: wlange@informatik.uni-tuebingen.de

ISSN 0946-3852

High-Level-Synthese einer ATM-Switch-Steuerung mit dem Behavioral Compiler™ von Synopsys™

Walter Lange Wolfgang Rosenstiel

Januar 1999

Zusammenfassung

In der vorliegenden Arbeit werden mit Hilfe eines handelsüblichen High-Level-Synthese-Werkzeugs, dem Behavioral Compiler™ (BC) der Firma Synopsys™ Schaltungen für eine ATM-Switch-Steuerung synthetisiert. Zweck der Untersuchung ist es festzustellen, ob es möglich und sinnvoll ist, Schaltungen für schnelle Datenübertragung mit Hilfe eines HLS-Werkzeugs zu erstellen. Die Syntheseergebnisse werden in der Zusammenfassung tabellarisch dargestellt.

Inhaltsverzeichnis

1 Einleitung

High-Level-Synthese-Werkzeuge sind erst seit wenigen Jahren auf dem Markt. Nachdem 1995 der "Behavioral Compiler™" der Fa. Synopsys™ angeboten wurde, erschien 1998 "Monet™ " der Fa. Mentor Graphics. Die Aufnahme der HLS-Werkzeuge von den Hardware-Entwicklern geschieht nur zögerlich, man scheut die Einarbeitungszeit und kann auch nicht abschätzen, ob der Einsatz der Werkzeuge zu einem Erfolg führt. Es gibt Beispiele für gute Synthesen von Verhaltensbeschreibungen von Matritzenberechnungen (MPEG-Komprimierung) oder Differentialgleichungen. Untersuchungen über den effektiven Einsatz von General-Purpose-HLS-Werkzeugen für die Synthese von Steuerschaltungen für Hochgeschwindigkeitsnetze sind noch nicht bekannt.

In der vorliegenden Arbeit wird eine Steuerschaltung für einen ATM-Switch (die ATM-Switch-Steuerung ASS) mit einem handelsüblichen HLS-Werkzeugen synthetisiert, die Ergebnisse werden tabellarisch dargestellt. Die Entwicklung der ASS ist am Wilhelm Schickard-Institut, am Lehrstuhl für Technische Informatik durchgeführt und im Internen Bericht: "Modellierung einer ATM-Switch-Steuerung" [LaRo97] ausführlich beschrieben worden.

Die in diesem Bericht häufig verwendete Namen "Synopsys" und "Behavioral Compiler" (BC) sind Handelsnamen.

Die Arbeit gliedert sich wie folgt: Im nächsten Abschnitt wird auf die High-Level-Synthese eingegangen, danach wird kurz die Anwendung: die ATM-Switch-Steuerung beschrieben. Im Kapitel 2 wird das verwendeteSynthesewerkzeug, der Behavioral Compiler, vorgestellt. In den Kapiteln 4 bis 10 wird die Synthese der einzelnen ATM-Module beschrieben. In der Zusammenfassung sind die Syntheseergebnisse tabellarisch dargestellt und werden diskutiert. Im Anhang sind die VHDL-Quellcodes der Module, die Testtreiber und die Syntheseberichte aufgelistet.

1.1 High-Level-Synthese

1.1.1 Grundlagen

Die High-Level-Synthese erzeugt aus der Verhaltensbeschreibung einer Schaltung (z.B.: in der Hardware-Beschreibungssprache VHDL) eine RT- (Register-Transfer) Struktur. Die RT-Struktur besteht aus einem Datenfluß-Teil und einem Kontrollfluß-Teil, der als FSM (Finite State Machine) ausgeführt ist [DeMi94] [Gaj92]. Der Datenfluß enthält die Lese-Operationen der Eingangssignale, die Datenverarbeitung derselben und die Schreiboperationen der Ausgangssignale.

Der Kontrollfluß definiert die zeitliche Abfolge in diskreten Taktschritten (Scheduling) der Datenflußoperationen.

Die High-Level-Synthese (HLS) wird von den heutigen HLS-Werkzeugen im wesentlichen in drei Schritten durchgeführt:

1. Allokation: Hier werden die Komponenten für die Schaltung selektiert.

2. Scheduling: Die Kontrollschritte werden festgelegt.

3. Assignment (oder Binding): Die in der Allokation definierten Komponenten werden den Operationen in den einzelnen Kontrollschritten zugeordnet.

Ein großer Vorteil für den Schaltungsentwickler ist, daß bei Einsatz eines HLS-Werkzeugs nicht nur die Selektion der einzelnen Komponenten vom Werkzeug durchgeführt wird, sondern auch die Entwicklung des Controllers (Finite State Machine, FSM) automatisch erfolgt. Der Entwickler kann sich auf die effiziente Beschreibung des Datenflusses und auf die Optimierung der Schaltung konzentrieren.

1.1.2 Ablauf der Synthese

Die High-Level-Synthese wird in folgenden Schritten durchgeführt:

1. Modifiziere die simulierte VHDL-Beschreibung in folgender Hinsicht:
   (a) Festlegung des “Reset” und entsprechende Kodierung. In unserem Fall ist ein synchrones Zurücksetzen ausreichend.
   (b) Festlegung des Eingabe-Ausgabe-Modus (siehe Kapitel: “Eingabe-Ausgabe-Modi des BC”). In unserem Fall ist der “superstate I/O-Modus” angebracht.
   (c) Berücksichtigung der Regeln für zusätzliche Kontrollschritte im Eingabe-Ausgabe-Modus.
   (d) Festlegung für das Aufrollen von Schleifen. In unserem Fall ist es meist angebracht, die Schleifen nicht aufzurollen, d.h. da das Aufrollen meist als Vorgabe (Default) genommen wird, muß ein entprechendes Attribut gesetzt werden. (don’t_unroll loop “label”).
2. Simulation des Moduls mit Hilfe eines Testtreibers. Damit wird die Funktionalität überprüft.
3. Synthese des Moduls mit folgenden Parametern:
   (a) I/O-Modus: “superstate fixed” I/O.
   (b) Ansiedlung im Entwicklungsraum: (Design Space): Die Schaltung mit geringer Verzögerung hat Vorrang vor geringer Fläche.
   Beim BC wird in der “time_design”-Stufe, in der die Allokation durchgeführt wird, die Option “fastest” gewählt.
   (c) Festlegung der Taktperiode. In unserem Fall wird jeweils ein Taktperiode von 25 ns gewählt.
   (d) Scheduling-Aufwand: Beim BC wird -effort “medium” aus 4 Stufen: (0, low, medium, high) gewählt.

   Das Ergebnis ist eine Strukturbeschreibung auf RT-Ebene in VHDL.
4. Simulation des synthestisierten Moduls mit Hilfe des oben verwendeten Testtreibers. Damit ist eine Überprüfung der Funktionalität der synthetisierten Schaltung möglich.

Die Simulation nach der Synthese mit anschließender Überprüfung der Funktionalität ist eine notwendige, wenn auch nicht hinreichende Bedingung für die Korrektheit der synthetisierten Schaltung.

1.2 Die Anwendung

Als Anwendung wird eine in der Hardwarebeschreibungssprache VHDL modellierte ATM-Switch-Steuerung verwendet ([LaRo97]). Auf dem Gebiet der Informationsübertragung ist ATM (Asynchroner Transfer Modus) eine weitverbreitete Übertragungsmethode. Die Datenübertragung erfolgt in Datenpaketen konstanter Länge (53 Bytes). Der hohen Durchsatzrate wegen werden Hardware-Lösungen bei der Steuerung der ATM-Schaltungen benötigt. Das Blockschaltbild (Abbildung 1) zeigt einen Kanal einer ATM-Übertragungsleitung in der sog. ATM-Schicht. Die Funktionen der einzelnen Module (AHT_IN, HT/RT, RC, SRC, CC, RD und AHT_Out) sind in [LaRo97] definiert und die Synthese dieser Module wird in den folgenden Kapiteln beschrieben.

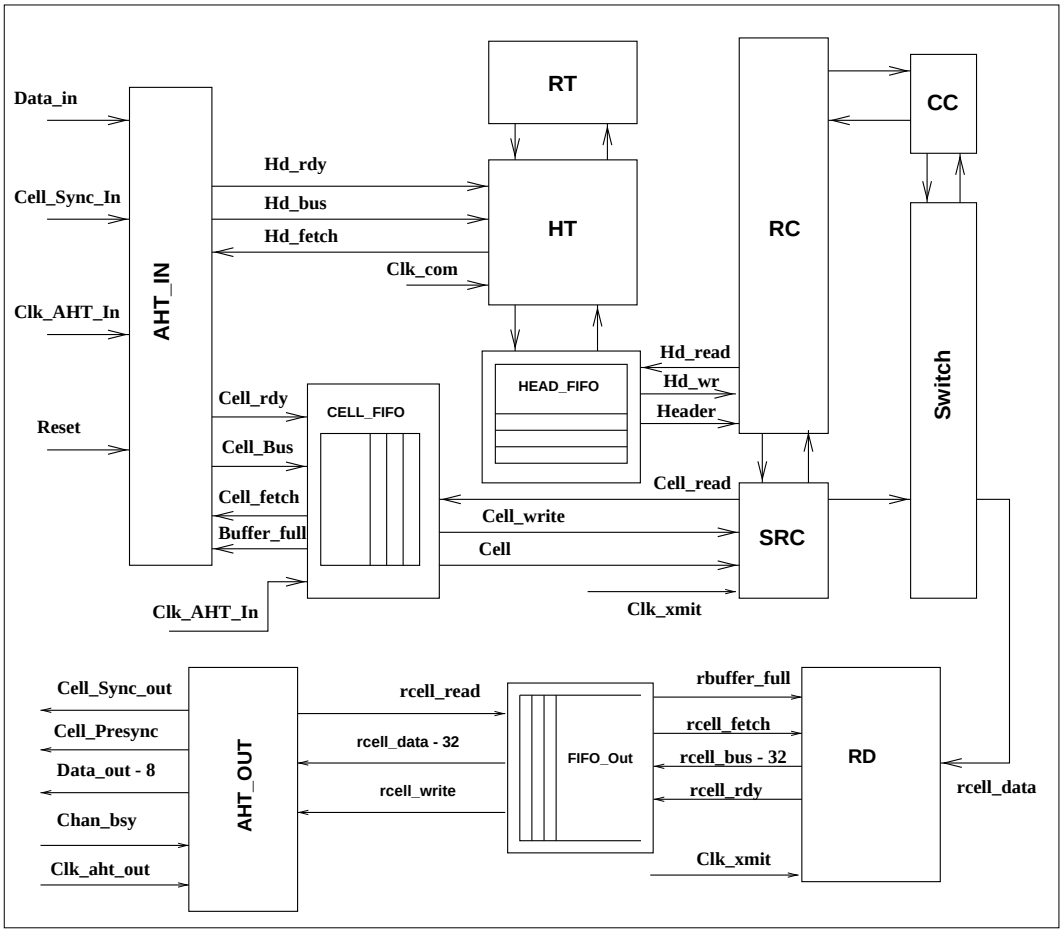


Abbildung 1: Blockschaltbild eines Kanals der ATM-Switch-Steuerung

1.3 Die HLS-Technologie-Bibliothek

Der Behavioral Compiler benötigt eine Technologie-Bibliothek (HLS Library) für die Abschätzung von Chipfläche (area) und Verzögerung (delay) und für die Erstellung der RTL-Netzliste als Ergebnis der High Level Synthese. In dieser Bibliothek ist eine Sammlung von meist generischen Komponenten basierend auf einer bestimmten Technologie (z.B. FPGA's, LCA's), die mit Attributen über Chipfläche und Verzögerung versehen sind, enthalten. Da die HLS-Bibliotheken nicht genormt sind, hat jedes Werzeug sein eigenes Bibliotheksformat, das anderen Werkzeugen nicht zugänglich ist.

Für den Behavioral Compiler werden folgende Bibliotheken eingesetzt:

- Die Komponenten werden aus den Synopsys"Design Ware"-Bibliotheken (DW01 bis DW07) genommen.
- Als Ziel-Technologie wird die Bibliothek "lsi_10k.lib" eingesetzt, die eine "Logic Cell Array" (LCA) Technologie repräsentiert. Die Flächeneinheit 1 wird durch den 1-Bit-Inverter dargestellt.

2 Der Behavioral Compiler (BC) von Synopsys

Bevor mit BC VHDL-Beschreibungen von Schaltungen für die High-Level-Synthese bearbeitet werden können, werden sie in verschiedenen Kompilierungsschritten auf Konformität mit den methodischen Richtlinien des BC [Syn97] überprüft.

2.1 Die Synthese-Schritte

Folgende Syntheseschritte, die jeweils im "bc_script" (siehe Anhang) festgelegt werden, müssen mit dem BC durchgeführt werden, um eine Schaltung auf RT-Ebene zu erhalten:

1. Erste Analyse der Schaltung: (analyze). Dies ist im wesentlichen eine Syntax-Prüfung des VHDL-Quellcodes, wobei vom Compiler nur VHDL-Sprachkonstrukte angenommen werden, die grundsätzlich synthetisierbar sind.
2. Prüfung auf Synthesefähigkeit des VHDL-Quellcodes (elaborate). In diesem Schritt wird ein gemischte "Steuerung-Datenfluß"- ("Control-Dataflow" oder FSMD) Repräsentation der Schaltung erstellt.
3. Definition der Taktperiode und der Zurücksetzungs- bzw. Reset-Methode. Die Taktperiode wird auf 25 ns festgelegt. Als Reset-Methode wird globales synchrones Reset gewählt.
4. Festlegung, ob Ein-oder Ausgänge der FSM mit Registern versehen werden. In unserem Fall wird beim Modul AHT_In der Eingang der FSM mit Registern versehen, um Synchronisation des Dateneingangs mit dem Synchronisationssignal zu erreichen.

5. Zeit-Untersuchung (time_design) auf der Basis einer vorher festgelegten Zieltechnologie. In diesem Schritt werden die Komponenten allokiert. Als Vorgabe (default) werden die Komponenten ausgewählt die bei kleinster Fläche innerhalb der definierten Taktperiode die entsprechende Operation ausführen können. Wählt man die Option “fastest”, so werden die Komponenten mit der geringsten Verzögerung selektiert.

   Der Datenfluß-Teil der Schaltung wird in eine Netzliste aus Basiskomponenten umgeformt und in die gewählte Zieltechnologie transformiert. Danach wird eine Zeitanalyse durchgeführt.

6. Scheduling/Assignment: Festlegung der Kontrollschritte und Zuteilung der Komponenten zu den Operationen.

7. Ergebnis-Ausdrucke (reports).

8. RTL-Schaltungs-Ausgabe in VHDL, entweder mit Komponenten oder mit flacher Hierarchie (simulierbar).

2.2 Regeln für VHDL-Beschreibungen für die Synthese mit BC

2.2.1 Synchronisation der Datenannahme

“Synchronisieren der Datenannahme” bedeutet, Daten von einer Datenleitung (hier z.B.: “Data_In”) dann in ein Register zu übernehmen, wenn ein Synchronisationssignal (hier z.B.: das Zellstartsignal “Cell_Sync_In”) zusammen mit der positiven Flanke des Taktes aktiv wird.

Beim BC wird dies wie folgt erreicht:

1. Durch eine Registrierung der Eingänge oder der Ausgänge der FSM des entsprechenden Prozesses: Meist hat die FSM weniger Eingänge als Ausgänge, daher ist es günstiger, die Eingänge zu speichern. (Geschieht durch ein Befehl im “Bc_script”).

2. Durch folgende “Start-Loop” für die zu synchronisierenden Signale: In unserem Beispiel sind es die Signale: Cell_Sync_In und Data_In:

```
start_loop: loop     -- fastest handshaking
  Hd_reg(31 downto 24) := Data_In;
  IF  (Cell_Sync_In = '1') THEN
    Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
    exit start_loop;
  else
    Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
  END IF;
End loop;
```

2.2.2 Die Eingabe-Ausgabe-Modi des BC

Als Scheduling-Option kann ein aus drei Eingabe-Ausgabe-Modi festgelegt werden:

1. “Cycle fixed” I/O-Modus. Hier werden die einzelnen Kontrollschritte vom Entwickler festgelegt und zwar dadurch, daß “WAIT UNTIL Clock .. ”-Anweisungen an den Stellen in den VHDL-Quellcode eingefügt werden, an denen Kontrollschritte nötig sind. Diese Methode ist für eine High-Level-Synthese nur dann angebracht, wenn dem “Scheduler” keine Freiheitsgrade bezüglich der Ein- oder Ausgabe gelassen werden dürfen. Für unsere Schaltungen ist dies nicht der Fall.

2. “Superstate fixed” I/O-Modus: “Superstates” sind Zustände zwischen zwei “Wait”-Anweisungen, in die der Scheduler noch weitere Kontrollschritte, wenn nötig, einfügt. Dieser Modus ist für unsere Anwendung insofern wichtig, da zwischen den einzelnen Modulen relativ viel Kommunikation stattfindet und die Zeitpunkte für Dateneingabe und Datenausgabe korrekt definiert sein müssen.

3. “Free floating” I/O-Modus: Hier ist es dem Scheduler vollkommen freigestellt, wie die Kontrollschritte gesetzt werden. Dieser Modus kann für unsere Anwendung nicht gewählt werden.

2.2.3 Regeln für “superstate fixed” I/O

Für die VHDL Beschreibung, die der BC im “superstate fixed” I/O-Modus synthetisieren soll, gibt es einige Regeln, die der Entwickler beachten muß. Falls der BC im Scheduling-Schritt auf einen Verstoß gegen eine der folgenden Regeln trifft, wird die Kompilierung mit einer entsprechenden Fehlermeldung abgebrochen. Meist sind es fehlenden Kontrollschritte, also “Waits”, die eingefügt werden müssen.

Die zusätzlichen Kontrollschritte werden nach folgenden Regeln für den I/O-Superstate-fixed Modus vom Compiler gefordert: (siehe [Syn97])

1. In einer Schleife darf als erste Anweisung dann keine I/O-Anweisung stehen, wenn im letzten “Superstate” der Schleife eine Schreibanweisung spezifiziert wird; denn die letzte und die erste Anweisung in einer Schleife gehören demselben Kontrollschritt an. Der Compiler meldet dies mit dem Fehler HLS-43.

2. Falls einer I/O-Anweisung eine Schleife folgt, die eine I/O Anweisung in ihrem ersten Superstate hat, so muß ein Kontrollschritt zwischen Schleife und I/O eingeführt werden. (HLS-44)

3. Einer While-Schleife, die von einer I/O-Operation gefolgt ist, darf kein Schreiben zu einem Port vorangehen. Auch hier muß ein Kontrollschritt in Form einer WAIT-Anweisung eingeführt werden. (HLS-45)

4. Eine Schleife, die ein Schreiben zu einem Port (write) im gleichen Superstate hat, in dem ein Exit aus der Schleife festgelegt ist, darf von keiner I/O-Operation gefolgt sein. Hier muß ein Kontrollschritt eingeführt werden. (HLS-46)

5. Bedingte Warte-Anweisungen können kein Schreiben zu einem Port und eine nachfolgende I/O-Operation trennen. Hier muß ein Kontrollschritt eingeführt werden. (HLS-47)

6. Wird in einer bedingten Anweisung (IF ..THEN) ein Kontrollschritt generiert, (WAIT-Anweisung) so muß im ELSE Zweig ebenfalls ein Kontrollschritt erzeugt werden. (HLS-47)

Zusammenfassend kann man sagen,

- I/O-Operationen werden wie folgt eingeplant:
  - Lese-Operationen (Read) können in jedem Kontrollschritt eines Superstates angelegt werden.
  - Schreib-Operationen (Write) können nur im letzten Kontrollschritt eines Superstates durchgeführt werden.
- Zwischen wiederholtem (einfachen oder mehrfachen) Schreiben oder (einfachen oder mehrfachen) Lesen eines Signals oder Ports muß eine Wait-Anweisung stehen. Dabei zählt ein kombiniertes WAIT auf den Takt und auf ein Signal z.B.:

  "WAIT UNTIL Clk_com'EVENT and Clk_com = '1'and Hd_rdy = '1';"

  als Schleife, nicht als Kontrollschritt.
- Stehen I/O's vor und hinter einer Schleife, dann muß eine WAIT-Anweisung dazwischengesetzt werden.
- Aufgerollte Schleifen dürfen keine geschachtelten Schleifen enthalten.

2.2.4 Zurücksetzen und Initialisieren der Schaltung (Reset)

Im BC-Benutzer Handbuch wird die Möglichkeit des expliziten oder impliziten synchronen Zurücksetzen beschrieben.

1. Explizites synchrones Zurücksetzen.

   Beim expliziten synchronen Zurücksetzen muß eine gesonderte "Reset_ Loop" beschrieben werden und nach jeder "Wait"-Anweisung folgende Exit-Anweisung eingesetzt werden: "exit Reset_loop when reset = '1';"

   Falls das "Reset"-Signal aktiv wird, wird es nach einem Kontrollschritt (d.h. synchron mit dem Takt) erkannt und die Schaltung verläßt die "Reset-Loop", d.h. die funktionale Ausführung beginnt von vorn im Zustand 1 der FSM.

   Diese Methode ist anzuwenden, wenn das Zurücksetzen von funktionaler Bedeutung ist und in der Simulation eingehend überprüft werden muß. Dies ist in unserer Anwendung nicht der Fall.

2. Implizites synchrones Zurücksetzen.

   Um die implizite Reset-Methode anzuwenden, wird in der VHDL-Beschreibung ein Reset-Signal (design_reset) eingeführt und im Synthese-Verlauf, (d.h. im Synthese-Script nach dem "elaborate"-Schritt) folgender zusätzlicher Befehl eingesetzt:

   "set_behavioral_reset design_reset -active high"

Die implizite synchrone Reset-Methode liefert dasselbe Ergebnis wie die explizite Methode, jedoch ist das Zurücksetzen nach der Synthese auf RTL-Ebene nicht simulierbar. Für unsere Anwendung wurde die implizite Reset-Methode gewählt.

3. Das Zurücksetzen nach einer der beiden obigen Methoden ist wichtig, da sonst eine Simulierung der Schaltung nach der HLS nicht möglich ist.

Die Initialisierung der Signale und Variablen, die meist in Registern abgelegt werden, wird nach Eintritt in den jeweiligen Prozeß durchgeführt (äußere Schleife im Prozeß). Die eigentlichen Funktionen des Prozesses werden in einer inneren Schleife ausgeführt, die eine “unendliche” Schleife sein kann (loop).

2.3 Verwendete BC-und Simulator-Version

Für die High-Level-Synthese mit dem Synopsys Behavioral Compiler wird die Version 1997.08 verwendet.

Für die Simulation und für die Fehlersuche wird der Synopsys-VSS-Simulator und Debugger (vhdldbx) derselben Version eingesetzt.

3 Zuverlässigkeit der Synthese

Um die Zuverlässigkeit des Synthesewerkzeugs beurteilen zu können, wir in diesem Zusammenhang ein Zuverlässigkeitsmaß Z wie folgt definiert:

1. **Z = 10:** Die Synthese wurde **fehlerlos** abgeschlossen. Das Ergebnis der Synthese (die Beschreibung der Schaltung auf RT-Ebene) ist nach Abschätzung des Entwicklers **akzeptabel** und **korrekt simulierbar,** d.h. die Simulation ist funktional gleich wie die Simulation der Verhaltensbeschreibung vor der Synthese. Das Ergebnis der Synthese kann dem nächsten Syntheseschritt (der RT-Synthese) zugeführt werden.

2. **Z = 8:** Die Synthese bricht mit einem **Programmfehler** ab. Durch Ändern der Verhaltensbeschreibung wird eine Umgehung des Fehlers erreicht. Dadurch wird das Ergebnis der Synthese **akzeptabel** und ist **korrekt** simulierbar. Das Ergebnis der Synthese kann dem nächsten Syntheseschritt zugeführt werden.

3. **Z = 5:** Die Synthese bricht mit einem Programmfehler ab. Durch Ändern der Verhaltensbeschreibung wird eine Umgehung des Fehlers erreicht. Dadurch wird das Ergebnis der Synthese nach Abschätzung des Entwicklers **schlechter** als bei Z = 10 oder Z = 8, jedoch das Ergebnis ist **korrekt** simulierbar. Das Ergebnis der Synthese kann dem nächsten Syntheseschritt zugeführt werden.

4. **Z = 2:** Die Synthese bricht mit einem Programmfehler ab. Es wird nur ein Teilergebnis erreicht, das zwar Aufschlüsse über die zu erwartende Schaltung ergibt, aber das Ergebnis der Synthese kann nicht weiter synthetisiert werden.

5. **Z = 0:** Die Synthese bricht mit einem Programmfehler ab. Es gibt keine brauchbaren Ergebnisse.

4 Das AHT-Eingangsmodul AHT_IN

Das AHT-Eingangsmodule AHT_IN nimmt Zellen aus der Physikalischen Schicht über die 8-Bit breite Datenleitung Data_in auf (siehe Bild 1 oder [LaRo97]).

Das erste Datenbyte kommt synchron mit dem Signal Cell_Sync_In und die restlichen Bytes der ATM-Zelle erscheinen synchron mit der positiven Flanke des Taktes Clk_AHT_in. Da die Daten kontinuierlich eintreffen, ist es wichtig, daß das Modul AHT_In auch in jedem Takt für Daten aufnahmenbereit ist.

Das Modul AHT_In umfaßt zwei Prozesse: den Prozeß “AHTIN” und den Prozeß “Payload”. Der AHTIN-Prozeß nimmt den 5 Byte großen Zellkopf an und gibt davon 4 Bytes (ohne HEC-Byte: Header Error Control) im Two-Way-Handshake-Modus an das Zellkopfübersetzer-Modul “HT” weiter. Der Payload-Prozeß nimmt die 48 Bytes der Nutzlast an und gibt sie an den CELL_FIFO-Speicher weiter.

4.1 Synthese mit Synopsys BC

Die Abbildung 2 zeigt die Simulation nach der Synthese mit dem Synopsys BC.

Bei dieser Synthese wurde im wesentlichen derselbe VHDL-Quellcode verwendet wie bei der funktionalen Simulation, mit der Ausnahme, daß die vom BC geforderten Taktzyklen eingefügt werden, um den VHDL-Modellierungs-Regeln für die Synthese im “superstate fixed” I/O-Modus zu genügen.

Das Bild zeigt zwei Probleme:

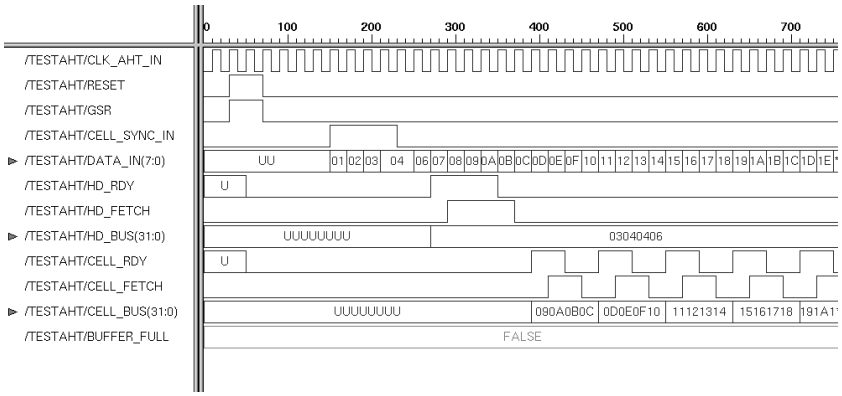
1. Das erste Datenbyte erscheint nicht synchron mit dem Zellstartsignal Cell_Sync_In. Dies ist daran zu erkennen, daß das Signal “Head_bus” das die Daten an das Zellkopfübersetzungs-Modul weitergibt, nicht korrekt mit dem ersten Byte beginnt, (“010203..”) sondern um zwei Bytes verschoben: (“030404..”. Das Byte ‘05’ darf nicht erscheinen, es ist das “HEC” Byte und wird verworfen.

2. Die Zellworte sind verschoben. Dies ist daran zu erkennen, daß die Nutzdaten (“Payload”) nicht mit dem sechsten Byte (“0607...”) beginnen, (Signal “Cell_Bus”), sondern mit dem neunten Byte (“090A..”).

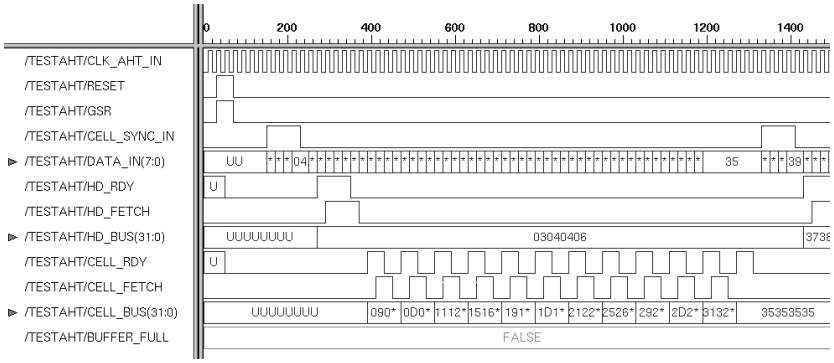
4.1.1 Graphische Darstellung der FSM’s

Mit Hilfe des Zusatzwerkzeugs “BC_View^{TM}” (BC_View ist ein Handelsname von Synopsys), ist es möglich, den Kontrollfluß bzw. die FSM graphisch darzustellen. Die Abbildung 3 zeigt die mit BC_View dargestellte FSM des Prozesses AHTIN ohne Synchronisierung des ersten Datenbytes. Die schwarz ausgefüllten Kreise stellen die einzelnen Zustände dar, die Verbindungen der Zustände stellen die Zustandsübergänge dar.

Das Bild der FSM des Prozesses Payload nach dem “ersten Versuch” zeigt Abbildung 4 (linke Seite).



Simulation nach der Synthese. Synchronisation mit Cell_Sync_In nicht korrekt



Simulation nach der Synthese. Zellworte sind verschoben.

Abbildung 2: Simulation des Moduls AHT_IN nach der Synthese ohne Synchronisation des ersten Datenbytes. Die oberen Skalen zeigen die Zeit in ns

4.1.2 Modifizierung des VHDL-Quellcodes für eine korrekte Synthese

1. Die Synchronisation mit dem Zellstartsignal wird wie folgt erreicht (siehe auch Kapitel 2):

   - Durch eine Speicherung der Eingänge der FSM für den Prozeß AHTIN.
   - Durch folgende “Start-Loop” für die beiden zu synchronisierenden Signale: Cell_Sync_In und Data_In:

```
start_loop: loop     -- fastest handshaking
  Hd_reg(31 downto 24) := Data_In;
  IF  (Cell_Sync_In = ’1’) THEN
    Wait UNTIL Clk_AHT_In’EVENT AND Clk_AHT_In = ’1’;
    exit start_loop;
  else
    Wait UNTIL Clk_AHT_In’EVENT AND Clk_AHT_In = ’1’;
```

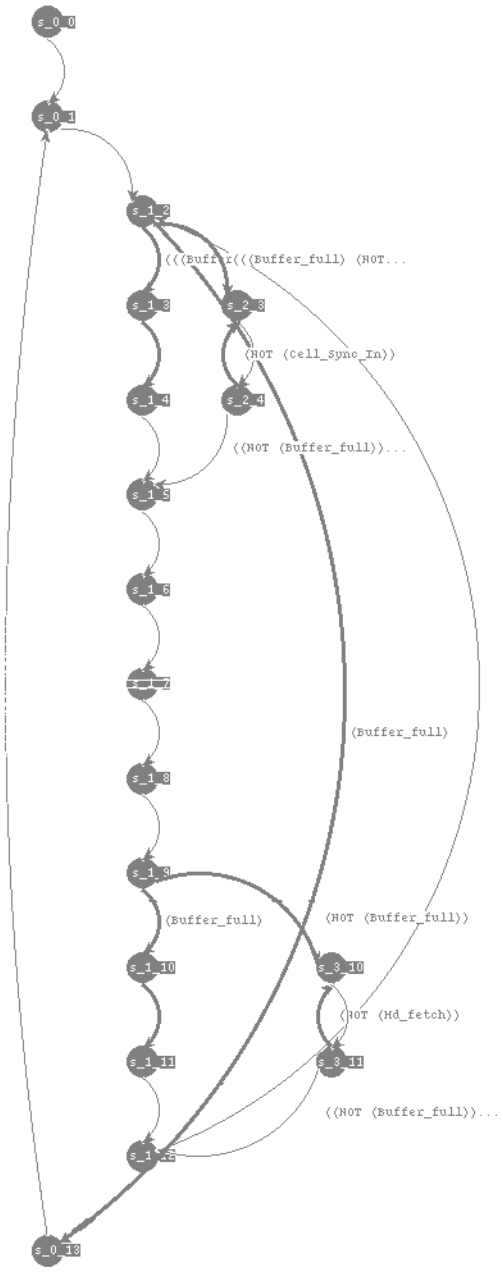
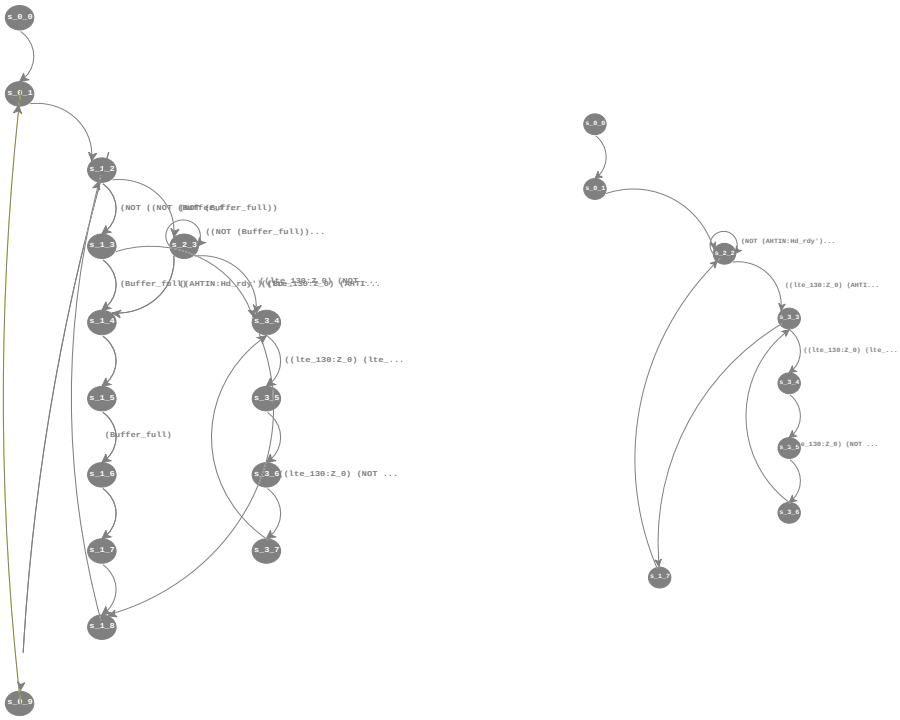


Abbildung 3: FSM des Prozesses AHTIN nach der Synthese ohne Synchronisation des ersten Datenbytes.

```
  END IF;
End loop;
```

Die FSM des AHTIN-Prozesses mit korrekter Synchronisation zeigt Abbildung 6.

2. Die kontinuierliche Datenübertragung wird wie folgt erreicht: Der Prozeß “Pay-



FSM des Prozesses Payload, erster Versuch

Endgültige FSM des Prozesses Payload

Abbildung 4: FSM des Prozesses Payload nach der Synthese, links: erster Versuch, rechts nach der Modifikation des Quellcodes

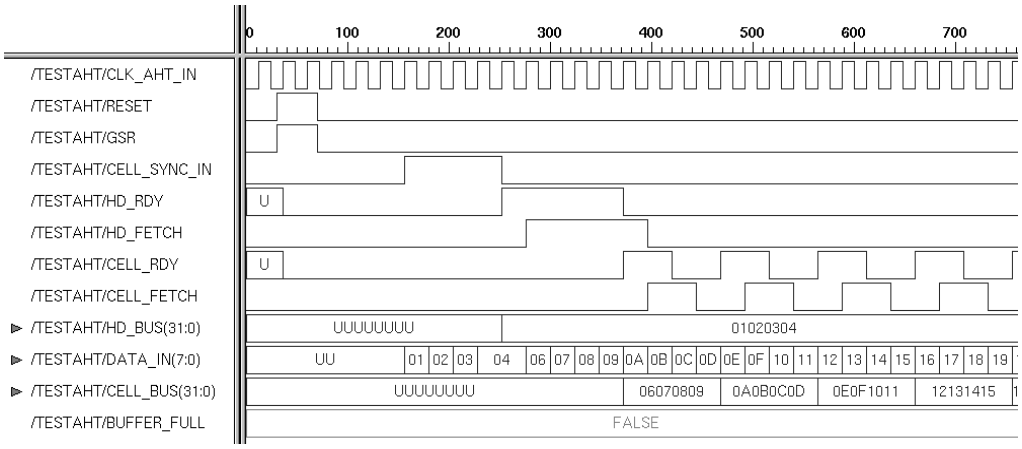
load" wird etwas früher aufgerufen (Das Signal Start_Payload wird im Prozeß AHTIN einen Kontrollschritt früher aktiv gesetzt). Die Abfrage nach dem Signal "Buffer_full" wird gestrichen, sie wird im AHTIN-Prozeß bereits durchgeführt. Dadurch tritt die Verschiebung der Übertragung, die im ersten Versuch sichtbar war, nicht auf. Die endgültige FSM des Payload-Prozesses zeigt Abbildung 4 (rechte Seite).

3. Die Kommunikation zwischen AHT_IN und dem FIFO_IN-Speicher wird von asynchron zu synchron geändert. Damit wird ein korrektes Zeitverhalten erreicht (siehe Bild 5).

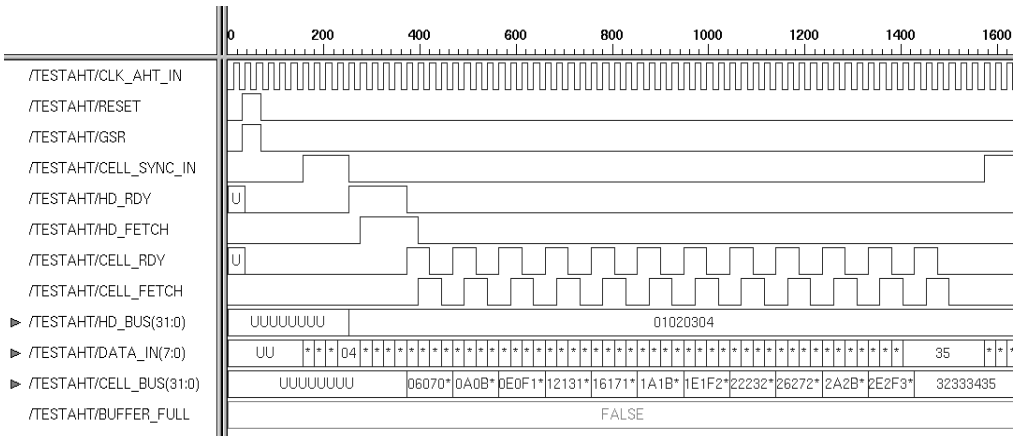
4.1.3 Diskussion der Payload-FSM's

Ein Vergleich der beiden FSM's in Abbildung 4 zeigt, daß die linke FSM fast doppelt soviele Zustände (17 Zustände) hat wie die rechte FSM (9 Zustände). Dies ist durch Streichung einer Bedingung (Signal Buffer_full) erreicht worden.

Viele Zustände in der linken FSM werden nicht erreicht, d.h. sie werden möglicherweise durch die Optimierung der FSM auch wieder eleminiert.



Simulation des AHT_IN mit korrekter Synchronisation (nach der Synthese)



Simulation AHT_IN (nach der Synthese): Übertragung einer Zelle

Abbildung 5: Simulation des Moduls AHT_IN nach der Synthese (zweiter Versuch). Die Synchronisation mit dem Startsignal ist korrekt. Die Datenübertragung ist kontinuierlich. Die oberen Skalen zeigen die Zeit in ns

Ein Vorschlag für die HLS wäre, daß bereits bei der Konstruktion der FSM diese Zustände nicht errechnet werden. Eine Möglichkeit dies zu erreichen wäre, daß die FSM in Teil-FSM's aufgeteilt wird dergestalt, daß bei der innersten Schleife begonnen wird, und das Warten auf zusätzliche Bedingungen (bzw. Signale) als Schleifen mit je zwei Zuständen davorgestellt werden.

In unserem Fall wäre die innerste Schleife die Payload-load-loop (load_loop2), die nicht aufgerollt werden darf. Davor steht die Warteschleife auf Start_Payld. Falls in der While-Schleife davor eine Bedingung eingeführt wird (z.B.: While not Buffer_full) sollte

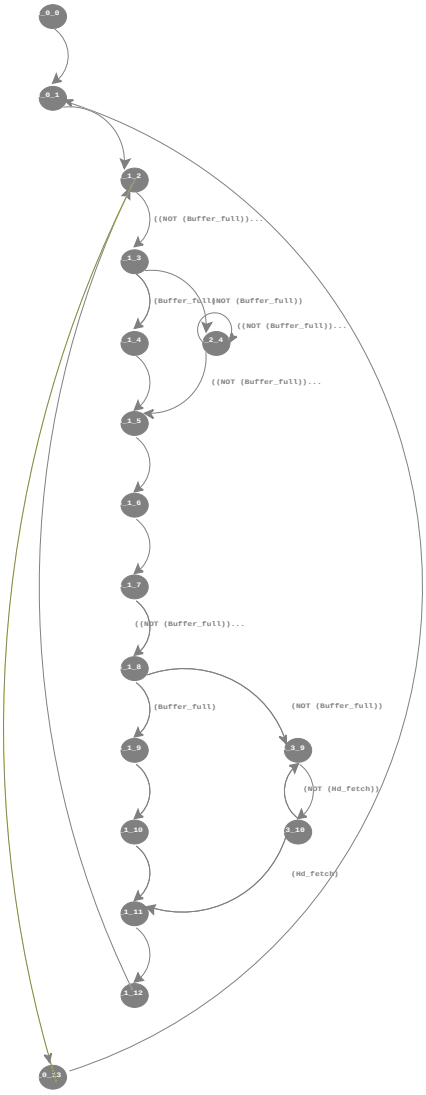


Abbildung 6: FSM des Prozesses AHTIN mit synchroner Datenannahme

dies mit einer Warteschleife realisiert werden und nicht dadurch, daß die Zustände danach verdoppelt werden.

4.1.4 Die Reservation Table

Die Abbildung 7 zeigt die sog. “Reservation Table”, die mit dem Visualisierungs-Werkzeug BC_View angezeigt wird. Die Reservation Table stellt in den Zeilen die Kontrollzyklen und in den Spalten die Operationen dar. Hier sind als Operationen die Schleifen (loops) und die Eingabe/Ausgabe-Operationen dargestellt. Die schwarzen Kreise zeigen die Operationen in den einzelnen Kontrollzyklen.

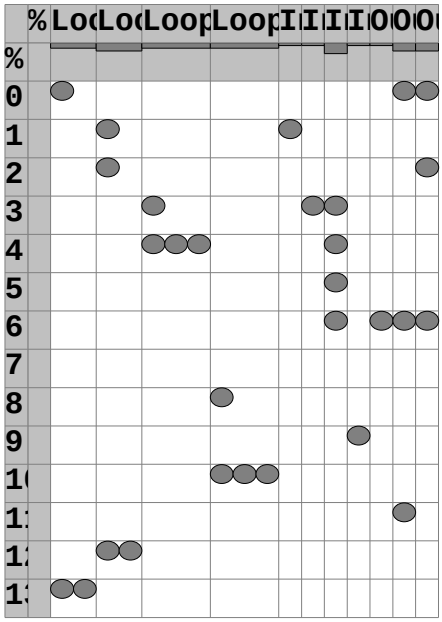


Abbildung 7: Reservation Table für den Prozeß ahtin

Die Reservation Table wird auch im “Scheduling-Report” als ASCII-Tabelle dargestellt, daher wird im folgenden die BC_View-Tabelle nicht mehr gezeigt.

4.1.5 Synthese-Daten

Aus den Scheduling-Reports (siehe Anhang) ergeben sich folgende Zahlen:

Rechenzeit und Speicherbedarf für das Scheduling:
(Sun Ultra 2 mit 640 MB RAM): 14 Sekunden.
Speicherbedarf: 8969 Kbytes.

Zuverlässigkeit:
Die Synthese wurde **fehlerlos** abgeschlossen.
Das Ergebnis der Synthese ist **akzeptabel** und **korrekt simulierbar**.
Zuverlässigkeitsmaß: 10.

Daten der synthetisierten Schaltkreise: (RT-Ebene):

Geschätzte Verzögerung: (Max. Cumulative delay) 6.86 ns.

1. **Prozeß AHTIN**

   (a) **Anzahl Kontrollzyklen:** 13 (Bei einer Taktperiode von 25 ns, simuliert mit Takt 25 ns).

   (b) **Geschätzte Schaltungsfläche (DP):**

- 4 Register, 7 I/O Ports

(c) Geschätzte Flächeneinheiten:

- Combinational Area: 40
- Sequential Area: 1030
- Summe: 1070

(d) **Anzahl der Operationen** nach SYNOPSYS-Definition: 29

(e) **Die FSM:** (siehe Bild 6).

- Anzahl der Zustände: 17
- Anzahl der Zustandsübergänge: (transitions) 22
- Steuereingänge: (control inputs) 4
- Steuerausgänge: (control outputs) 12

2. **Prozeß Payload**

(a) **Anzahl Kontrollzyklen:** 8. (Bei einer Taktperiode von 25 ns, simuliert mit Takt 25 ns).

(b) **Komponenten**

- 4 Register, 4 I/O Ports
- 1 Incrementer
- 1 Comparator

(c) **Geschätzte Schaltungsfläche (DP):**

- Combinational Area: 99
- Sequential Area: 1560
- Gesamt: 1659

(d) **Anzahl der Operationen** nach SYNOPSYS-Definition: 26

(e) **Die FSM:** (siehe Abbildung 4, rechte Seite)

- Anzahl der Zustände: 8
- Anzahl der Zustandsübergänge: (transitions) 10
- Steuereingänge: (control inputs) 4
- Steuerausgänge: (control outputs) 15

5 Das Zellkopfübersetzungsmodul HT

Das Zellkopfübersetzungsmodul (HT) erhält den Zellkopf vom Modul AHT_IN und adressiert mit den Feldern VPI/VCI die Routing-Tabelle. Die Routing-Tabelle gibt die neuen VPI/VCI-Werte zusätzlich mit einer Routing-Information zurück. Die neuen VPI/VCI-Werte werden in den Zellkopf eingesetzt und in den Header-FIFO abgespeichert.

5.1 Synthese mit Synopsys BC

Das Zellkopfübersetzungsmodul hat im wesentlichen nur Eingabe- und Ausgabefunktionen, verbunden mit Bitfelder-Verschiebungen, daher wird eine Synthese mit dem Synopsys BC im “cycle fixed” I/O- Modus versucht.

Dabei ergeben sich folgende Probleme:

1. Die synchronen WAIT-Anweisungen für ein Signal, z.B.:

   “WAIT UNTIL Clk_com’EVENT and Clk_com = ’1’and Hd_rdy = ’1’;”

   funktionnieren nicht mehr. Sie müssen durch “while”-Schleifen ersetzt werden. z.B. passend zum obigen Fall:

   ```
   while Hd\_rdy = ’0’ loop
     ‘WAIT UNTIL  Clk\_com’EVENT and Clk\_com = ’1’;
   end loop;
   ```

2. Dazu müssen an allen Stellen, an denen zusätzliche Kontrollschritte vom BC gefordert werden, “WAITS” eingefügt werden. Im “superstate fixed” I/O-Modus werden viele dieser Kontrollschritte vom BC automatisch angelegt, d.h. die Vorbereitung des Quellcodes für die Synthese im “cycle fixed” Modus ist wesentlich aufwendiger als im “superstate fixed” Modus.

Daher wird auch hier für die Synthese der “superstate fixed” I/O-Modus gewählt.

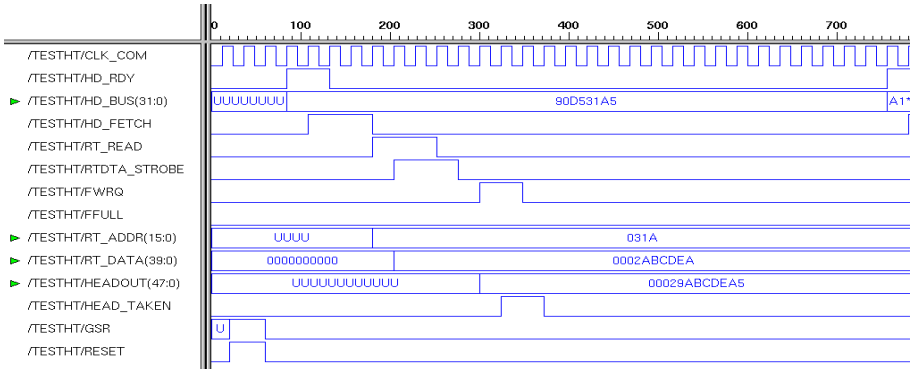
Das Modul “HT” kann in der vorliegenden VHDL-Beschreibung mit dem BC fehlerfrei synthetisiert werden, wenn an einigen Stellen Wait-Anweisungen nach den BC-Regeln, (zusätzliche Kontrollschritte) eingeführt werden.

Das Ergebnis der Synthese kann hierarchisch oder “flach” als VHDL-RTL-Datei ausgegeben werden. Die “flache” Datei ist wieder simulierbar.

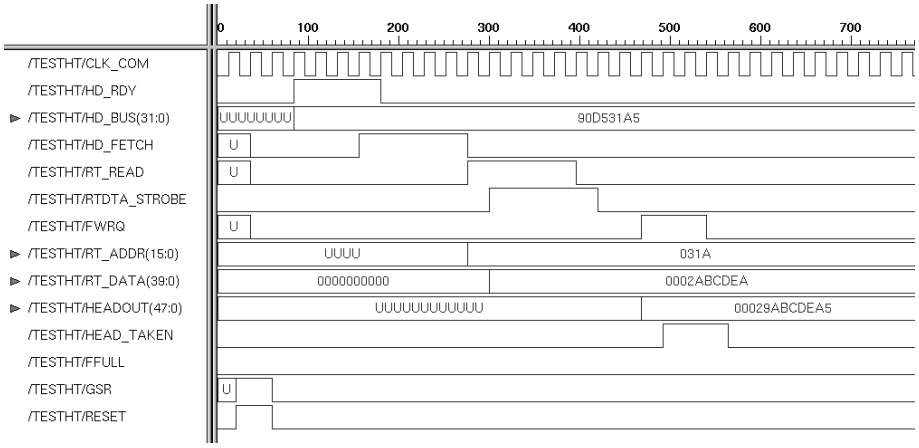
Die Ergebnisse der Synthese zeigt die Simulation (Bild 8).

Vergleicht man die funktionale Simulation (siehe oberes Bild) mit der Simulation der im “superstate fixed” I/O-Modus synthetisierten Verhaltenbeschreibung, so sieht man daß zusätzlich Kontrollschritte vom BC eingefügt wurden. Die funktionale Simulation dauert, gemessen von der positiven Flanke des Signals Hd_rdy bis zur positiven Flanke des Signals Head_taken 10 Takte, die Simulation der synthetisierten Schaltung dauert 17 Takte. Der Behavioral Compiler liefert uns hier eine zusätzliche Verzögerung für die Zellkopfübersetzung von 7 Taktzyklen.

Die Abbildung 9 zeigt den Kontrollpfad (die FSM) für den Prozess “htproc”, sichtbar gemacht mit BC_View.



Funktionale Simulation des HT-Prozesses



Simulation des synthetisierten HT-Prozesses

Abbildung 8: Simulationen des HT-Prozesses. Die oberen Skalen zeigen die Zeit in ns

5.1.1 Scheduling-Daten für den Prozess “htproc”

Aus den Scheduling-Reports (siehe Anhang) ergeben sich folgende Zahlen:

Rechenzeit und Speicherbedarf für das Scheduling:
(Sun Ultra 2 mit 640 MB RAM):
Rechenzeit: 9 Sekunden.
Speicherbedarf: 8537 Kbytes

Zuverlässigkeit:
Die Synthese wurde **fehlerlos** abgeschlossen.
Das Ergebnis der Synthese ist **akzeptabel** und **korrekt simulierbar**.
Zuverlässigkeitsmaß: 10.

Daten des synthetisierten Schaltkreises: (RT-Ebene):

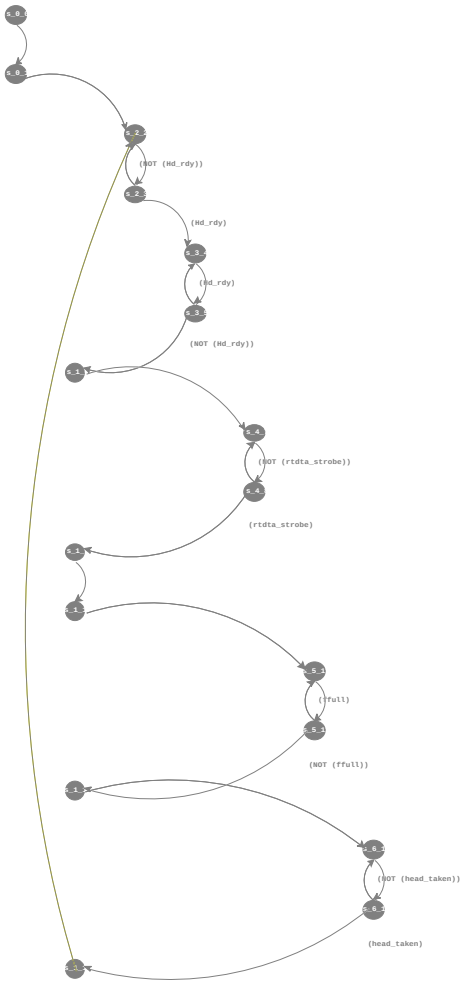


Abbildung 9: FSM des HT-Prozesses

Geschätzte Verzögerung: 0 nsec.

Verzögerung 0 ns ist darauf zurückzuführen, daß keine Komponenten mit arithmetischen Operationen verwendet werden. Für das Laden der Register werden keine Verzögerungen geschätzt.

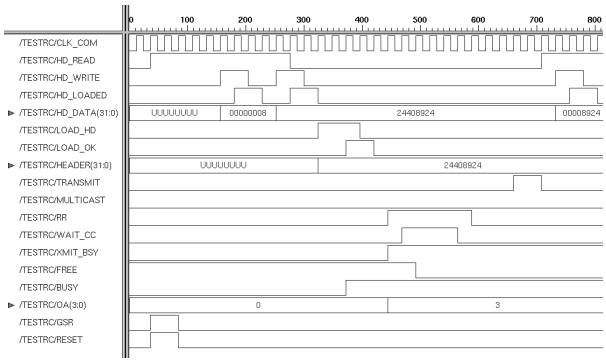
1. **Anzahl Kontrollzyklen:** 17. (Bei einer Taktperiode von 25 ns, simuliert mit Takt 50 ns).
2. **Geschätzte Schaltungsfläche (DP):**
   - combinational area 186
   - sequential area 1670
   - gesamt 1856
3. Komponenten:

(a) 4 Register + 11 I/O Ports

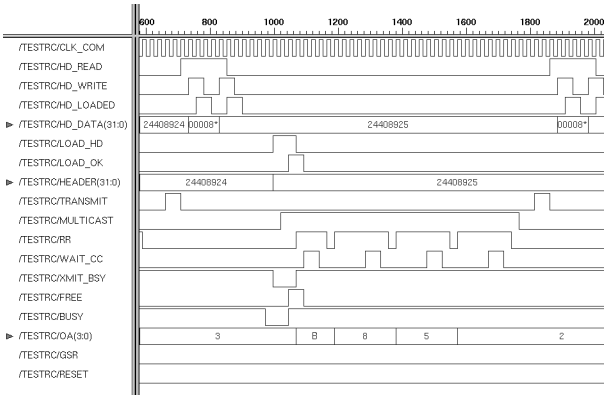
4. **Anzahl der Operationen** nach SYNOPSYS-Definition: 44
5. **Daten der FSM** :
    - Anzahl der Zustände: 17
    - Anzahl der Zustandsübergänge: (transitions) 22
    - Steuereingänge: 3.
    - Steuerausgänge: 11.

6 Die Routingsteuerung RC

Die Routingsteuerung (Routing Control RC) nimmt den Zellkopf aus dem FIFO-Speicher, trennt die Routing-Information ab, interpretiert sie und gibt entsprechende Verbindungsanfragen (RR) an die Verbindungs-Steuerung (CC) weiter.



Funktionale Simulation der Routing-Steuerung (Singlecast)



Funktionale Simulation der Routing-Steuerung (Multicast)

Abbildung 10: Simulation des RC-Moduls. Die oberen Skalen zeigen die Zeit in ns

Die Routing-Information (der Routing-Tag) enthält die Information, ob eine Einzelverbindung (Singlecast) oder eine Mehrfachverbindung (Multicast) gefordert ist.

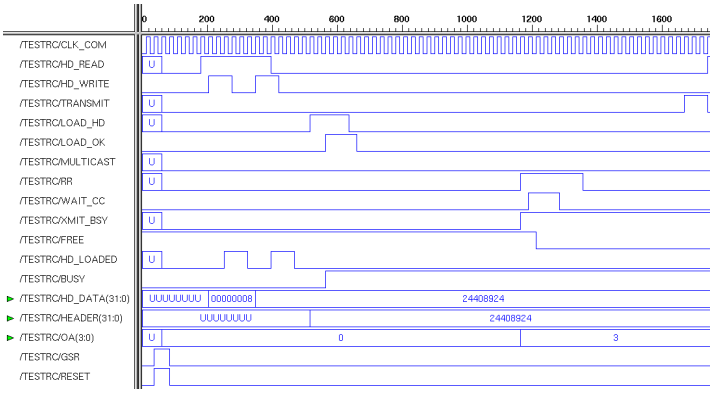
Die Verhaltensbeschreibung der VHDL-Architektur enthält zwei Prozesse: RCIN, der das Interface zum FIFO-Speicher und zur Verbindungssteuerung bedient und RCOUT, der die Verbindung zur Schieberegister-Steuerung unterhält.

Der Prozeß RCIN ist wesentlich komplizierter als das Zellkopf-Übersetzungs-Modul. Es enthält eine Schleife, die nicht aufgerollt werden darf, sowie mehrfach verschachtelte IF-Strukturen.

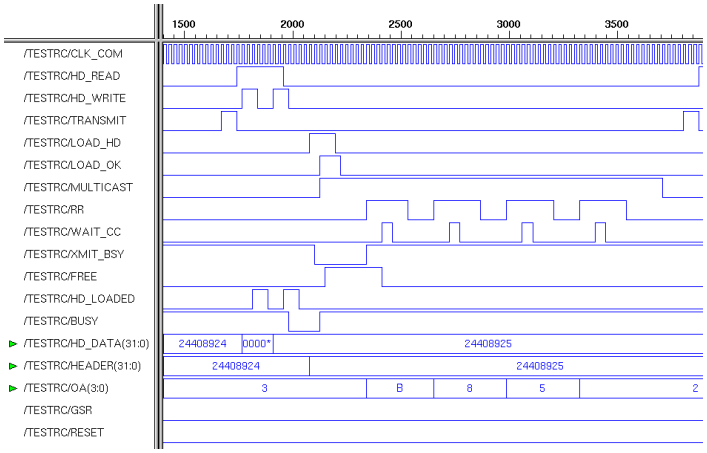
6.1 Synthese mit Synopsys BC

Folgende Änderungen des VHDL-Quellcodes waren nötig, um das Modul mit dem Synopsys BC zu synthetisieren:

Während bei der funktionalen Simulation eine synchrone Übertragung des Zellkopfs plus Routing-Information aus dem Zellkopf-Speicher angenommen wurde, kann dies für eine Synthese mittels Synopsys Behavioral Compiler nicht mehr durchgeführt werden, da durch zusätzliche Verzögerungen Unsicherheiten in der Datenübertragung auftreten. In der Verhaltensbeschreibung wird die Übernahme des Zellkopf mittels eines Two Way Handshaking-Protokolls eingeführt. Eine weitere Änderung bedeutet die Einführung meh-



Simulation der Routing-Steuerung nach der HLS (Singlecast)



Simulation der RC nach der High-Level-Synthese (Multicast)

Abbildung 11: Simulation des RC-Moduls nach der Synthese. Die oberen Skalen zeigen die Zeit in ns

rerer Kontrollschritte durch zusätzliche WAIT's (11 Kontrollschritte für den Prozeß RCIN und 3 für den Prozeß RCOUT) nach den Regeln die für die Synthese mit dem BC gelten.

Dabei ist anzumerken, daß jede Anweisung, die ein synchrones Warten auf ein Signal

einleitet, (z. B.: Wait UNTIL Clk_com’EVENT AND Clk_com = ’1’ and Hd_write = ’1’;) vom Compiler in eine Schleife umgewandelt wird.

Das Modul RC kann nach den o.g. Änderungen in der vorliegenden VHDL-Beschreibung mit dem Synopsys BC im “superstate fixed” I/O-Modus fehlerfrei synthetisiert werden.

Die Ergebnisse der funktionalen Simulation mit“Two Way Handshaking”-Kommunikation zum Head-FIFO zeigt die Abbildung 10.

Die Simulation der VHDL-RTL-Datei nach der High-Level-Synthese zeigt die Abbildung 11. Durch die zusätzlich eingeführten Kontrollschritte wird die Datenübertragung

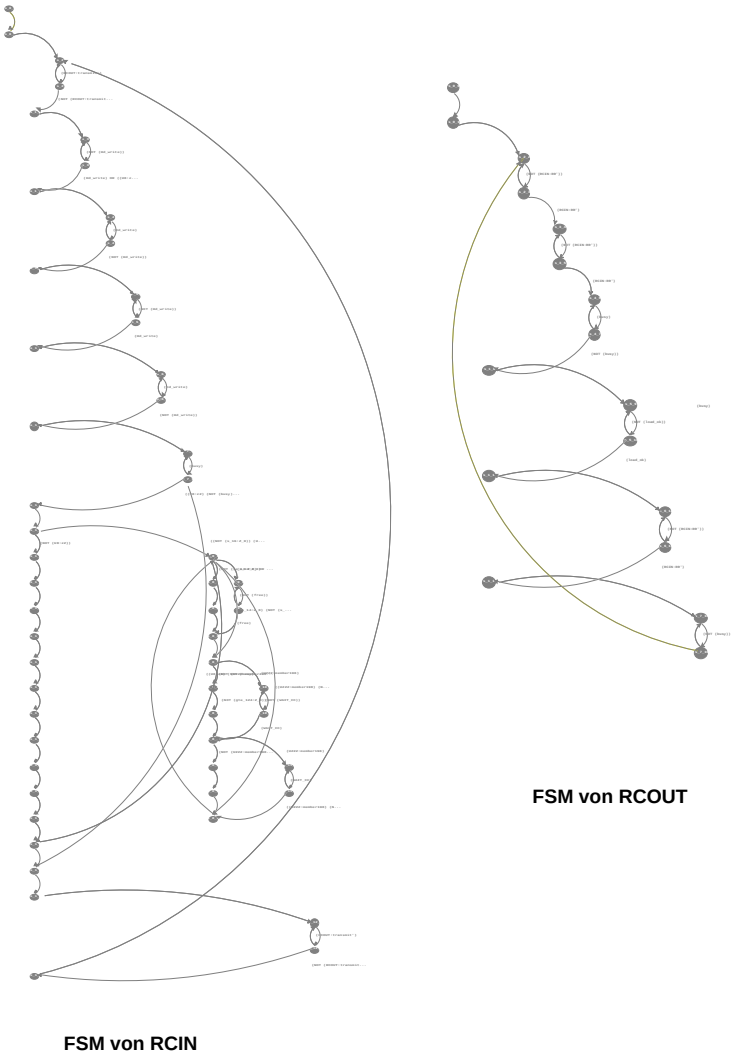


Abbildung 12: Darstellung der Finite State Machines der RC-Prozesse

verlangsamt. Während bei der funktionalen Simulation (Abbildung 10) von der positiven Flanke von “Head_read” bis zur positiven Flanke von “transmit” 25 Takte verbraucht werden, sind es nach der High-Level Synthese 60 Takte. Da der Board-Takt mit 25 ns angesetzt wird, ist dies ein Unterschied von 625 ns zu 1500 ns.

Ein Teil dieser Verzögerung ist darauf zurückzuführen, daß die Schleife, die aus dem Adressbit im“Routing-Tag” die Ausgangsadresse OA(i) ableitet, 14 mal durchlaufen wird. In der funktionalen Simulation ist in dieser Schleife kein Kontrollschritt enthalten, während bei der HLS der BC hier einen Kontrollschritt verlangt. Für den “Multicast”-Fall ist diese Lösung annehmbar. Für den “Singlecast”-Fall kann an dieser Stelle statt der Schleife eine einfache Transformation eines eins-aus-sechzehn Codes in einen 4-Bit Code durchgeführt werden, die in einem Kontrollschritt erledigt wäre. Dies würde eine Verbesserung von ca. 13 Takten für den “Singlecast”-Fall bringen.

Die Kontrollpfade (die FSM’s) der Prozesse RCIN und RCOUT können mit Hilfe des Synopsys-Werkzeugs BC_VIEW graphisch dargestellt werden. (Siehe Bild 12).

6.1.1 Scheduling-Daten

Aus den Scheduling-Reports ergeben sich folgende Daten:

Rechenzeit und Speicherbedarf für das Scheduling:
(Sun Ultra 2 mit 640 MB RAM):
Rechenzeit: 55 sec.
Speicherbedarf: 22585 Kbytes.

Zuverlässigkeit:
Die Synthese wurde **fehlerlos** abgeschlossen.
Das Ergebnis der Synthese ist **akzeptabel** und **korrekt simulierbar**.
Zuverlässigkeitsmaß: 10.

Geschätzte Verzögerung: 9.36 ns

Daten des synthetisierten Schaltkreises: (RT-Ebene):

1. **Prozeß RCIN:**
   (a) **Anzahl Kontrollzyklen:** 37. (Bei einer Taktperiode von 25 ns).
   (b) **Geschätzte Schaltungsfläche (DP):**
   - Combinational area 218
   - Sequential area 2400
   - Gesamt 2618

   (c) **Komponenten**
   - 11 Register + 16 I/O Ports
   - 1 Komparator (5 Bit)
   - 1 Dekrementer (5 Bit)

   (d) **Anzahl der Operationen** nach SYNOPSYS-Definition: 95
   (e) **Die FSM:**
   - Anzahl der Zustände: 55
   - Anzahl der Zustandsübergänge: (transitions) 73.
   - Steuereingänge: (control inputs) 12

- Steuerausgänge: (control outputs) 53

2. **Prozeß RCOUT:**

   (a) **Anzahl Kontrollzyklen:** 16.
   (Bei einer Taktperiode von 25 ns).

   (b) **Geschätzte Schaltungsfläche (DP):**

   - Combinational area 32
   - Sequential area 710
   - Gesamt 742

   (c) **Komponenten**

   - 2 1-Bit Register

   (d) **Anzahl der Operationen** nach SYNOPSYS-Definition: 47

   (e) **Die FSM:**

   - Anzahl der Zustände: 17
   - Anzahl der Zustandsübergänge: (transitions) 23
   - Steuereingänge: (control inputs) 3
   - Steuerausgänge: (control outputs) 12

7 Die Schieberegister-Steuerung SRC

Die Schieberegister-Steuerung übernimmt den Zellkopf von der Routing-Steuerung und die Nutzlast direkt aus dem “Payload-FIFO”. Um eine sichere Übertragung zu gewährleisten, werden sowohl Zellkopf als auch Nutzlast in Einheiten zu 32 Bit mit dem 4B/5B Verfahren verschlüsselt. Jeder Einheit wird ein Startzeichen vorangestellt, wobei der Zellkopf ein eigenes Startzeichen erhält, damit der Zellstart eindeutig erkennbar ist.

Die Verschlüsselungsfunktion wird im Package “utilc_src” beschrieben. Hier ist es angebracht, das Synopsys-Attribut “preserve_function” zu wählen. Dadurch wird die Verschlüsselungsfunktion vom Scheduler wie eine Komponente behandelt. Es wird vermieden, daß bei jeder Verschlüsselungs-Aktion (je zwei pro Byte) der Verschlüsselungs-Quellcode inline in die VHDL-Beschreibung hineinkopiert wird.

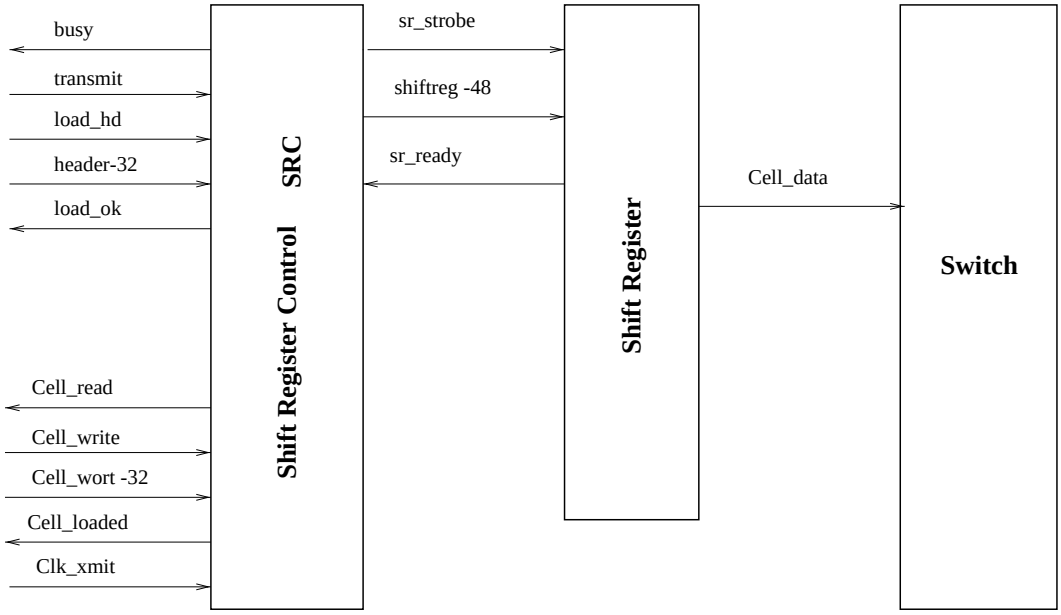


Abbildung 13: Schnittstelle des SRC-Moduls mit externem Schieberegister

Die Einheiten werden in ein 48 Bit breites Schieberegister geladen und seriell durch den Switch geschoben, wenn die Verbindung geschaltet ist.

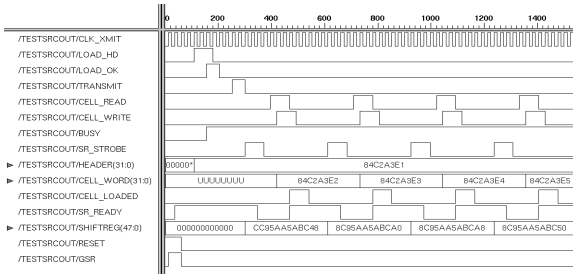
7.1 Synthese mit Synopsys BC

Das Schieberegister wurde anfangs in VHDL durch eine Schleife beschrieben, in der - durch eine ‘Wait-Anweisung’ - ein Taktzyklus den Schieberythmus angibt, mit dem die Datenbits serialisiert werden.

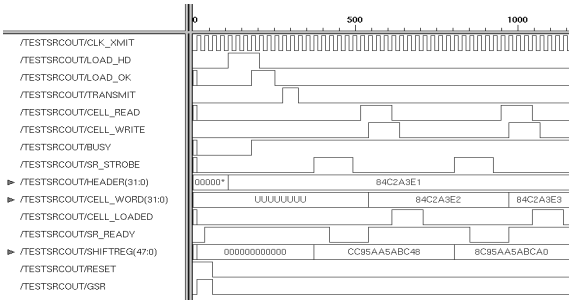
Die Synthese im “superstate fixed” I/O-Modus zeigt, daß der BC in der Schieberegisterschleife zwei Taktzyklen hinzufügt, sodaß die übertragenen Daten verfälscht werden.

Es liegt daher nahe, das Schieberegister entweder in einer RT-Beschreibung zu integrieren, oder als externes Modul zu betrachten. Letzteres ist auch insofern sinnvoll, als das

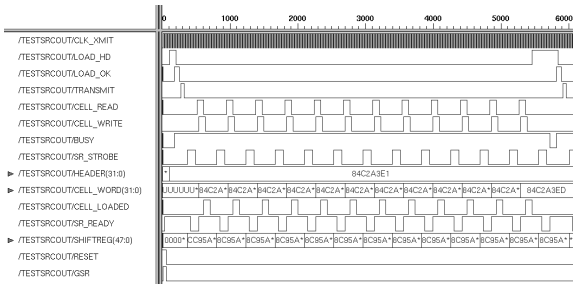
Schieberegister auf das Spannungsniveau des ATM-Switches, der in PECL-Technologie vorliegt, angehoben werden muß. Für das Schieberegister lohnt es sich auch, eine Technologie zu wählen, die eine sehr hohe Taktfrequenz erlaubt, um die Datenübertragung zu beschleunigen. Die Schnittstellen des SRC mit externem Schieberegister zeigt Bild 13.



Funktionale Simulation des SRC ohne Schieberegister



Simulation des SRC nach der Synthese



Simulation des SRC nach der Synthese: Übertragung einer Zelle

Abbildung 14: Simulation des SRC-Moduls ohne Schieberegister. Die oberen Skalen zeigen die Zeit in ns

Die funktionale Simulation wird mit der Verhaltensbeschreibung für die Synthese mit BC durchgeführt. Die Verhaltensbeschreibung wurde mit dem Programm ‘bc_check_design’ geprüft, das nur dann fehlerfrei abschließt, wenn alle geforderten Taktzyklen (in unserem Fall 6 Taktzyklen) eingeführt wurden. Die funktionale Simulation zeigt das obere Diagramm, die Simulation nach der Synthese die beiden unteren Diagramme in Abbildung 14.

Ein Vergleich der Simulationen vor und nach der Synthese zeigt, daß die Zeitdifferenz pro Registerübergabe ca. 125 ns beträgt. Damit ergibt sich eine effektive Verzögerung

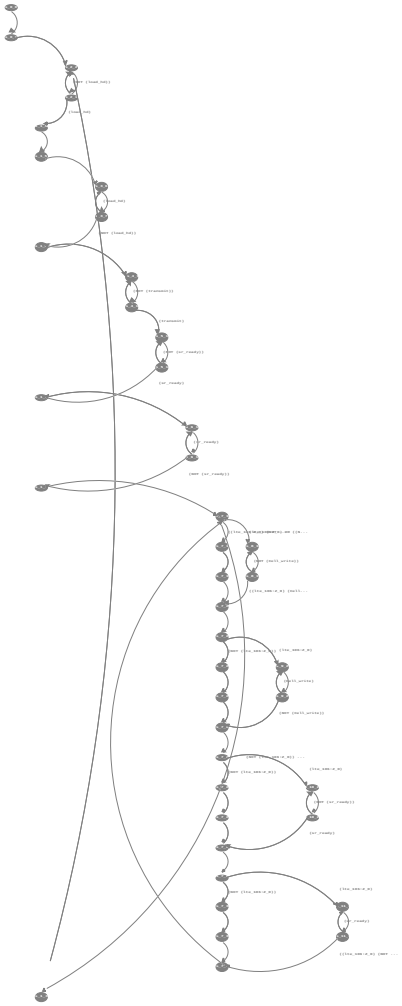


Abbildung 15: Darstellung der Finite State Machine des SRC-Prozesses

von 5 Taktzyklen. Aus dem Simulationsdiagramm vor der Synthese entnimmt man, daß für eine Registerübergabe 300 ns benötigt werden (12 Takte). Nach der Synthese werden für eine Registerübergabe 17 Takte benötigt. Damit beträgt die Verzögerung, die durch die BC-Synthese verursacht wird, ca. 42 Prozent.

Das in Abbildung 15 gezeigte Schaubild der FSM ist mit dem SYNOPSYS-Werkzeug BC_View aufgenommen worden.

7.1.1 Synthese-Daten

Aus den Scheduling-Reports (siehe Anhang) ergeben sich folgende Zahlen:

Rechenzeit und Speicherbedarf für das Scheduling:
(Sun Ultra 2 mit 640 MB RAM):
1 min, 38 sec (98 sec).

Speicherbedarf: 19097 Kbytes.

Zuverlässigkeit:
Die Synthese wurde **fehlerlos** abgeschlossen.
Das Ergebnis der Synthese ist **akzeptabel** und **korrekt simulierbar**.
Zuverlässigkeitsmaß: 10.

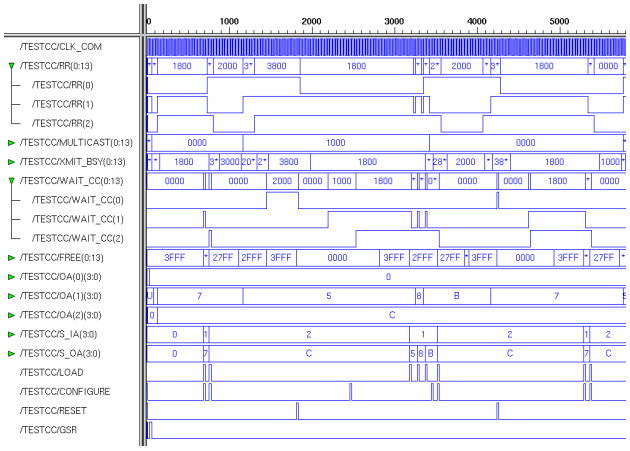
Geschätzte Verzögerung: (Cumulative delay) 9.065 ns

Daten des synthetisierten Schaltkreises: (RT-Ebene):

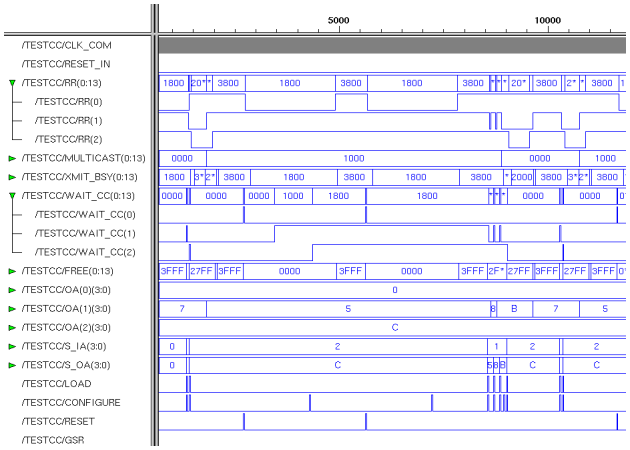
1. **Anzahl Kontrollzyklen:** 33. (Bei einer Taktperiode von 25 ns).
2. **Geschätzte Schaltungsfläche (DP):**
   - Combinational area 543
   - Sequential area 2230
   - Gesamt 2773
3. **Komponenten:**
   - 12 Register + 12 Ports
   - 1 Komparator
   - 1 Inkrementer (1 Bit-Adder).
   - 4 (4 to 5) Bit Encoder
4. **Anzahl der Operationen** nach SYNOPSYS-Definition: 109
5. **Die FSM:**
   - Anzahl der Zustände: 42
   - Anzahl der Zustandsübergänge: (transitions) 56
   - Steuereingänge: (control inputs) 5
   - Steuerausgänge: (control outputs) 30

8 Die Verbindungs-Steuerung CC

Die Verbindungs-Steuerung (Connection Control CC) nimmt die Verbindungsanfragen (Routing Requests RR) von 14 Routing-Steuerungen (RC) an und bearbeitet sie der Reihe nach. Eine Verbindungsanfrage kann ‘Singlecast’ sein, d.h. die Verbindung zu nur einem Ausgang wird angefordert, sie kann ‘Multicast’ sein, (Verbindungen zu mehreren Ausgängen) oder sie kann ‘Broadcast’ sein (Verbindung von Kanal 0 zu allen Ausgängen).



Funktionale Simulation vor Einfügen von Kontrollschritten für die Synthese



Funktionale Simulation nach Einfügen von Kontrollschritten für die Synthese

Abbildung 16: Simulation des CC-Moduls. Die oberen Skalen zeigen die Zeit in ns

8.1 Synthese mit Synopsys BC

Die zusätzlichen Kontrollschritte nach den Regeln für den “superstate fixed” I/O-Modus werden eingeführt. Der Vergleich der funktionalen Simulation (obiges Teilbild) vor Einfügen der zusätzlichen Kontrollschritte und danach (unteres Teilbild) zeigt die Abbildung 16.

Der BC akzeptiert keine “Wait” Anweisungen in Prozeduren oder Funktionen für die Scheduling-Phase. Daher muß die Prozedur “connect”, die “Waits” enthält, wieder ‘inline’ gesetzt werden.

Aus den Simulationsbildern ist folgendes ersichtlich: Eine einfache Routinganfrage von Kanal 1 (RR(1)) wird vor Einfügen der Kontrollschritte (obiges Teilbild) in 720 ns abgehandelt, danach in 1200 ns. Die Broadcast-Anfrage (RR(0)) wird oben in 1150 ns abgehandelt, darunter in 1350 ns. Bei der nächsten Verbindungsanfrage ergibt sich eine sehr große Verzögerung für die beiden folgenden Verbindungsanfragen der Kanäle eins und zwei. Im obigen Bild werden diese Anfragen noch vor Eintreffen der zweiten Broadcast-Anfrage erledigt, also innerhalb von ca. 2250 ns (für RR(1)), im unteren Bild kommt bei RR(1) und RR(2) eine Broadcast-Anfrage dazwischen und läßt RR(1) und RR(2) warten, bis die RR(0)-Anfrage abgearbeitet ist, sodaß die beiden Verbindungsanfragen auf ca. 7100 ns verzögert werden.

Die Simulation der VHDL-RTL-Datei nach der High-Level-Synthese zeigt Bild 17.

Abbildung 17: Simulation des CC-Moduls nach der Synthese. Die oberen Skalen zeigen die Zeit in ns

Die Zeitverhältnisse gegenüber der funktionalen Simulation haben sich verändert. Die erste Routing-Anfrage von Kanal 1 (RR(1)) wird in 160 ns abgehandelt, die erste Broadcast-Anfrage (RR(0)) dagegen in 2230 ns. Die zweite Broadcastanfrage muß zu lange auf die Abarbeitung warten, d. h. an dieser Stelle müßte noch Design-Arbeit aufge-

wendet werden, etwa in der Art, daß nach einem RR(0) keine weiteren RR(i) zugelassen werden, bis RR(0) beendet ist.

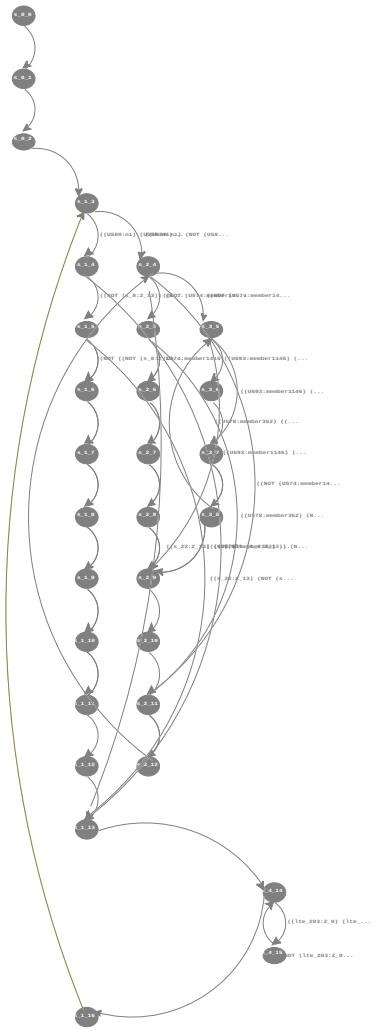


Abbildung 18: FSM der Verbindungssteuerung CC.

Das in Abbildung 18 gezeigt Schaubild zeigt die FSM der Verbindungssteuerung, aufgenommen mit dem Visualisierungs-Werkzeug BC_VIEW.

8.1.1 Synthese-Daten

Aus den Scheduling-Reports ergeben sich folgende Daten:

Rechenzeit und Speicherbedarf für das Scheduling:
(Sun Ultra 2 mit 640 MB RAM):
Rechenzeit: 14 min, 33 sec (873 sec) .
Speicherbedarf: 118105 Kbytes.

Zuverlässigkeit:

Die Synthese wurde **fehlerlos** abgeschlossen.
Das Ergebnis der Synthese ist **akzeptabel** und **korrekt simulierbar**.
Zuverlässigkeitsmaß: 10.

Geschätzte Verzögerung: (Cumulative Delay) 10.38 ns

Prozess CCIN:
Daten des synthetisierten Schaltkreises: (RT-Ebene):

1. **Anzahl Kontrollzyklen:** 16. (Bei einer Taktperiode von 25 ns).
2. **Geschätzte Schaltungsfläche (DP):**
   - Combinational area 3634
   - Sequential area 9420
   - Gesamt 13054

   (a) **Komponenten:**
   - 29 Register + 11 I/O-Ports
   - 1 Komparator (4 bit)
   - 1 Inkrementer (4-Bit-Adder).
   - 1 Inkrementer (31-Bit-Adder)
3. **Anzahl der Operationen** nach SYNOPSYS-Definition: 80.
4. **Die FSM:**
   - Anzahl der Zustände: 30
   - Anzahl der Zustandsübergänge: (transitions) 42.
   - Steuereingänge: (control inputs) 25
   - Steuerausgänge: (control outputs) 99

9 Das Datenannahme-Modul RD

Für jeden Kanal existiert ein “Datenannahme”-Modul (RD) am Ausgang des ATM-Switches der die seriellen Bitströme empfängt. Die Daten werden in Einheiten zu je 48 Bit übertragen. Jede 48-Bit Einheit ist im 4B/5B Verfahren codiert und trägt zu Beginn ein Startzeichen, wobei zwischen Zellkopf- und Nutzlast- Startzeichen unterschieden wird.

Analog zur Schieberegister-Steuerung SRC wird auch hier das Schieberegister für die Synthese mit dem BC entfernt. Die sich daraus ergebenden Schnittstellen sind in Abbildung 19 ersichtlich.

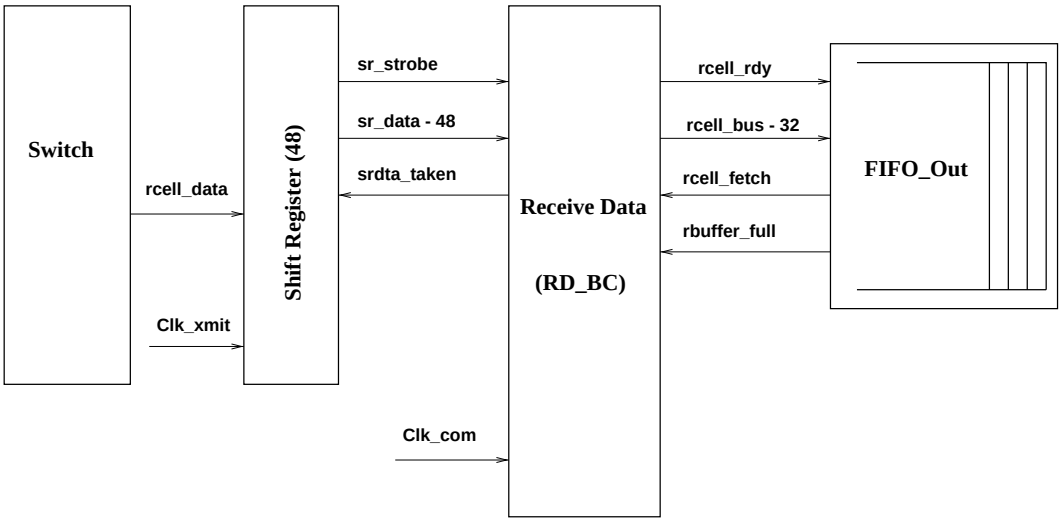


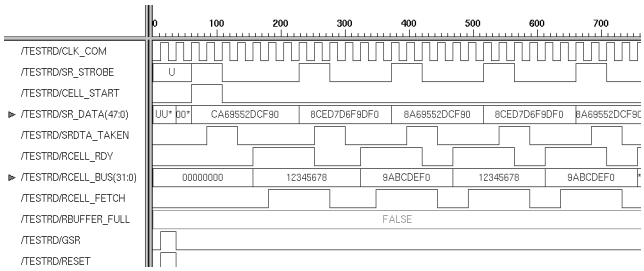
Abbildung 19: Schnittstellen des RD-Moduls nach Separierung des Schieberegisters

9.1 Synthese mit dem Synopsys BC

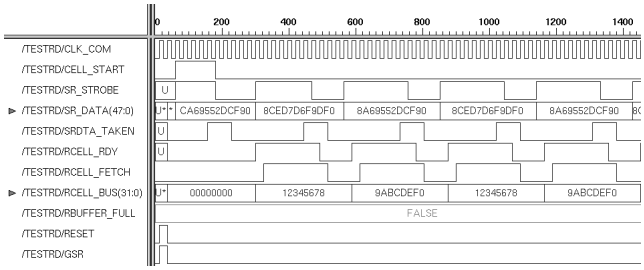
Auch hier wird versucht, wie im Modul “Schieberegistersteuerung” (SRC) für die 5B/4B Dekodierung das Attribut “preserve_function” einzusetzen. Das Attribut bewirkt, daß die Dekoder-Funktion wie eine Komponente behandelt wird. Dadurch wird die FSM wesentlich einfacher, die Steuereingänge und Steuerausgänge werden stark reduziert. Im “elaborate”-Schritt tritt jedoch ein schwerwiegender und vom Programmierer nicht korrigierbarer BC-Programmfehler auf. Daher wird hier das Ergebnis der Synthese ohne das Attribut “preserve_function” gezeigt.

Die Abbildung 20 zeigt ein Vergleich zwischen funktionaler Simulation und der Simulation nach der Synthese mit dem Synopsys BC. Das obere Diagramm zeigt die funktionale Simulation, die beiden unteren Diagramme zeigen die Simulation nach der Synthese.

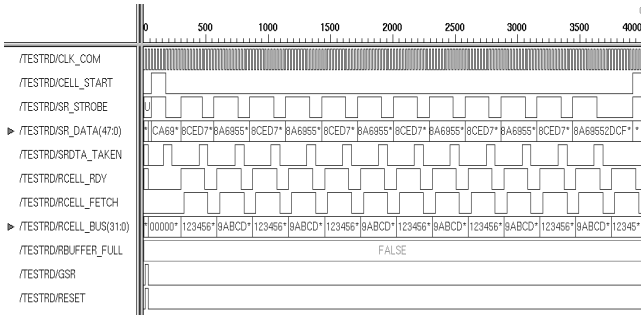
Bei der funktionalen Simulation werden für eine Registerübergabe 6 Takte (150 ns) benötigt, bei der Simulation des Moduls RD nach der Synthese werden 11 Takte (275 ns) benötigt. Dies entspricht einer Verzögerung von ca. 83 Prozent.



Funktionale Simulation des Moduls RD



Simulation des Moduls RD nach der Synthese



Simulation von RD nach der Synthese: Übergabe einer ganzen Zelle

Abbildung 20: Simulation des RD-Moduls ohne Schieberegister. Die oberen Skalen zeigen die Zeit in ns

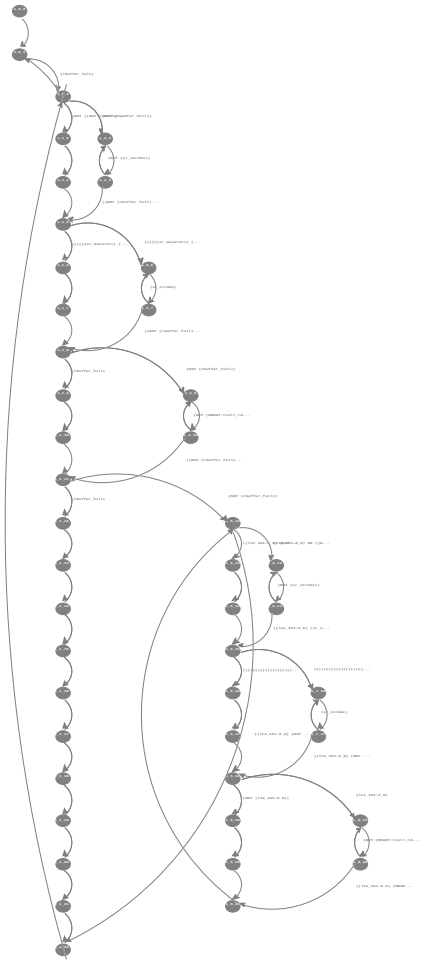
9.1.1 Die automatisch generierte FSM des RD

Das in Abbildung 21 gezeigten Schaubilder der FSM's sind mit dem SYNOPSYS-Werkzeug BC_View aufgenommen worden.

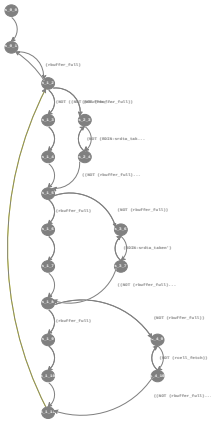
9.1.2 Synthese-Daten

Aus den Scheduling-Reports (siehe Anhang) ergeben sich folgende Zahlen:

Rechenzeit und Speicher für das Scheduling:
(Sun Ultra 2 mit 640 MB RAM):
2 h, 0 min, 58 sec. (7258 sec)
Benötigter Speicher: 603641 Kbytes.



FSM des Prozesses RDIN



FSM des Prozesses RDOUT

Abbildung 21: Darstellung der Finite State Machines der RD-Prozesse. Die linke FSM gehört zum Prozeß RDIN, die rechte FSM zum Prozeß RDOUT

Zuverlässigkeit:

Die Synthese bricht mit einem Programmfehler ab, wenn das Attribut “preserve_function” im Decode-Unterprogramm gesetzt wird.

Streicht man obiges Attribut, so wird das Ergebnis der Synthese nach Abschätzung des Entwicklers **schlechter;** denn die Chipfläche wird größer, jedoch das Ergebnis ist **korrekt** simulierbar.

Zuverlässigkeitsmaß: 5.

Geschätzte Verzögerung: (Cumulative delay) 6.86 ns

Daten der synthetisierten Schaltkreise: (RT-Ebene):

1. **Prozeß RDIN**

   (a) **Anzahl Kontrollzyklen:** 22. (Bei einer Taktperiode von 25 ns.

(b) **Geschätzte Schaltungsfläche:**

- Combinational area 319
- Sequential area 3110
- Gesamt 3429

(c) **Komponenten:**

- 104 Register + 7 I/O- Ports
- 1 Komparator
- 1 Inkrementer (1 - Bit Adder)

(d) **Anzahl der Operationen** nach SYNOPSYS-Definition: 64

(e) **Die FSM:**

- Anzahl der Zustände: 45
- Anzahl der Zustandsübergänge: (transitions) 60.
- Steuereingänge: (control inputs) 248
- Steuerausgänge: (control outputs) 308

2. **Prozeß RDOUT**

(a) **Anzahl Kontrollzyklen:** 11 (Bei einer Taktperiode von 25 ns, simuliert mit Takt 40 ns).

(b) **Geschätzte Schaltungsfläche (DP):**

- Combinational area 46
- Sequential area 2930
- Gesamt 2976

(c) **Komponenten:**

- 1 Register + 7 I/O Ports

(d) **Anzahl der Operationen** nach SYNOPSYS-Definition: 32

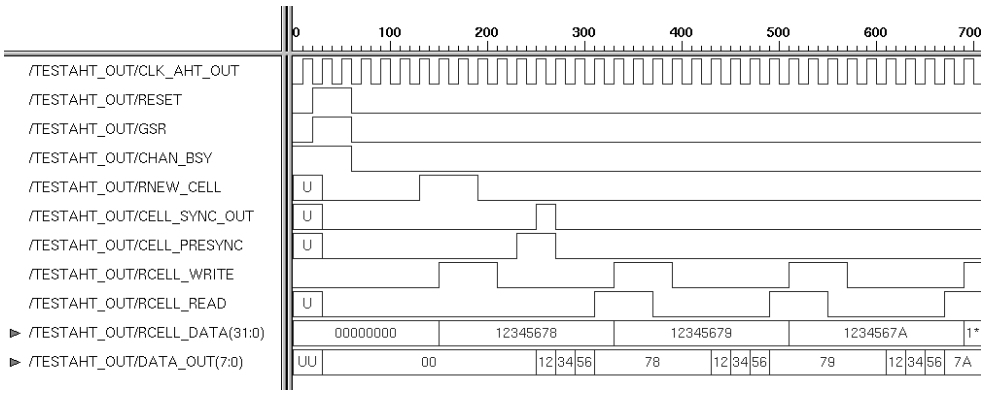
(e) **Die FSM:**

- Anzahl der Zustände: 18
- Anzahl der Zustandsübergänge: (transitions) 25
- Steuereingänge: (control inputs) 3
- Steuerausgänge: (control outputs) 8

10 Das Ausgangsmodul AHT_OUT

Das AHT-Ausgangsmodule AHT_OUT nimmt Zellen aus dem FIFO_OUT-Speicher und gibt sie auf den 8-Bit breiten Ausgangskanal. Einen Takt vor jeder Zellübertragung wird ein Cell_Presync-Signal aktiv. Mit Zellbeginn wird das Signal Cell_Sync_out aktiv gesetzt. Wichtig ist, daß die Zelldaten kontinuierlich synchron mit dem Takt Clk_aht_out auf die Leitung Data_out gegeben werden.

Das Modul AHT_OUT umfaßt zwei Prozesse: AHTOUT, der die Zelldaten in einer Breite von 32-Bit aus dem FIFO_OUT-Speicher holt und AHTOUT_Transmit, der die Daten in einer Breite von 8 Bit auf die Ausgangsleitung setzt.



Simulation des synthetisierten Moduls AHT-OUT

Abbildung 22: Simulation des Moduls AHT_OUT nach der Synthese, erster Versuch

10.1 Synthese mit Synopsys BC

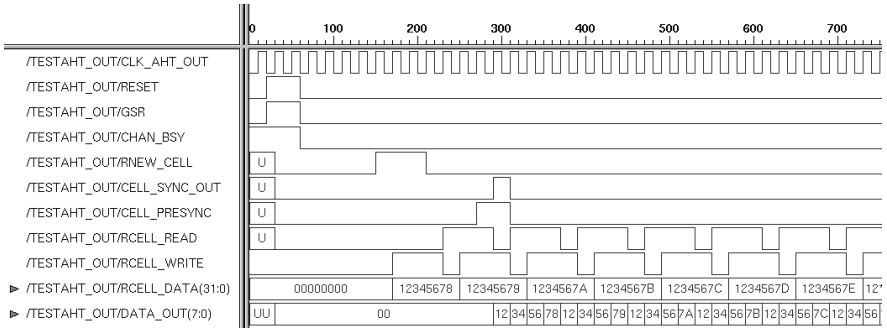
Die Abbildung 22 zeigt die Simulation nach der Synthese mit dem Synopsys BC. Bei dieser Synthese wurde im wesentlichen derselbe VHDL-Quellcode verwendet wie bei der funktionalen Simulation, mit der Ausnahme, daß die vom BC geforderten Taktzyklen eingefügt wurden, um den VHDL-Modellierungs-Regeln für die Synthese im “superstate fixed” I/O-Modus zu genügen.

Die Datenübertragung ist nicht kontinuierlich synchron mit dem Ausgangstakt Clk_aht_out.

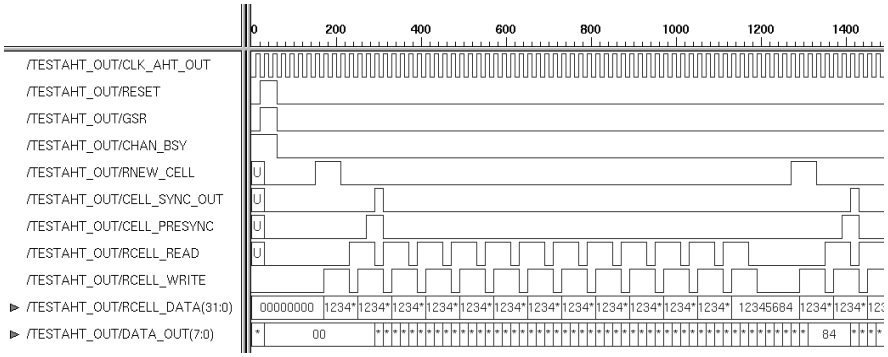
Das Bild zeigt, daß zwischen der Übertragung der einzelnen 32-Bit-Einheiten, bedingt durch die 32-Bit Übertragung von FIFO_OUT zu AHT_OUT, ein Zwischenraum von jeweils 5 Taktzyklen eingeschoben wurde.

Das Bild der FSM (siehe Abbildung 24), aufgenommen mit dem Visualisierungs-Werkzeug BC_View, veranschaulicht die Verzögerung durch zusätzliche Taktzyklen.

In Bild 24 ist auf der linken Seite die FSM des Prozesses AHTOUT dargestellt. Dieser Prozeß holt den Zellkopf aus dem FIFO_OUT-Speicher und gibt ihn an den Prozess



Simulation des AHT_OUT nach der Synthese



Simulation des AHT_OUT nach der Synthese: Übertragung einer Zelle

Abbildung 23: Simulation des Moduls AHT_OUT nach der Synthese nach der Eleminierung der Schleifen

AHTOUT_Transmit weiter. Die oberen beiden Zustände stellen die “Start” und “Reset”-Zustände dar. Die nächsten drei Zustandstripel beinhalten die Warteschleifen für die Signale “chan_bsy”, “rcell_write” und “reg_taken”.

Darunter ist eine Schleife, die 12 mal durchlaufen wird und die das Abholen und die Weitergabe der Nutzlast (Payload) bewirkt. Die Schleife benötigt 7 Zustände (Taktzyklen), obwohl nur 4 Zustände verbraucht werden dürfen, damit die Zellübertragung kontinuierlich ist.

Auf der rechten Seite des Bildes 24 ist die FSM des Prozesses AHTOUT_Transmit dargestellt. Dieser Prozeß soll kontinuierlich die Daten, die in einer Breite von 4 Byte vom Prozeß AHTOUT aus dem FIFO geholt werden, in einer Breite von 8 Bit auf die Ausgangsleitung (Data_out) legen. Auch hier benötigt die Schleife 7 Zustände. Da sich die Schleifen der beiden Prozesse zeitlich überlagern, ergibt sich eine Gesamtverzögerung von 5 Taktzyklen, wie es die Simulation zeigt.

Um die Zeitbeschränkungen einzuhalten, werden die VHDL-Quellcodes für beide

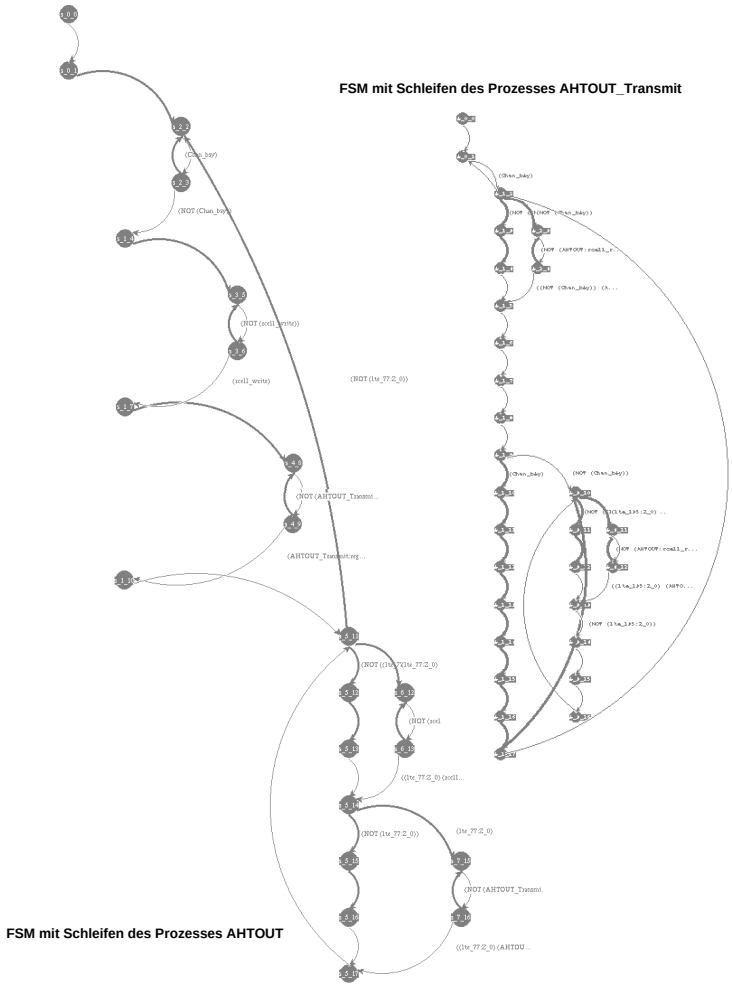


Abbildung 24: FSM's des Moduls AHT_OUT nach der Synthese, mit Schleifen. (Erster Versuch)

Prozesse des Moduls AHT_OUT wie folgt modifiziert:

- Die Schleifen werden "aneinandergereiht", d.h. anstatt das "Loop"-Konstrukt zu verwenden, wird der Schleifenkörper wiederholt aufgelistet.
- Die Abfrage nach dem Signal "chan_bsy", d.h. die Prüfung, ob der Ausgangskanal belegt ist, wird im ersten Prozeß durchgeführt und ist daher im zweiten Prozeß unnötig.

Die Simulation der modifizierten Verhaltensbeschreibung (siehe Bild 23) zeigt, daß das gewünschte Ergebnis erreicht wird.

Die FSM's der modifizierten Verhaltensbeschreibung sehen deutlich einfacher aus. Anstatt der komplizierten Schleifen mit nebenläufigen Zuständen sind einfachere Strukturen getreten, wenn sich auch die absolute Anzahl der Zustände erhöht hat. Der in Abbildung 25 gezeigte Graph zeigt die FSM des Prozesses AHTOUT ohne Schleifen und Bild

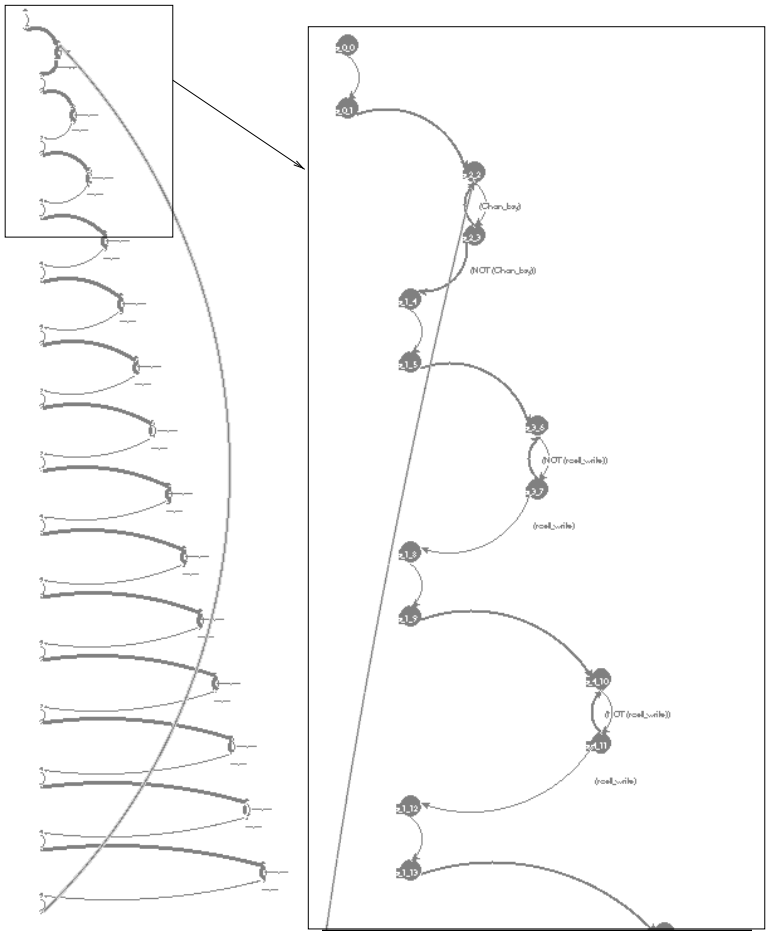


Abbildung 25: FSM des Prozesses AHTOUT ohne Schleifen, rechts eine Vergrößerung des oberen Teils

26 zeigt die FSM des Prozesses AHTOUT_Transmit ohne Schleifen.

10.1.1 Synthese-Daten

Aus den Scheduling-Reports für die Lösung ohne Schleifen im VHDL Quellcode (siehe Anhang) ergeben sich folgende Zahlen:

Rechenzeit für das Scheduling:
(Sun Ultra 2 mit 640 MB RAM): 86 Sekunden.

Zuverlässigkeit:
Die Synthese wurde **fehlerlos** abgeschlossen.
Das Ergebnis der Synthese ist **akzeptabel** und **korrekt simulierbar**.
Zuverlässigkeitsmaß: 10.

Geschätzte Verzögerung: (Cumulative delay) 0 ns.

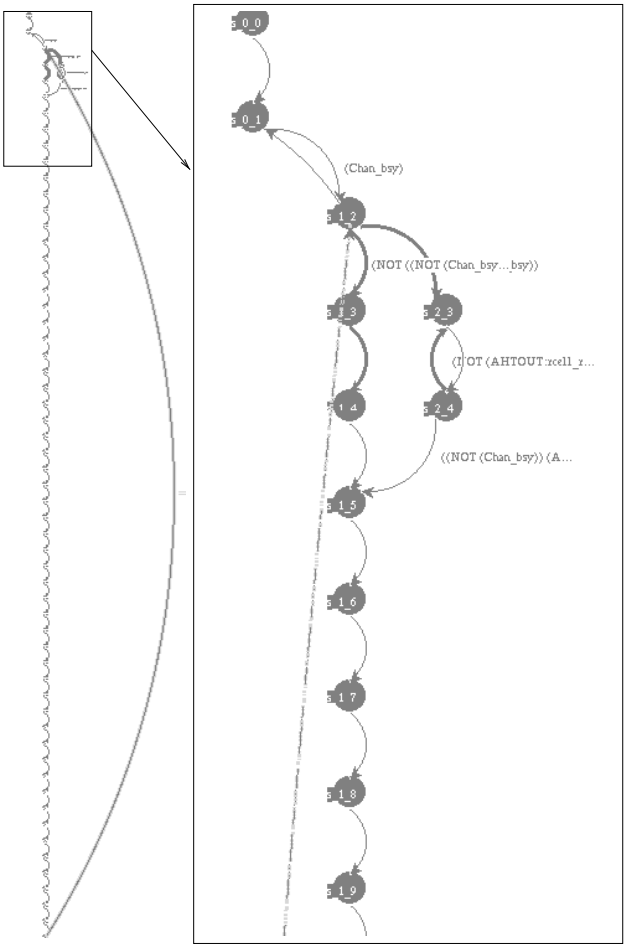


Abbildung 26: FSM des Prozesses AHTOUT_Transmit ohne Schleifen, rechts eine Vergrößerung des oberen Teils der FSM

Verzögerung 0 ns ist darauf zurückzuführen, daß keine Komponenten mit arithmetischen Operationen verwendet werden. Für das Laden der Register werden keine Verzögerungen angegeben.

Daten der synthetisierten Schaltkreise: (RT-Ebene):

1. **Prozeß AHTOUT**

   (a) **Anzahl Kontrollzyklen:** 57. (Bei einer Taktperiode von 25 ns).

   (b) **Geschätzte Schaltungsfläche (DP):**

   - Combinational area 83
   - Sequential area 1850
   - Gesamt 1933

   (c) **Komponenten:**

- 2 Register + 9 I/O Ports

(d) **Anzahl der Operationen** nach SYNOPSYS-Definition: 158

(e) **Die FSM:**

- Anzahl der Zustände: 58
- Anzahl der Zustandsübergänge: (transitions) 72.
- Steuereingänge: (control inputs) 7
- Steuerausgänge: (control outputs) 42

2. **Prozeß AHTOUT_Transmit**

(a) **Anzahl Kontrollzyklen:** 57. (Bei einer Taktperiode von 25 ns, simuliert mit Taktperiode 25 ns).

(b) **Geschätzte Schaltungsfläche (DP):**

- Combinational area 213
- Sequential area 2330
- Gesamt 2543

(c) **Komponenten:**

- 3 Register + 8 I/O Ports.

(d) **Anzahl der Operationen** nach SYNOPSYS-Definition: 112

(e) **Die FSM:**

- Anzahl der Zustände: 60
- Anzahl der Zustandsübergänge: (transitions) 63.
- Steuereingänge: (control inputs) 3
- Steuerausgänge: (control outputs) 56

11 Zusammenfassung

In der vorliegenden Arbeit wird eine ATM-Switch-Steuerung (ASS) (siehe [LaRo97]) die in VHDL verhaltensbasiert beschrieben ist, mit dem Behavioral Compiler von Synopsys synthetisiert. Die Ergebnis-Netzlisten werden simuliert. Von der Beschreibung und der Synthese ausgenommen sind die FIFO-Speicher und die Routing-Tabelle (RT).

Dabei zeigt sich, daß alle in der mit VHDL beschriebenen und simulierten Spezifikationen (nach Einfügen von Änderungen) synthetisiert werden können.

Achtet man bei den Verhaltensbeschreibungen darauf, daß nur die synthetisierbare Untermenge der sehr umfangreichen Hardwarebeschreibungssprache VHDL verwendet wird, so halten sich die nötigen Änderungen, (die auch abhängig vom Synthese-Werkzeug sind) im Rahmen.

Mit den Syntheseergebissen kann ein Prototyp der ATM-Switch-Steuerung erstellt werden. Selbst bei den Eingangs- und Ausgangsmodulen AHT_IN und AHT_OUT ist erreicht worden, daß die Eingangsdaten synchron mit einem Synchronisiersignal aufgenommen und die Ausgangsdaten kontinuierlich weitergeleitet werden.

Vergleicht man die Simulation der Verhaltensbeschreibung eines Moduls mit der Simulation nach der Synthese, so treten bei den meisten Modulen Verzögerungen durch die Synthese auf, die bei der RTL- Synthese wahrscheinlich geringer ausfallen würden.

11.1 Verwendete Bibliothek und Parametereinstellungen

Abbildung 1 zeigt die verwendete Bibliothek und Parametereinstellungen für Synopsys BC.

Bibliothek	Scheduling effort	I/O-Mode	Allocation-Modus	Takt-periode	Reset
lsi_10k.db	medium	superstate	fastest	25 ns	synchron

Tabelle 1: Verwendetet Bibliothek und Parametereinstellungen

Die **BC-Bibliothek** ist eine LCA- (Logic Cell Array) Bibliothek bei der die Basis-Zelle mit der Flächeneinheit 1 durch den Ein-Bit-Inverter dargestellt wird.

Der **Scheduling-Effort** kann beim BC in 4 Stufen eingestellt werden: (0, low, medium, high). Hier wird “medium” gewählt.

Der **Allocation-Modus** gibt die Ansiedlung der Schaltung im Entwicklungsraum an: In unserem Fall hat bei der Allocation eine Komponente mit geringer Verzögerung Vorrang vor der Komponente mit geringer Fläche.

Beim BC wird dieser Modus als Parameter für den Schritt “bc_time_design” angegeben: “bc_time_design -fastest”.

Die **Taktperiode** wird bei allen Modulen mit 25 ns gewählt.

Als **Reset** oder Zurücksetzungsmodus wird das globale synchrone Zurücksetzen gewählt.

11.2 Ergebnisse der Synthese

11.2.1 Fläche, Verzögerung, FSM

Die Tabelle Abbildung 27 zeigt die Ergebnisse der Synthese der einzelnen Module und Prozesse in Bezug auf nc-loc's (non-commentary lines of code), Chipfläche (Area), Verzögerung (delay), Anzahl Kontrollschritte (#csteps), Anzahl Operationen (#ops) und FSM-Zustände.

Behavioral VHDL Description				Behavioral Compiler					
Modul	Process	nc-loc	min csteps	nc-loc	Area (DP)	Delay (ns)	#csteps	#ops	FSM States
AHT_IN	AHTIN	41	9	59	1070	6.9	13	29	17
	Payload	30	7	35	1659		8	26	8
HT	Htproc	56	7	61	1856	0	17	44	17
RC	RCIN	94	15	121	2618	9.4	37	95	55
	RCOUT	22	6	26	742		16	47	17
SRC	Shiftproc	123	15	132	2773	9.1	33	109	42
CC	CCin	152	13	199	13054	10.4	16	80	30
RD	RDIN	158	9	165	3429	6.9	22	64	45
	RDOUT	20	6	22	2976		11	32	18
AHT_OUT	AHTOUT	35	10	140	1933	0	57	158	58
	AHTOUT_T	35	15	163	2543		57	112	60

Abbildung 27: Zusammenfassung der Syntheseergebnisse

Im ersten Block der Tabelle **"Behavioral VHDL Description"** werden die einzelnen Module und die dazugehörigen Prozesse aufgeführt. Die Spalte **"nc-loc"** (non-commentary lines of code) gibt die effektive Anzahl der VHDL-Quellcode-Zeilen, ohne Kommentar- und Leerzeilen für die simulierbare Verhaltensbeschreibung an. Die Spalte **"min csteps"** gibt die theoretisch kleinste Anzahl Kontrollschritte des Moduls an, bedingt durch die eingefügten "Wait"-Anweisungen.

Im zweiten Block **"Behavioral Compiler"** stehen die Ergebnisse der Synthese mit dem Behavioral Compiler (BC) von Synopsys.

Die **"nc-loc"** Werte sind höher als die der VHDL-Beschreibung, bedingt durch zusätzliche Attribute, Wait-Anweisungen, und folgende funktionale Änderungen:

1. Prozeß Payload: Um Verzögerungen zu vermeiden, wird die Weitergabe der Zelldaten an den Cell_FIFO-Speicher synchron anstatt asynchron durchgeführt.

2. Prozeß RCIN: Hier wird die umgekehrte Vorgehensweise eingeführt: Um den Zellkopf sicher vom Head_FIFO zu laden, wird die “Two way handshaking” Kommunikation statt der synchronen Übertragung angewendet.

3. Prozeß Shift_Proc: Das Schieberegister wird ausgelagert. Es kann als externe Komponente auf Logikebene in VHDL geschrieben und zugefügt werden.

4. Prozeß RDIN: Auch auf der Datenannahmeseite des Switches wird das Schieberegister ausgelagert.

Im Modul (AHT_OUT) ist die Erhöhung der nc-loc erheblich. Dies ist darauf zurückzuführen, daß die Schleifen von Hand aneinandergereiht werden, damit eine kontinuierliche Datenübertragung gewährleistet ist.

Bei der Chipfläche für den Datenpfad **(Area (DP))** ist die Flächeneinheit 1 die Fläche einer LCA-Basiszelle (eines 1-Bit-Inverters). Die Chipfläche wird vom BC nach dem Allocation-Schritt abgeschätzt und im “Scheduling-Report” des Moduls aufgeführt. Sie ist nicht optimiert. Eine Optimierung der Chipfläche kann bei Synopsys während der RT-Synthese mit dem Design Compiler (DC) durchgeführt werden. Die Chipfläche der FSM ist im “Scheduling-Report” des BC nicht angegeben.

Als Verzögerung **(Delay)** wird die größte Verzögerung (Cumulative Delay) aus dem “Timing-Report” genommen. Diese Verzögerung wird verursacht durch die “langsamste” Operation plus eventuell sequentielle Logik in dem entsprechenden Prozeß. Wichtig ist, daß diese Verzögerung kleiner ist als die Taktperiode (25 ns), sonst fügt der Scheduler einen Kontrollschritt hinzu.

Bei den Modulen “HT” und “AHT_OUT” sind die Verzögerungen null, da lediglich Register-Umladungen stattfinden für die der BC keine Verzögerungen anrechnet.

Die Anzahl Kontrollschritte **(#csteps)** sind beim BC durchweg höher als in der VHDL-Beschreibung, da zusätzliche Kontrollschritte eingefügt werden, um den BC-Regeln des “superstate fixed” I/O-Modus zu genügen.

Für die **(FSM)** sind als wichtigste Charakteristik die Zustände (states) aufgeführt. Die Chipfläche der FSM wird in den Synthese-Berichten des BC nicht angegeben.

11.2.2 Rechenzeit, Speicherbedarf und Zuverlässigkeit

Die Tabelle Abbildung 28 zeigt die Ergebnisse der Synthese der einzelnen Module und Prozesse in Bezug auf Rechenzeit, Speicherbedarf und Zuverlässigkeit.

Als **Rechenzeit** wird die Zeit für den “Scheduling”-Schritt aus dem “Scheduling-Report” angegeben.

Der **Speicherbedarf** wird für den BC dem “Scheduling-Report” entnommen. Aus der Tabelle ist ersichtlich, daß die Module SRC, CC, RD und AHT_OUT einen Rechner benötigen, der mit einem Arbeitsspeicher größer als 64 MB ausgerüstet ist.

Die **Zuverlässigkeit** ist in Kapitel 3 definiert. Das Zuverlässigkeitsmaß Z gibt an, in welchem Maß das Ergebnis der Synthese korrekt und akzeptabel ist.

	Behavioral Compiler		
Modul	Rechen-zeit (sec)	Speicher-bedarf (MB)	Zuver-lässigkeit (Z)
AHT_IN	14	8.9	10
HT	9	8.5	10
RC	55	22.6	10
SRC	98	19.1	10
CC	873	118.1	10
RD	7258	603.6	5
AHT_OUT	86	21.3	10

Abbildung 28: Zusammenfassung der Syntheseergebnisse in Bezug auf Rechenzeit, Speicherbedarf und Zuverlässigkeit

11.2.3 Dokumentation und Einarbeitungszeit

Die **Dokumentation** für den BC [Syn97] [SynUg97] ist sehr umfangreich und sehr detailliert. Das ist einerseits positiv, aber andererseits bedeutet das auch längeres Suchen, um eine bestimmte Auskunft zu finden.

Die **Einarbeitszeit** für den BC ist relativ lang. Der Benutzer sollte wenigstens den "BC User's Guide" lesen und dazu die "BC Methodology Guidlines". Zusätzlich ist noch vorausgesetzt, daß der Benutzer über den Synopsys "Design Compiler" Bescheid weiß. Trotzdem wird er noch in einigen weiteren (online) Handbüchern nachschlagen müssen um alle Informationen zu erhalten. (Beispiel: "fastest" Allocation ist eine Option des Befehls "bc_time_design". Das heißt: im Kapitel "BC Commands" nachschlagen).

12 Anhang: Quellcodes und Berichte (reports) der High-Level-Synthese

12.1 Eingangsmodul AHT_IN

Quellcode der VHDL-Verhaltensbeschreibung

```
-- Verhaltensbeschreibung des AHT_In fuer
-- Synthese mit Synopsys - BC
-- ahtin_bc.vhd
-- AHT_In hat 2 Prozesse: einen fuer die Aufnahme und asynchrone Weiterleitung
-- des Zell-Headers, der zweite Prozess nimmt die Payload auf,  und die
-- asynchrone Weitergabe des Payload Registers an den Cell-Fifo.
-- Version mit Synchronisation fuer Cell_Sync_In (start_loop)
-- Aenderungen fuer Synopsys:
-- design_reset eingefuehrt
-- Attribute fuer "dont_unroll" of load_loops
-- Das handshaking zum Payload FIFO geht hier so nicht. Synopsys BC bringt
-- Fehler. Man sollte synchron
-- zum Payload FIFO uebertragen. (Siehe Payload-Prozess)
-- Aenderung in Gliwice am 8.7.98: Im Payload Process: Warten auf
-- Start_Payl durch ein loop ersetzt. Vorher gab es eine Verschiebung bei der
-- Uebertragung jeder 2. Zelle.
-- Autor: W. Lange Januar 1998

Library synopsys;
use synopsys.attributes.all;
library dware;
-- use dware.behavioral.all;
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

ENTITY aht_in IS
  PORT (Data_In     : IN UNSIGNED(7 DOWNTO 0);
        Clk_AHT_In  : IN STD_LOGIC;
        Cell_Sync_In : IN STD_LOGIC;
        Reset       : IN STD_LOGIC;   -- fuer Synopsys eingefuehrt
        Hd_fetch    : IN STD_LOGIC;
        Cell_fetch  : IN STD_LOGIC;
        Buffer_full  : IN BOOLEAN;
        Hd_Bus      : OUT UNSIGNED(31 DOWNTO 0);
        Cell_Bus    : OUT UNSIGNED(31 DOWNTO 0);
        Hd_rdy      : OUT STD_LOGIC;
        Cell_rdy    : OUT STD_LOGIC);
END aht_in;
---------------------------------

ARCHITECTURE aht1ar OF aht_in IS
 SIGNAL Start_Payl:  STD_LOGIC;

BEGIN
  -- Der AHTIN Process nimmt den Header auf und leitet ihn asynchron
  -- an HT weiter.

  AHTIN: PROCESS
   VARIABLE Hd_reg     : UNSIGNED(31 downto 0);  -- Reg fuer Header B.

  BEGIN -- Process

  --  Reset: Setze alle Variablen und Indices zurueck ----------------------
    Hd_rdy <= '0';
    Start_Payl <= '0';   -- fuer BC zurueckgenommen
    Hd_reg := "00000000000000000000000000000000";
    Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';

  -- Cell_Sync_In erscheint, eine neue Zelle kommt an...

  main_loop: WHILE  NOT Buffer_full  loop   -- main loop
        Start_Payl <= '0';
    Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';  -- fuer BC

    start_loop: loop     -- fastest handshaking
      Hd_reg(31 downto 24) := Data_In;
      IF  (Cell_Sync_In = '1') THEN
        Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
        exit start_loop;
      else
        Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
      END IF;
    End loop;

    -- Cell_Sync ist da, lade den Header..     ------------------------------

Hd_reg(23 downto 16) := Data_In;
Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
Hd_reg(15 downto 8)  := Data_In;
```

```
Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
       Start_Payl <= '1';   -- 2  Kontrollschritte vorverlegt fuer BC
Hd_reg(7 downto 0)   := Data_In;
Hd_Bus <= Hd_reg;    -- Setze das Hdreg. auf den Bus und
       Hd_rdy <= '1';       -- Vergiss den HEC...
Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
       -- Das HEC Byte bleibt unberuecksichtigt.
-- Start_Payl <= '1';    -- fuer BC
Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1' and Hd_fetch = '1';
       Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';  -- fuer BC
Hd_rdy <= '0';
Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
    End loop; -- Main Loop
    Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1'; -- for BC
  End Process;


  Payload: PROCESS
  VARIABLE Cell_reg    : UNSIGNED(31 downto 0);  -- Reg fuer Cell Bytes
  attribute dont_unroll : boolean;
  attribute dont_unroll of load_loop2 : label is true;

  -- Der Payload Process nimmt den Header auf und leitet ihn asynchron
  -- an den Cell-Fifo weiter.

  BEGIN -- Process
  --  Reset: Setze alle Variablen und Indices zurueck ----------------------
      Cell_rdy  <= '0';
      Cell_reg := "00000000000000000000000000000000";
  Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
  -- Die Payload wird geladen

  main_loop2: loop               -- main loop
  -- main_loop2: While not Buffer_full loop
  --  Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1'; -- fuer BC

  -- Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1' and Start_Payl = '1';
    stpayl_loop: loop     -- fastest handshaking
     IF  (Start_Payl = '1') THEN
        Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
        exit stpayl_loop;
      else
        Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
      END IF;
    End loop;

  load_loop2: For k in 1 TO 12 loop
   --  4 Byte der Zelle werden geladen..

      -- Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1'; -- fuer BC
      -- IF Cell_fetch = '1' THEN Cell_rdy <= '0';  END IF;
      -- Das handshaking geht hier so nicht. Man sollte synchron
      -- zum Payload FIFO uebertragen.
      -- Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1'; -- fuer BC

Cell_reg(31 downto 24) := Data_In;
          Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
Cell_reg(23 downto 16) := Data_In;
 Cell_rdy  <= '0';
          Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
Cell_reg(15 downto 8)  := Data_In;
          Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
Cell_reg(7 downto 0)   := Data_In;
            Cell_Bus <= Cell_reg;   -- Lade 4 Bytes
            Cell_rdy <= '1';
          Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1';

   End loop;   -- loop2: Payload loop   -------------------------
             Cell_rdy <= '0';
  Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1'; -- for BC
  END loop; -- main loop
  Wait UNTIL Clk_AHT_In'EVENT AND Clk_AHT_In = '1'; -- for BC
  END Process;


 END aht1ar;
```

Quellcode des AHT_IN-Testtreibers

```
-- Testdriver testaht11.vhd  fuer aht11.vhd, aht12.vhd und aht14.vhd
-- Simulation mit Synopsys Simulater V. 1997.08
-- Autor: W. Lange. August 1997

library IEEE;
library synopsys;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_unsigned.all;
   use IEEE.std_logic_misc.all;
  use IEEE.std_logic_arith.all;
--  use IEEE.std_logic_components.all;
  use synopsys.attributes.all;
```

```
  use STD.Textio.all;

Entity testaht IS END;

-- USE  work.utilsAHT.all;

ARCHITECTURE test_driver OF testaht IS

  procedure Clock (signal C: out STD_LOGIC; HT, LT: Time) is
  begin
    loop
    C <= '0' , '1' AFTER LT ;
    Wait for (LT + HT);
    end loop;
  end;

COMPONENT aht_in
  PORT  (Data_In     : IN UNSIGNED(7 DOWNTO 0);
       Clk_AHT_In   : IN STD_LOGIC;
       Cell_Sync_In : IN STD_LOGIC;
       Reset        : IN STD_LOGIC;
       Hd_fetch     : IN STD_LOGIC;
       Cell_fetch   : IN STD_LOGIC;
       Buffer_full  : IN BOOLEAN;
       Hd_Bus       : OUT UNSIGNED(31 DOWNTO 0);
       Cell_Bus     : OUT UNSIGNED(31 DOWNTO 0);
       Hd_rdy       : OUT STD_LOGIC;
       Cell_rdy     : OUT STD_LOGIC);
END COMPONENT;

FOR ALL : aht_in  USE entity work.aht_in(SYN_aht1ar); --Fuer RTL Beschreibung
--FOR ALL : aht_in  USE entity work.aht_in(aht1ar);  -- Fuer Beh.  Description

  SIGNAL Cell_Sync_In,Hd_fetch,GSR,Clk_Aht_In : STD_LOGIC;
  SIGNAL  Hd_rdy, Cell_rdy,Cell_fetch,Reset   : STD_LOGIC;
  SIGNAL Hd_Bus    : UNSIGNED(31 DOWNTO 0);
  SIGNAL Data_In   : UNSIGNED(7 DOWNTO 0);
  SIGNAL Cell_Bus  : UNSIGNED(31 DOWNTO 0);
  SIGNAL Buffer_full      : BOOLEAN;

BEGIN

 UUT: aht_in port map (Data_In,Clk_AHT_In,Cell_Sync_In,Reset,Hd_fetch,
   Cell_fetch,Buffer_full, Hd_Bus, Cell_Bus,Hd_rdy,Cell_rdy);

 CLK: CLOCK(Clk_AHT_In, 10 ns,10 ns); -- oben declariert oder im package.

 Stimulus: Process
 VARIABLE hilf : UNSIGNED(7 downto 0);

 BEGIN
 -- Clk_com sollte mit der Prozedur Clock loslaufen....

 -- Reset first..
     GSR <= '0';          -- Global reset
     Reset <= '0';
     Cell_fetch <= '0';
     Buffer_full <= false;
     Hd_fetch <= '0';
   --  Clk_AHT_In <= '0';
     hilf := "00000000";
     Cell_Sync_In <= '0';

     WAIT FOR 30 ns;        -- Warte ein wenig
     GSR <= '1';
     Reset <= '1';
     WAIT FOR 40 ns;
     GSR <= '0';
     Reset <= '0';
     WAIT FOR 20 ns;
     FOR i in 1 to 2 loop
 Wait until Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
     end loop;



 WHILE TRUE loop
 --      Data_In <=  "00000001";
     Wait until Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
     Cell_Sync_In <= '1';
 -- Bringe erst den Header (komplett)
    FOR i in 1 to 4 loop
        hilf := hilf + "00000001";
        Data_In <= hilf;
    Wait until Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
    end loop;

 -- Hd_rdy kommt von AHT_IN, Hd_fetch vom HT
    Cell_Sync_In <= '0';

 -- Drop one byte (hec Byte)
    hilf := hilf + "00000001";
    Wait until Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
```

```
 -- Jetzt kommt die Zelle
    FOR i in 1 to 48 loop
        hilf := hilf + "00000001";
        Data_In <= hilf;
 IF Hd_rdy = '1' THEN Hd_fetch <= '1'; END IF;
         IF Hd_rdy = '0'  THEN Hd_fetch <= '0'; END IF;
         IF Cell_rdy = '1' THEN Cell_fetch <= '1';  END IF;
         IF Cell_rdy = '0'  THEN Cell_fetch <= '0';  END IF;
    Wait until Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
    end loop;

--         IF Cell_rdy = '0'  THEN Cell_fetch <= '0';  END IF;
--    Wait until Clk_AHT_In'EVENT AND Clk_AHT_In = '1';

    FOR i in 1 to 4 loop
         IF Cell_rdy = '1' THEN Cell_fetch <= '1';  END IF;
         IF Cell_rdy = '0'  THEN Cell_fetch <= '0';  END IF;
    Wait until Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
   End loop;

     Wait until Clk_AHT_In'EVENT AND Clk_AHT_In = '1';
  END loop; -- main loop;
  END Process;
END;
```

Das Synthese-Script

```
/* bc_script fuer no loops (nl) - ahtin_nl_...  */

bc_enable_analysis_info =  "true" /* For BC_View */

analyze -f vhdl ahtin_bc.vhd

elaborate -s aht_in
/* reset- Setzen NACH elaborate ! */
set_behavioral_reset Reset -active high
register_control -process AHTIN -inputs
create_clock Clk_AHT_In  -period 25
write -h -o ahtin_nl_elab.db   /* save the elaborated design */

bc_check_design -io superstate_fixed
/* set_operating_conditions wccom */

bc_time_design -force -fastest > ahtin_time_bc.rep
write -h -o ahtin_bc_timed.db    /*  save the timed design */

schedule -io superstate_fixed -effort medium   > ahtin_bc_sched.rep

/* rename_design ahtin_scheduled */ /* keeps the names clean */
report_schedule -abstract > ahtin_bc_fsm.rep
/* report_schedule -operations > ahtin__sched_op.rep */
report_schedule -summary > ahtin_bc_sched_sum.rep

/* write -h -o ahtin_cf.db */       /* save the RTL design to a db file */
vhdlout_dont_write_types = "true"     /* prevents clashes in elaboration */
vhdlout_use_packages = { ieee.std_logic_1164,ieee.std_logic_arith }
vhdlout_equations = "true"      /* ignore gate delays */
vhdlout_levelize = true

write -hier -f vhdl -out ahtin_bc_reg.vhd
                         /* saves a simulatable register_level design */
quit
```

BC-Scheduling-Reports

```
Warning: Design has already been timed. (HLS-117)
Loading db file '/afs/wsi/ti/synopsys/1997.8/libraries/syn/gtech.db'
Information: Scheduling 'loop_90_design' ... (HLS-152)
Information: Scheduling 'start_loop_design' ... (HLS-152)
Information: Scheduling 'main_loop_design' ... (HLS-152)
Information: Scheduling 'AHTIN_design' ... (HLS-152)
Information: Allocating hardware for 'AHTIN_design' ... (HLS-153)
****************************************************************************
               Date      : Wed Sep 30 15:47:00 1998
               Version   : 1997.08
               Design    : aht_in
****************************************************************************

****************************************
*  Operation schedule of process AHTIN:  *
****************************************

Resource types
==================================
    loop......loop boundaries
    p0........1-bit registered output port Hd_rdy
    p1........1-bit registered output port Start_Payl
```

```
    p2........32-bit registered output port Hd_Bus
    p3........1-bit input port Buffer_full
    p4........8-bit input port Data_In
    p5........1-bit input port Hd_fetch
    p6........1-bit input port Cell_Sync_In


                  p     p     p     p     p     p     p
                  o     o     o     o     o     o     o
                  r     r     r     r     r     r     r
                  t     t     t     t     t     t     t
-------+------+-----+-----+-----+-----+-----+-----+-----
 cycle | loop | p3  | p4  | p5  | p6  | p0  | p1  | p2
-------------------------------------------------------
   0   |..L0..|.....|.....|.....|.....|.W55.|.W56.|.....
   1   |..L3..|.R62.|.....|.....|.....|.....|.....|.....
   2   |..L6..|.....|.....|.....|.....|.....|.W63.|.....
   3   |..L9..|.....|.R67.|.....|.R68.|.....|.....|.....
   4   |.L14..|.....|.R78.|.....|.....|.....|.....|.....
       |.L13..|.....|.....|.....|.....|.....|.....|.....
       |.L10..|.....|.....|.....|.....|.....|.....|.....
   5   |......|.....|.R80.|.....|.....|.....|.....|.....
   6   |......|.....|.R83.|.....|.....|.W85.|.W82.|.W84.
   7   |......|.....|.....|.....|.....|.....|.....|.....
   8   |..L7..|.....|.....|.....|.....|.....|.....|.....
   9   |......|.....|.....|.R90.|.....|.....|.....|.....
  10   |.L12..|.....|.....|.....|.....|.....|.....|.....
       |.L11..|.....|.....|.....|.....|.....|.....|.....
       |..L8..|.....|.....|.....|.....|.....|.....|.....
  11   |......|.....|.....|.....|.....|.W92.|.....|.....
  12   |..L5..|.....|.....|.....|.....|.....|.....|.....
       |..L4..|.....|.....|.....|.....|.....|.....|.....
  13   |..L2..|.....|.....|.....|.....|.....|.....|.....
       |..L1..|.....|.....|.....|.....|.....|.....|.....


Operation name abbreviations
============================
    L0.......loop boundaries AHTIN_design_loop_begin
    L1.......loop boundaries AHTIN_design_loop_end
    L2.......loop boundaries AHTIN_design_loop_cont
    L3.......loop boundaries main_loop/main_loop_design_loop_begin
    L4.......loop boundaries main_loop/main_loop_design_loop_end
    L5.......loop boundaries main_loop/main_loop_design_loop_cont
    L6.......loop boundaries main_loop/EXIT_L62
    L7.......loop boundaries main_loop/loop_90/loop_90_design_loop_begin
    L8.......loop boundaries main_loop/loop_90/loop_90_design_loop_end
    L9.......loop boundaries main_loop/start_loop/start_loop_design_loop_begin
    L10......loop boundaries main_loop/start_loop/start_loop_design_loop_end
    L11......loop boundaries main_loop/loop_90/loop_90_design_loop_cont
    L12......loop boundaries main_loop/loop_90/EXIT_L90
    L13......loop boundaries main_loop/start_loop/start_loop_design_loop_cont
    L14......loop boundaries main_loop/start_loop/EXIT_L70
    R62......1-bit read main_loop/Buffer_full_62
    R67......8-bit read main_loop/start_loop/Data_In_67
    R68......1-bit read main_loop/start_loop/Cell_Sync_In_68
    R78......8-bit read main_loop/Data_In_78
    R80......8-bit read main_loop/Data_In_80
    R83......8-bit read main_loop/Data_In_83
    R90......1-bit read main_loop/loop_90/Hd_fetch_90
    W55......1-bit write Hd_rdy_55
    W56......1-bit write Start_Payl_56
    W63......1-bit write main_loop/Start_Payl_63
    W82......1-bit write main_loop/Start_Payl_82
    W84......32-bit write main_loop/Hd_Bus_84
    W85......1-bit write main_loop/Hd_rdy_85
    W92......1-bit write main_loop/Hd_rdy_92


*************************************************************************
Information: registering status inputs of control FSM
Control FSM has 17 states
One-Hot coding style selected, state vector is 17 bits long.
State  Code
s_0_0  1----------------
s_0_1  -1---------------
s_0_13  --1--------------
s_1_2  ---1-------------
s_1_3  ----1------------
s_1_4  -----1-----------
s_1_5  ------1----------
s_1_6  -------1---------
s_1_7  --------1--------
s_1_8  ---------1-------
s_1_9  ----------1------
s_1_10  -----------1-----
s_1_11  ------------1----
s_1_12  -------------1---
s_2_4  --------------1--
s_3_9  ---------------1-
s_3_10  ----------------1
```

```
Control unit for process AHTIN of design aht_in is hardwired
Control unit for process AHTIN synthesized.
Information: Scheduling 'stpayl_loop_design' ... (HLS-152)
Information: Scheduling 'load_loop2_design' ... (HLS-152)
Information: Scheduling 'main_loop2_design' ... (HLS-152)
Information: Scheduling 'Payload_design' ... (HLS-152)
Information: Allocating hardware for 'Payload_design' ... (HLS-153)
******************************************************************************
               Date      : Wed Sep 30 15:47:07 1998
               Version   : 1997.08
               Design    : aht_in
******************************************************************************

*****************************************
*  Operation schedule of process Payload:  *
*****************************************

Resource types
====================================
    loop......loop boundaries
    p0........1-bit registered output port Cell_rdy
    p1........32-bit registered output port Cell_Bus
    p2........8-bit input port Data_In
    p3........1-bit input port Start_Payl
    r770......(4_4->1)-bit DW01_cmp2
    r779......(4->4)-bit DW01_inc



                                D
                                W      D
                                0      W
                                1      0
                                _      1
                   p      p     c      _      p      p
                   o      o     m      i      o      o
                   r      r     p      n      r      r
                   t      t     2      c      t      t
-------+------+------+------+------+------+------+------
 cycle | loop |  p2  |  p3  | r770 | r779 |  p0  |  p1
-----------------------------------------------------------
   0   |..L0..|......|......|......|......|.W109.|......
   1   |..L8..|......|.R120.|......|......|......|......
       |..L3..|......|......|......|......|......|......
   2   |.L13..|.R137.|......|.o583.|......|......|......
       |.L12..|......|......|......|......|......|......
       |..L9..|......|......|......|......|......|......
       |..L6..|......|......|......|......|......|......
   3   |.L11..|.R139.|......|......|......|.W140.|......
   4   |......|.R142.|......|......|......|......|......
   5   |......|.R144.|......|......|.o843.|.W146.|.W145.
   6   |.L10..|......|......|......|......|.W150.|......
       |..L7..|......|......|......|......|......|......
   7   |..L5..|......|......|......|......|......|......
       |..L4..|......|......|......|......|......|......
   8   |..L2..|......|......|......|......|......|......
       |..L1..|......|......|......|......|......|......


Operation name abbreviations
============================
    L0........loop boundaries Payload_design_loop_begin
    L1........loop boundaries Payload_design_loop_end
    L2........loop boundaries Payload_design_loop_cont
    L3........loop boundaries main_loop2/main_loop2_design_loop_begin
    L4........loop boundaries main_loop2/main_loop2_design_loop_end
    L5........loop boundaries main_loop2/main_loop2_design_loop_cont
    L6........loop boundaries main_loop2/load_loop2/load_loop2_design_loop_begin
    L7........loop boundaries main_loop2/load_loop2/load_loop2_design_loop_end
    L8........loop boundaries main_loop2/stpayl_loop/stpayl_loop_design_loop_begin
    L9........loop boundaries main_loop2/stpayl_loop/stpayl_loop_design_loop_end
    L10.......loop boundaries main_loop2/load_loop2/load_loop2_design_loop_cont
    L11.......loop boundaries main_loop2/load_loop2/EXIT_L128
    L12.......loop boundaries main_loop2/stpayl_loop/stpayl_loop_design_loop_cont
    L13.......loop boundaries main_loop2/stpayl_loop/EXIT_L122
    R120......1-bit read main_loop2/stpayl_loop/Start_Payl_120
    R137......8-bit read main_loop2/load_loop2/Data_In_137
    R139......8-bit read main_loop2/load_loop2/Data_In_139
    R142......8-bit read main_loop2/load_loop2/Data_In_142
    R144......8-bit read main_loop2/load_loop2/Data_In_144
    W109......1-bit write Cell_rdy_109
    W140......1-bit write main_loop2/load_loop2/Cell_rdy_140
    W145......32-bit write main_loop2/load_loop2/Cell_Bus_145
    W146......1-bit write main_loop2/load_loop2/Cell_rdy_146
    W150......1-bit write main_loop2/Cell_rdy_150
    o583......(4_4->1)-bit LEQ_UNS_OP main_loop2/load_loop2/lte_128
    o843......(4_1->4)-bit ADD_UNS_OP main_loop2/load_loop2/add_128


******************************************************************************
Information: control FSM status and output not registered
Control FSM has 8 states
One-Hot coding style selected, state vector is 8 bits long.
```

```
State  Code
s_0_0  1-------
s_0_1  -1------
s_1_7  --1-----
s_2_2  ---1----
s_3_3  ----1---
s_3_4  -----1--
s_3_5  ------1-
s_3_6  -------1

Control unit for process Payload of design aht_in is hardwired
Control unit for process Payload synthesized.

Memory usage during scheduling 8969 Kbytes.
CPU usage during scheduling was 14 seconds.
Scheduled design aht_in.
Warning: Overwriting design file '/afs/informatik.uni-tuebingen.de/home/wlange/hilan/aht/ahtin_bc/aht_in.db'. (DDB-24)
Current design is 'aht_in'.


********************************************************************************
               Date      : Wed Sep 30 15:47:11 1998
               Version   : 1997.08
               Design    : aht_in
********************************************************************************

************************************
*  Summary report for process AHTIN:  *
************************************
--------------------------------------------------------------------------------
     Timing Summary
--------------------------------------------------------------------------------
 Clock period 25.00
 Loop timing information:
    AHTIN.....................................13 cycles (cycles 0 - 13)
        main_loop.............................11 cycles (cycles 1 - 12)
            (exit) EXIT_L62.............................. (cycle 2)
            start_loop....................1 cycle  (cycles 3 - 4)
                (exit) EXIT_L70.......................... (cycle 4)
            loop_90.............................2 cycles (cycles 8 - 10)
                (exit) EXIT_L90.......................... (cycle 10)
--------------------------------------------------------------------------------
     Area Summary
--------------------------------------------------------------------------------
 Estimated combinational area    40
 Estimated sequential area       1030
 TOTAL                           1070

 17 control states
 22 basic transitions
 4 control inputs
 12 control outputs

--------------------------------------------------------------------------------
     Resource types
--------------------------------------------------------------------------------
Register Types
=======================================
1-bit register.....................3
32-bit register....................1

Operator Types
=======================================

I/O Ports
=======================================
1-bit input port...................3
1-bit registered output port.......2
8-bit input port...................1
32-bit registered output port......1

--------------------------------------------------------------------------------
****************************************
*  Summary report for process Payload:  *
****************************************
--------------------------------------------------------------------------------
     Timing Summary
--------------------------------------------------------------------------------
 Clock period 25.00
 Loop timing information:
    Payload...........................................8 cycles (cycles 0 - 8)
        main_loop2.................................6 cycles (cycles 1 - 7)
            stpayl_loop...........................1 cycle  (cycles 1 - 2)
                (exit) EXIT_L122.......................... (cycle 2)
            load_loop2............................4 cycles (cycles 2 - 6)
                (exit) EXIT_L128.......................... (cycle 3)
--------------------------------------------------------------------------------
     Area Summary
--------------------------------------------------------------------------------
 Estimated combinational area    99
 Estimated sequential area       1560
 TOTAL                           1659

 8 control states
```

```
 10 basic transitions
 4 control inputs
 15 control outputs

------------------------------------------------------------------------
     Resource types
------------------------------------------------------------------------
Register Types
=======================================
1-bit register.....................2
4-bit register.....................1
32-bit register....................1

Operator Types
=======================================
(4->4)-bit DW01_inc................1
(4_4->1)-bit DW01_cmp2.............1

I/O Ports
=======================================
1-bit input port...................1
1-bit registered output port.......1
8-bit input port...................1
32-bit registered output port......1

------------------------------------------------------------------------
```

12.2 Zellkopfübersetzungsmodul HT

12.2.1 Synthese mit dem Synopsys BC^{TM}

Quellcode der VHDL-Verhaltensbeschreibung

```
-- Verhaltensbeschreibung des HT (Header Translator)  fuer Synopsys
-- Behavioral Compiler
-- ht_bc/ht_bc.vhd
-- Aenderungen gegenueber htcad5/ht.vhd:
-- Reset wird eingefuehrt.
-- Autor: W. Lange. Letzte Aenderung: 9. 2. 1998
-- 60 NCLOC's (5.5.98)

Library IEEE;
USE IEEE.std_logic_1164.all;
--  use IEEE.std_logic_unsigned.all;
--  use IEEE.std_logic_misc.all;
    use IEEE.std_logic_arith.all;
--  use IEEE.std_logic_components.all;
--  use synopsys.attributes.all;
--  use STD.Textio.all;

ENTITY ht IS
  PORT ( Clk_com     : IN  STD_LOGIC ;       -- Clock common
Hd_Bus       : IN  STD_LOGIC_Vector(31 DOWNTO 0); -- Header - 4 Bytes
Hd_rdy       : IN  STD_LOGIC;   -- AHT1 has Header for you
Hd_fetch     : OUT STD_LOGIC;   -- Header arrived in HT: Header fechted
RT_Addr      : OUT STD_LOGIC_Vector(15 downto 0);
rt_read      : OUT STD_LOGIC;
RT_data      : IN  STD_LOGIC_Vector(39 downto 0);
       rtdta_strobe : IN  STD_LOGIC;
fwrq         : OUT STD_LOGIC;                        -- Fifo write re.
  headout      : Out STD_LOGIC_Vector(47 downto 0);   -- Fifo input
       head_taken   : IN  STD_LOGIC;
ffull        : IN  STD_LOGIC;                         -- fifo ist voll
       Reset        : IN  STD_LOGIC);
END ht;
---------------------------------

ARCHITECTURE htar OF ht IS

BEGIN
 htproc : PROCESS

  VARIABLE  Address    : STD_LOGIC_Vector(0 TO 15);
  VARIABLE  Header     : STD_LOGIC_Vector(47 DOWNTO 0);  -- Header w. RTag
  VARIABLE  rtag       : STD_LOGIC_Vector(15 DOWNTO 0);  -- Hilfsvar. f.rtag
  VARIABLE  vpivci     : STD_LOGIC_Vector(23 DOWNTO 0);  -- Hilfsvar. f. vpivci
  VARIABLE  rtdata     : STD_LOGIC_Vector(39 downto 0);  -- Hilfsvar.
  VARIABLE  rdwrt      : STD_LOGIC;

 BEGIN -- Process ----------------------------------------------------

     Hd_fetch <= '0';  -- Setze die Signale auf null..
     fwrq     <= '0';
     rt_read  <= '0';
     --   Headfifo_reset;    -- Reset Header - Puffer (noch nicht eingebaut)

     WAIT UNTIL Clk_com'event and Clk_com = '1';
```

```
    ------------------------ Haupt-Loop ---------------------------------
    WHILE TRUE LOOP  --  fuer SYNOPSYS
      -- Hole einen Header vom AHT1_In ab                    -----------------
      WAIT UNTIL  Clk_com'EVENT and Clk_com = '1'and Hd_rdy = '1';
      Header(31 downto 0) := Hd_Bus;
      Hd_fetch <= '1';                  -- Sag AHT1_In: Header abgeholt!
      WAIT UNTIL Clk_com'event and Clk_com = '1' and Hd_rdy = '0';

      --  Adressiere die Routing-Tabelle VPI_2, VCI_2
      Address := Header(27 downto 24) & Header(15 downto 4); -- Get VCI/VPI
      -- Hier kann man die Adressrechnung einsetzen Addr:=F1(Addr);
      -- Adressiere die RT und hole die neuen VPI und VCI's
      WAIT UNTIL Clk_com'event and Clk_com = '1';            -- fuer BC (1)
      RT_Addr <= Address; -- Setze die Adresse
      rt_read <= '1';
      Hd_fetch <= '0';

      -- Warte auf das RT-Data Ergebnis
      WAIT UNTIL Clk_com'event and Clk_com = '1' and rtdta_strobe = '1';
      WAIT UNTIL Clk_com'event and Clk_com = '1';            -- fuer BC (2)
      rtdata := RT_data;
      rtag := rtdata(39 downto 24);      -- Hilfsvariablen wegen CADDY
      vpivci := rtdata(23 downto 0);   -- Hilfsvariablen wegen CADDY
      -- Header(31 DOWNTO 28) ist GFC, von Bit 3 bis 0: PTI und CLP
      Header(47 downto 4) := rtag & Header(31 DOWNTO 28) & vpivci;

      rt_read <= '0';
      WAIT UNTIL Clk_com'event and Clk_com = '1';            -- fuer BC (3)

     -- Speichere das Headerregister im Headerpuffer ab    ------------------
     WAIT UNTIL Clk_com'event and Clk_com = '1' and ffull = '0';
     fwrq <= '1';
     headout <= Header;
     WAIT UNTIL Clk_com'event and Clk_com = '1';           -- fuer BC (4)

     WAIT UNTIL Clk_com'event and Clk_com = '1' and head_taken = '1';
     fwrq <= '0';
     WAIT UNTIL Clk_com'event and Clk_com = '1';
   END LOOP;
   WAIT UNTIL Clk_com'event and Clk_com = '1'; -- Fuer BC
 END Process;
END;          -- End architecture
```

Quellcode des AHT_IN-Testtreibers

```
-- Testdriver fuer ht.vhd
-- ~/hilan/aht/htcad5/testht.vhd
-- Autor W. Lange
-- Letzte Aenderung 3. 12. 97

library IEEE;
library synopsys;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_unsigned.all;
   use IEEE.std_logic_misc.all;
  use IEEE.std_logic_arith.all;
--  use IEEE.std_logic_components.all;
  use synopsys.attributes.all;
  use STD.Textio.all;

Entity testht IS END;

-- USE  work.utilsAHT.all;

ARCHITECTURE test_driver OF testht IS

COMPONENT ht
  PORT ( Clk_com     : IN  STD_LOGIC ;       -- Clock common
Hd_Bus       : IN  STD_LOGIC_Vector(31 DOWNTO 0); -- Header - 4 Bytes
Hd_rdy       : IN  STD_LOGIC;   -- AHT1 has Header for you
Hd_fetch     : OUT STD_LOGIC;   -- Header arrived in HT: Header fechted
RT_Addr      : OUT STD_LOGIC_Vector(15 downto 0);
rt_read      : OUT STD_LOGIC;
RT_data      : IN  STD_LOGIC_Vector(39 downto 0);
       rtdta_strobe : IN  STD_LOGIC;
fwrq         : OUT STD_LOGIC;                      -- Fifo write re.
  headout      : Out STD_LOGIC_Vector(47 downto 0);   -- Fifo input
       head_taken   : IN  STD_LOGIC;
ffull        : IN  STD_LOGIC;     -- fifo ist voll
       Reset        : IN  STD_LOGIC);
End COMPONENT;

-- FOR ALL : ht  USE entity work.ht(SYN_htar);
FOR ALL : ht  USE entity work.ht(htar);

  SIGNAL Clk_com,Hd_rdy : STD_LOGIC := '0';
  SIGNAL Hd_Bus    : std_logic_Vector(31 DOWNTO 0);
  SIGNAL Hd_fetch,rt_read,fwrq,ffull,rtdta_strobe,head_taken : STD_LOGIC;
  SIGNAL RT_Addr   : STD_LOGIC_Vector(15 downto 0);
  SIGNAL RT_data   : STD_LOGIC_Vector(39 downto 0);
  SIGNAL headout   : STD_LOGIC_Vector(47 downto 0);
  SIGNAL GSR,Reset       : std_logic;
```

```
BEGIN

 UUT: ht port map (Clk_com, Hd_Bus, Hd_rdy, Hd_fetch, RT_Addr, rt_read,
  RT_data, rtdta_strobe, fwrq, headout, head_taken, ffull,Reset);

  Clk_com <= not Clk_com after 25 ns;

 Stimulus: Process

 BEGIN
 -- Clk_com sollte mit der Prozedur Clock loslaufen....

 -- Reset first..
     RT_data <= "00000000000000000000000000000000000000000";
     rtdta_strobe <= '0';
     head_taken <= '0';
     ffull  <= '0';
     Reset  <= '0';
     WAIT FOR 20 ns;        -- Warte ein wenig
     GSR <= '1';
     Reset <= '1';
     WAIT FOR 40 ns;
     GSR <= '0';
     Reset <= '0';
     WAIT FOR 20 ns;

  WHILE TRUE loop

     Wait until Clk_com'EVENT AND Clk_com = '1';
 ------------------------2. Takt-------------------------------------
     Hd_Bus <= "10010000110101010011000110100101"; --32 beliebige Bit
     Hd_rdy <= '1';
    Wait until Clk_com'EVENT AND Clk_com = '1' and Hd_fetch = '1';
     Hd_rdy <= '0';

 ------------------------3. Takt-------------------------------------
    Wait until Clk_com'EVENT AND Clk_com = '1' and rt_read = '1';
     rtdta_strobe <= '1';
     RT_data <= "00000000000000101010101111001100111101010";   -- 40 bit
    Wait until Clk_com'EVENT AND Clk_com = '1' and rt_read = '0';
     rtdta_strobe <= '0';

 ------------------------4. Takt  ---------------------------
    Wait until Clk_com'EVENT AND Clk_com = '1' and fwrq = '1';
     head_taken <= '1';

 ------------------------5. Takt  ---------------------------
    Wait until Clk_com'EVENT AND Clk_com = '1' and fwrq = '0';
     head_taken <= '0';

----------------------- Folgende Takte --------------------
    -- Simuliere 16 Takte, (40 Bytes ...)

     For i in 1 to 16 loop
   Wait until Clk_com'EVENT AND Clk_com = '1';
     end loop;

     for i in 1 to 5 loop
       Hd_Bus <= Hd_Bus + "00010000110101010011000110100101";
       Hd_rdy <= '1';
    Wait until Clk_com'EVENT AND Clk_com = '1' and Hd_fetch = '1';
       Hd_rdy <= '0';

 ------------------------3. Takt-------------------------------------
    Wait until Clk_com'EVENT AND Clk_com = '1' and rt_read = '1';
     rtdta_strobe <= '1';
     RT_data <= "00000000000000101010101111001100111101010";   -- 40 bit
    Wait until Clk_com'EVENT AND Clk_com = '1' and rt_read = '0';
     rtdta_strobe <= '0';

 ------------------------4. Takt  ---------------------------
    Wait until Clk_com'EVENT AND Clk_com = '1' and fwrq = '1';
     head_taken <= '1';

 ------------------------5. Takt  ---------------------------
    Wait until Clk_com'EVENT AND Clk_com = '1' and fwrq = '0';
     head_taken <= '0';

----------------------- Folgende Takte --------------------

     For i in 1 to 8 loop
   Wait until Clk_com'EVENT AND Clk_com = '1';
     end loop;

     end loop;
  end loop;
 END process;
END;
```

Das Synthese-Script

```
/* bc_script fuer no loops (nl) - ahtin_nl_...  */
```

```
bc_enable_analysis_info =  "true" /* For BC_View */

analyze -f vhdl ht_bc.vhd

/* analyze -f vhdl ht_cf_bc.vhd */   /* fuer cycle fixed mode */

elaborate -s ht
/* reset- Setzen NACH elaborate ! */
set_behavioral_reset Reset -active high
create_clock Clk_com  -period 25
write -h -o ht_elab.db   /* save the elaborated design */

/* bc_check_design -io cycle_fixed */

bc_check_design -io superstate_fixed
/* set_operating_conditions wccom */

bc_time_design -fastest > ht_time.rep
write -h -o ht_timed.db    /*  save the timed design */
sh date >> sched.time
schedule -io superstate_fixed -effort medium  > ht_sched.rep
sh date >> sched.time
/* rename_design ahtin_scheduled */ /* keeps the names clean */
report_schedule -abstract > ht_fsm.rep
/* report_schedule -operations > ht_sched_op.rep */
report_schedule -summary > ht_sched_sum.rep

/* write -h -o ahtin_cf.db */       /* save the RTL design to a db file */
vhdlout_dont_write_types = "true"     /* prevents clashes in elaboration */
vhdlout_use_packages = { ieee.std_logic_1164,ieee.std_logic_arith }
vhdlout_equations = "true"       /* ignore gate delays */
/* vhdlout_equations = "false"*/
vhdlout_levelize = true

write -hier -f vhdl -out ht_lev_reg.vhd
                         /* saves a simulatable register_level design */
quit
```

BC-Scheduling-Reports

```
Warning: Design has already been timed. (HLS-117)
Loading db file '/afs/wsi/ti/synopsys/1997.8/libraries/syn/gtech.db'
Information: Scheduling 'loop_63_design' ... (HLS-152)
Information: Scheduling 'loop_76_design' ... (HLS-152)
Information: Scheduling 'loop_88_design' ... (HLS-152)
Information: Scheduling 'loop_93_design' ... (HLS-152)
Information: Scheduling 'loop_59_design' ... (HLS-152)
Information: Scheduling '_L0_design' ... (HLS-152)
Information: Scheduling 'htproc_design' ... (HLS-152)
Information: Allocating hardware for 'htproc_design' ... (HLS-153)
****************************************************************************
               Date      : Wed Sep 30 15:57:49 1998
               Version   : 1997.08
               Design    : ht
****************************************************************************

*****************************************
*  Operation schedule of process htproc:  *
*****************************************

Resource types
==================================
    fwrq......1-bit registered output port
    loop......loop boundaries
    p0........1-bit registered output port Hd_fetch
    p1........1-bit registered output port rt_read
    p2........16-bit registered output port RT_Addr
    p3........48-bit registered output port headout
    p4........32-bit input port Hd_Bus
    p5........40-bit input port RT_data
    p6........1-bit input port head_taken
    p7........1-bit input port ffull
    p8........1-bit input port rtdta_strobe
    p9........1-bit input port Hd_rdy



                p     p     p     p     p     p     p      p     p     p     p
                o     o     o     o     o     o     o      o     o     o     o
                r     r     r     r     r     r     r      r     r     r     r
                t     t     t     t     t     t     t      t     t     t     t
-------+------+-----+-----+-----+-----+-----+-----+-----+------+-----+-----+-----
 cycle | loop | p4  | p5  | p6  | p7  | p8  | p9  | p0  | fwrq | p1  | p2  | p3
-------------------------------------------------------------------------------
   0   |..L0..|.....|.....|.....|.....|.....|.....|.W49.|.W50..|.W51.|.....|.....
   1   |.L14..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
       |..L3..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
   2   |......|.....|.....|.....|.....|.....|.R59.|.....|......|.....|.....|.....
   3   |.L25..|.R60.|.....|.....|.....|.....|.....|.W61.|......|.....|.....|.....
       |.L24..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
       |.L15..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
```

```
   4   |.L12..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
   5   |......|.....|.....|.....|.....|.....|.R63.|.....|......|.....|.....|.....
   6   |.L23..|.....|.....|.....|.....|.....|.....|.W72.|......|.W71.|.W70.|.....
       |.L22..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
       |.L13..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
       |.L10..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
   7   |......|.....|.....|.....|.....|.R76.|.....|.....|......|.....|.....|.....
   8   |.L21..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
       |.L20..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
       |.L11..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
   9   |......|.....|.R78.|.....|.....|.....|.....|.....|......|.W84.|.....|.....
  10   |..L8..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
  11   |......|.....|.....|.....|.R88.|.....|.....|.....|......|.....|.....|.....
  12   |.L19..|.....|.....|.....|.....|.....|.....|.....|.W89..|.....|.....|.W90.
       |.L18..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
       |..L9..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
  13   |..L6..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
  14   |......|.....|.....|.R93.|.....|.....|.....|.....|......|.....|.....|.....
  15   |.L17..|.....|.....|.....|.....|.....|.....|.....|.W94..|.....|.....|.....
       |.L16..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
       |..L7..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
  16   |..L5..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
       |..L4..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
  17   |..L2..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....
       |..L1..|.....|.....|.....|.....|.....|.....|.....|......|.....|.....|.....


Operation name abbreviations
============================
    L0.......loop boundaries htproc_design_loop_begin
    L1.......loop boundaries htproc_design_loop_end
    L2.......loop boundaries htproc_design_loop_cont
    L3.......loop boundaries _L0/_L0_design_loop_begin
    L4.......loop boundaries _L0/_L0_design_loop_end
    L5.......loop boundaries _L0/_L0_design_loop_cont
    L6.......loop boundaries _L0/loop_93/loop_93_design_loop_begin
    L7.......loop boundaries _L0/loop_93/loop_93_design_loop_end
    L8.......loop boundaries _L0/loop_88/loop_88_design_loop_begin
    L9.......loop boundaries _L0/loop_88/loop_88_design_loop_end
    L10......loop boundaries _L0/loop_76/loop_76_design_loop_begin
    L11......loop boundaries _L0/loop_76/loop_76_design_loop_end
    L12......loop boundaries _L0/loop_63/loop_63_design_loop_begin
    L13......loop boundaries _L0/loop_63/loop_63_design_loop_end
    L14......loop boundaries _L0/loop_59/loop_59_design_loop_begin
    L15......loop boundaries _L0/loop_59/loop_59_design_loop_end
    L16......loop boundaries _L0/loop_93/loop_93_design_loop_cont
    L17......loop boundaries _L0/loop_93/EXIT_L93
    L18......loop boundaries _L0/loop_88/loop_88_design_loop_cont
    L19......loop boundaries _L0/loop_88/EXIT_L88
    L20......loop boundaries _L0/loop_76/loop_76_design_loop_cont
    L21......loop boundaries _L0/loop_76/EXIT_L76
    L22......loop boundaries _L0/loop_63/loop_63_design_loop_cont
    L23......loop boundaries _L0/loop_63/EXIT_L63
    L24......loop boundaries _L0/loop_59/loop_59_design_loop_cont
    L25......loop boundaries _L0/loop_59/EXIT_L59
    R59......1-bit read _L0/loop_59/Hd_rdy_59
    R60......32-bit read _L0/Hd_Bus_60
    R63......1-bit read _L0/loop_63/Hd_rdy_63
    R76......1-bit read _L0/loop_76/rtdta_strobe_76
    R78......40-bit read _L0/RT_data_78
    R88......1-bit read _L0/loop_88/ffull_88
    R93......1-bit read _L0/loop_93/head_taken_93
    W49......1-bit write Hd_fetch_49
    W50......1-bit write fwrq_50
    W51......1-bit write rt_read_51
    W61......1-bit write _L0/Hd_fetch_61
    W70......16-bit write _L0/RT_Addr_70
    W71......1-bit write _L0/rt_read_71
    W72......1-bit write _L0/Hd_fetch_72
    W84......1-bit write _L0/rt_read_84
    W89......1-bit write _L0/fwrq_89
    W90......48-bit write _L0/headout_90
    W94......1-bit write _L0/fwrq_94


********************************************************************************
Information: control FSM status and output not registered
Control FSM has 17 states
One-Hot coding style selected, state vector is 17 bits long.
State  Code
s_0_0  1----------------
s_0_1  -1---------------
s_1_4  --1--------------
s_1_9  ---1-------------
s_1_10  ----1------------
s_1_13  -----1-----------
s_1_16  ------1----------
s_2_2  -------1---------
s_2_3  --------1--------
s_3_5  ---------1-------
s_3_6  ----------1------
s_4_7  -----------1-----
s_4_8  ------------1----
```

```
s_5_11  -------------1---
s_5_12  --------------1--
s_6_14  ---------------1-
s_6_15  ----------------1

Control unit for process htproc of design ht is hardwired
Control unit for process htproc synthesized.

Memory usage during scheduling 8537 Kbytes.
CPU usage during scheduling was 9 seconds.
Scheduled design ht.
Warning: Overwriting design file '/afs/informatik.uni-tuebingen.de/home/wlange/hilan/aht/ht_bc/ht.db'. (DDB-24)
Current design is 'ht'.


****************************************************************************
               Date      : Wed Sep 30 15:58:40 1998
               Version   : 1997.08
               Design    : ht
****************************************************************************

***************************************
*  Summary report for process htproc:  *
***************************************
----------------------------------------------------------------------------
     Timing Summary
----------------------------------------------------------------------------
 Clock period 25.00
 Loop timing information:
    htproc.....................................17 cycles (cycles 0 - 17)
        _L0....................................15 cycles (cycles 1 - 16)
            loop_59.............................2 cycles (cycles 1 - 3)
                (exit) EXIT_L59..................... (cycle 3)
            loop_63.............................2 cycles (cycles 4 - 6)
                (exit) EXIT_L63..................... (cycle 6)
            loop_76.............................2 cycles (cycles 6 - 8)
                (exit) EXIT_L76..................... (cycle 8)
            loop_88.............................2 cycles (cycles 10 - 12)
                (exit) EXIT_L88..................... (cycle 12)
            loop_93.............................2 cycles (cycles 13 - 15)
                (exit) EXIT_L93..................... (cycle 15)
----------------------------------------------------------------------------
     Area Summary
----------------------------------------------------------------------------
 Estimated combinational area    186
 Estimated sequential area       1670
 TOTAL                           1856

 17 control states
 22 basic transitions
 3 control inputs
 11 control outputs

----------------------------------------------------------------------------
     Resource types
----------------------------------------------------------------------------
Register Types
=======================================
1-bit register.....................2
16-bit register....................1
48-bit register....................1

Operator Types
=======================================

I/O Ports
=======================================
1-bit input port...................4
1-bit registered output port.......3
16-bit registered output port......1
32-bit input port..................1
40-bit input port..................1
48-bit registered output port......1

----------------------------------------------------------------------------
```

12.3 Routing-Steuerung (RC)

Quellcode der VHDL-Verhaltensbeschreibung

```
-- Verhaltensbeschreibung des RC fuer
-- Synthese mit BC von Synopsys.
-- rc_bc.vhd hat ein two-way handshaking IF auch zum Header_FIFO
-- rc_bc.vhd hat Interfaces zu Head_FIfo, CC und SRC_Out,
-- in dieser Version wurde die Abfrage von 'free' umgestellt.
-- dadurch ist die Simulation stabiler.
-- moegliche Verbesserungen: siehe notes.
-- 2 Prozesse werden verwendet RCIN, RCOUT.
-- RCIN nimmt header und Routing Tag auf, untersucht den Routing Tag
-- und kommuniziert mit dem Connection Controller (CC)
```

```
-- RCOUT gibt den header an SRC weiter und kommuniziert mit SRC
-- multicast muss sein, damit der CC weiss, wann "configure" gegeben wird
-- Das Signal free gibt an, wann xmit_bsy nach einer Uebertragung
-- wieder aktiv werden darf, d.h. wann die Belegung eines Ausgangs
-- aufgehoben ist.
-- Aenderungen, um v_parser zum Laufen zu bringen:
-- use IEEE.std_logic_unsigned.all; auskommentiert
-- Aenderungen fuer BC:
-- Attribute: dont_unroll fuer die LOOPS.
-- viele Wait's gesetzt um HLS-45/etc. Fehler zu eleminieren.
-- Package ersetzt durch std. library function
-- Autor: W. Lange.   14. 9. 98

Library IEEE;
USE IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

ENTITY rc IS
  PORT (
       Clk_com      : IN STD_LOGIC;
       Reset        : IN STD_LOGIC;
       Hd_read      : OUT STD_LOGIC;            -- HEADER_FIFO
       Hd_write     : IN  STD_LOGIC;
       Hd_Data      : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
       Hd_loaded    : OUT STD_LOGIC; -- for tw-handshake
       busy         : IN  STD_LOGIC;            -- SRC_OUT
       transmit     : OUT STD_LOGIC;
       load_hd      : OUT STD_LOGIC;
       header       : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
       load_ok      : IN  STD_LOGIC;
       multicast    : OUT STD_LOGIC;   -- CC
       RR           : OUT STD_LOGIC;
       OA           : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
       WAIT_CC      : IN  STD_LOGIC;
       free         : IN STD_LOGIC;
       xmit_bsy     : OUT STD_LOGIC);
 END rc;
---------------------------------
-- Use work.utilsSRC.all; ersetzt durch Standardfunktionen

ARCHITECTURE rcar OF rc IS
  SIGNAL connected    : BOOLEAN;
  SIGNAL head_reg     : STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL hd_reg_bsy   : BOOLEAN;
  SIGNAL hd_r_b       : BOOLEAN;    -- sagt RCOUT proc, head_reg busy
  SIGNAL head_ready   : BOOLEAN;

BEGIN

 RCIN: PROCESS -- Der SRCIN Process nimmt den Header und den R-tag auf
   VARIABLE r_tag        :  STD_LOGIC_VECTOR(15 DOWNTO 0);
   VARIABLE r_tag_old    :  STD_LOGIC_VECTOR(15 DOWNTO 0);
   VARIABLE hilf         :  STD_LOGIC_VECTOR(31 DOWNTO 0);
   VARIABLE first        :  BOOLEAN;
   attribute dont_unroll : boolean;
   attribute dont_unroll of L1 : label is true;

 BEGIN -- Process
  --  Reset: Setze alle Variablen und Indices zurueck ----------------------
    RR <= '0';
    OA <= "0000";
    hd_r_b <= false;
    Hd_read <= '0';
    Hd_loaded <= '0';
    head_ready <= false;
    connected       <= FALSE;
    r_tag_old := "0000000000000000";
    r_tag     := "0000000000000000";
    multicast <= '0';
    xmit_bsy <= '0';
  Wait UNTIL Clk_com'EVENT AND Clk_com = '1';

  WHILE  TRUE loop    -- main loop ---------------------------
    first := TRUE;
   -- Warte bis das Header Reg. frei ist.
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and not hd_reg_bsy;
    -- Setze Hd_read: Lies den Zellkopf ein  -----------------
    Hd_read <= '1';
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1';   -- fuer BC(HLS-45)(10)
    -- Warte bis ein Header da ist.
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and Hd_write = '1';
    r_tag := Hd_Data(15 downto 0);
    Hd_loaded <= '1'; -- two way handshake

    Wait UNTIL Clk_com'EVENT AND Clk_com = '1';   -- fuer BC(HLS-45)
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and Hd_write = '0';
    Hd_loaded <= '0';
    hd_r_b <= true; -- kuendige den Header an
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1';   -- fuer BC
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and Hd_write = '1';
    head_reg <= Hd_Data;
    Hd_read <= '0';
    Hd_loaded <= '1';
```

```
    head_ready <= true; -- bedeutet: der Header ist geladen
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and Hd_write = '0';
    Hd_loaded <= '0';
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and  busy = '0';

    IF (r_tag_old /= r_tag) THEN
      r_tag_old := r_tag;
      xmit_bsy  <= '0';  -- CC: Gib die Verbindungen frei
      connected <= FALSE;
      Wait UNTIL Clk_com'EVENT AND Clk_com = '1';

      IF r_tag(15) = '1' THEN
         multicast <= '1';
         Wait UNTIL Clk_com'EVENT AND Clk_com = '1';   -- fuer BC(HLS-45)(7)
      ELSE
         Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
      END IF;
       -- Generierung der Ausgangsadresse, (oder Adressen bei Multicast)
      L1:  FOR i IN 13 DOWNTO  0 loop  -- Wir haben nur 14 Kanaele
        IF r_tag(i) = '1' THEN

           IF first  THEN  -- Tue's nur das erste mal
--           Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
             Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and free = '1';
             Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
             xmit_bsy <= '1';  -- Wenn die Verbindung geloest ist ..
             first := FALSE;
             -- OA <= convert(i);
     OA  <= std_logic_vector(conv_unsigned(i, 4)); -- library fct..
             RR <= '1';
           ELSE
             -- OA <= convert(i);
     OA  <= std_logic_vector(conv_unsigned(i, 4)); -- library fct..
             RR <= '1';
     Wait UNTIL Clk_com'EVENT AND Clk_com = '1'; -- fuer BC(HLS47)
           END IF;  -- IF first

   Wait UNTIL Clk_com'EVENT AND Clk_com = '1';  -- fuer BC(HLS-4
           Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and WAIT_CC = '1';
   WAIT UNTIL Clk_com'EVENT AND Clk_com = '1'; -- fuer BC (12)
           Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and WAIT_CC = '0';
           RR <= '0';
           Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
ELSE
       Wait UNTIL Clk_com'EVENT AND Clk_com = '1';   -- fuer BC(HLS-47)(5)
        END IF;    -- IF r_tag(i) = '1'
       END loop;
       Wait UNTIL Clk_com'EVENT AND Clk_com = '1';   -- fuer BC(HLS-47)(8)
       multicast <= '0';
       Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
     ELSE
       Wait UNTIL Clk_com'EVENT AND Clk_com = '1';   -- fuer BC(HLS-47)(9)

    END IF;  -- If r_tag-old /= r_tag ..

    connected <= TRUE;
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1';   -- fuer BC(HLS-45)(11)
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and not hd_reg_bsy;
        hd_r_b <= false;
        head_ready <= false;
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
   end loop;
  END process;


 RCOUT: PROCESS
  BEGIN -- Process
  --  Reset: Setze alle Variablen und Indices zurueck ----------------------
    transmit  <=  '0';
    load_hd <= '0';
    hd_reg_bsy <= false;
  Wait UNTIL Clk_com'EVENT AND Clk_com = '1';

  WHILE  TRUE loop    -- main loop ---------------------------

    -- Warte bis SRC_Out der header kommt   -----------------------
   Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and hd_r_b;
      hd_reg_bsy <= true;

   -- Warte bis der Header geladen ist. (head_ready)
   Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and head_ready;
   Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and busy = '0';
     header <= head_reg;
     load_hd <= '1';
   Wait UNTIL Clk_com'EVENT AND Clk_com = '1';  -- fuer BC(HLS-45)(1)
   -- Der header ist abgegeben, Das Register ist wieder frei
   Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and load_ok = '1';

      hd_reg_bsy <= FALSE;
      load_hd <= '0';
   Wait UNTIL Clk_com'EVENT AND Clk_com = '1';  -- fuer BC(HLS-45)(2)
```

```
   -- Warte bis die Verbindung steht, dann kann uebertragen werden  ---
   Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and connected;
      transmit <= '1';
   Wait UNTIL Clk_com'EVENT AND Clk_com = '1';  -- fuer BC(HLS-45)(3)
   Wait UNTIL Clk_com'EVENT AND Clk_com = '1' and busy = '1';
      transmit <= '0';

  END loop; -- main loop
 END Process;

END rcar;
```

Quellcode des RC-Testtreibers

```
-- Testdriver testrc.vhd fuer rc_bc.vhd
-- 3 Prozesse
-- Geaendert fuer tway handshake zw. Head-fifo und RC (16.2.98)
-- Simulation mit Synopsys Simulater V. 1997.08
-- Autor: W. Lange. 12. 2. 98

library IEEE;
library synopsys;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_unsigned.all;
   use IEEE.std_logic_misc.all;
  use IEEE.std_logic_arith.all;
--  use IEEE.std_logic_components.all;
  use synopsys.attributes.all;
  use STD.Textio.all;

Entity testrc IS END;

ARCHITECTURE test_driver OF testrc IS

COMPONENT rc
  PORT (
       Clk_com      : IN STD_LOGIC;
       Reset        : IN STD_LOGIC;
       Hd_read      : OUT STD_LOGIC;            -- HEADER_FIFO
       Hd_write     : IN  STD_LOGIC;
       Hd_Data      : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
       Hd_loaded    : OUT STD_LOGIC;             -- for tw-handshake
       busy         : IN  STD_LOGIC;            -- SRC_OUT
       transmit     : OUT STD_LOGIC;
       load_hd      : OUT STD_LOGIC;
       header       : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
       load_ok      : IN  STD_LOGIC;
       multicast    : OUT  STD_LOGIC;  -- CC
       RR           : OUT STD_LOGIC;
       OA           : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
       WAIT_CC      : IN  STD_LOGIC;
       free         : IN STD_LOGIC;
       xmit_bsy     : OUT STD_LOGIC);
END COMPONENT;

-- FOR ALL : rc  USE entity work.rc(rcar);  -- Fuer funktionale Simulation

FOR ALL : rc  USE entity WORK.rc(SYN_rcar); -- Conf. der Archit. nach Synthese

  SIGNAL  Clk_com : STD_LOGIC := '0';
  SIGNAL  Hd_read, Hd_write, transmit,load_hd,load_ok,multicast : STD_LOGIC;
  SIGNAL  RR,WAIT_CC,xmit_bsy,free,Hd_loaded  : STD_LOGIC;
  SIGNAL  busy        : STD_LOGIC;
  SIGNAL  Hd_Data     : STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL  header      : STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL  OA          : STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL  GSR,Reset         : STD_LOGIC; -- Global Set Reset von Synopsys

BEGIN

 UUT: rc port map (Clk_com,Reset,Hd_read,Hd_write,Hd_Data,Hd_loaded,busy,
       transmit,load_hd,header,load_ok,multicast,RR,OA,WAIT_CC,free,xmit_bsy);

  Clk_com <= not Clk_com after 12.5 ns;

 Head_Fifo_stim: Process

 VARIABLE rtag_singlecast : STD_LOGIC_VECTOR(31 downto 0) := "00000000000000000000000000001000" ; -- x00000008

 VARIABLE rtag_multicast : STD_LOGIC_VECTOR(31 downto 0) := "00000000000000001000100100100100" ;

 VARIABLE hilf : STD_LOGIC_VECTOR(31 downto 0) := "00100100010000001000100100100100" ; -- x24408924

 BEGIN
 -- Reset first..
     GSR <= '0';
     Reset <= '0';
     Hd_write <= '0';
     Wait until Clk_com'EVENT AND Clk_com = '1';
     Wait until Clk_com'EVENT AND Clk_com = '1';
     GSR <= '1';
     Reset <= '1';
```

```
     Wait until Clk_com'EVENT AND Clk_com = '1';
     Wait until Clk_com'EVENT AND Clk_com = '1';
     GSR <= '0';
     Reset <= '0';
     Wait until Clk_com'EVENT AND Clk_com = '1';
     Wait until Clk_com'EVENT AND Clk_com = '1';

  WHILE TRUE loop

      Wait until Clk_com'EVENT AND Clk_com = '1' and Hd_read = '1';
      -- Bringe erst den Header (komplett) mit Routing Tag
      -- die 1. 16 Bit ist der R-Tag (Kaenele B,8,5,2)
      -- die 2. und 3. Bytes sind der Header.
      Hd_write <= '1';
      Hd_Data <= rtag_singlecast;
      Wait until Clk_com'EVENT AND Clk_com = '1'and Hd_loaded = '1'; -- neu
-- fuer two-way-handshake
      Hd_write <='0';
      Wait until Clk_com'EVENT AND Clk_com = '1'and Hd_loaded = '0'; -- neu
      Hd_write <= '1';
      Hd_Data <= hilf;
      hilf := hilf + "00000000000000000000000000000001";
      Wait until Clk_com'EVENT AND Clk_com = '1';

      Wait until Clk_com'EVENT AND Clk_com = '1' and Hd_read = '0';
        Hd_write <= '0';

      FOR i in 1 to 15 loop
       Wait until Clk_com'EVENT AND Clk_com = '1';
      End loop;
      Wait until Clk_com'EVENT AND Clk_com = '1';

    --------------2. Runde mit multicast ------------------------
   FOR i in 1 to 2 loop  -- zweimal, um gleiche Ausgangsadresse zu testen

     Wait until Clk_com'EVENT AND Clk_com = '1' and Hd_read = '1';
      -- Bringe erst den Header (komplett) mit Routing Tag
      -- die 1. 16 Bit ist der R-Tag (Kaenele B,8,5,2)
      -- die 2. und 3. Bytes sind der Header.

      Hd_write <= '1';
      Hd_Data <= rtag_multicast;
      Wait until Clk_com'EVENT AND Clk_com = '1'and Hd_loaded = '1'; -- neu
-- fuer two-way-handshake
      Hd_write <='0';
      Wait until Clk_com'EVENT AND Clk_com = '1'and Hd_loaded = '0'; -- neu
      Hd_write <= '1';
      Hd_Data <= hilf;
      hilf := hilf + "00000000000000000000000000000001";
      Wait until Clk_com'EVENT AND Clk_com = '1';

      Wait until Clk_com'EVENT AND Clk_com = '1' and Hd_read = '0';
        Hd_write <= '0';

      FOR i in 1 to 3 loop
       Wait until Clk_com'EVENT AND Clk_com = '1';
      End loop;

      Wait until Clk_com'EVENT AND Clk_com = '1';
    end loop; -- zweimal

      Wait until Clk_com'EVENT AND Clk_com = '1';
    end loop;
      Wait until Clk_com'EVENT AND Clk_com = '1'; ----
   END Process;

   ---------------- Jetzt wird CC simuliert...------------------------------
 CC_Stimulus: Process
   BEGIN
    -- Reset
    WAIT_CC <= '0';
    free <= '1';
    Wait until Clk_com'EVENT AND Clk_com = '1';

    ------------------- Main loop ---------------------------------
    WHILE TRUE loop  -- main loop
     Wait until Clk_com'EVENT AND Clk_com = '1';

     IF multicast = '0' THEN

       IF xmit_bsy = '0' THEN
         Wait until Clk_com'EVENT AND Clk_com = '1';
         free <= '1';
       END IF;

       IF RR = '1' THEN   -- so ist's richtig!

         WAIT_CC <= '1';
         Wait until Clk_com'EVENT AND Clk_com = '1'; -- Achtung, neu

         free <= '0';
         For i in 1 to 2 loop
           Wait until Clk_com'EVENT AND Clk_com = '1';
```

```
        END loop;
        Wait until Clk_com'EVENT AND Clk_com = '1';
        WAIT_CC <= '0';   -- connected

        Wait until Clk_com'EVENT AND Clk_com = '1';
        Wait until Clk_com'EVENT AND Clk_com = '1';

        -- Warte etwas, bis die Verbindung steht..

        Wait until Clk_com'EVENT AND Clk_com = '1' and xmit_bsy = '1';
        Wait until Clk_com'EVENT AND Clk_com = '1';
        Wait until Clk_com'EVENT AND Clk_com = '1';
      END IF;

   ELSE   -- multicast = '1'

     IF RR = '1' THEN

        WAIT_CC <= '1';
        free <= '0';
        Wait until Clk_com'EVENT AND Clk_com = '1';

        Wait until Clk_com'EVENT AND Clk_com = '1';
        WAIT_CC <= '0';   -- connected
        Wait until Clk_com'EVENT AND Clk_com = '1';
        Wait until Clk_com'EVENT AND Clk_com = '1'  and   RR = '0';

        Wait until Clk_com'EVENT AND Clk_com = '1';

    END IF;
       Wait until Clk_com'EVENT AND Clk_com = '1';  -- neu
  END IF;
      Wait until Clk_com'EVENT AND Clk_com = '1';
  End loop;
  Wait until Clk_com'EVENT AND Clk_com = '1';  -- neu
 End Process;

   -------  SRC_Out wird simuliert...------------------------------
 SRC_OUT_Stim: Process

 BEGIN
    busy <= '0';
    load_ok <= '0';

    Wait until Clk_com'EVENT AND Clk_com = '1';

   WHILE TRUE loop
     Wait until Clk_com'EVENT AND Clk_com = '1' and load_hd = '1';
     -- Der header kann fuer den transmit geladen werden:
     Wait until Clk_com'EVENT AND Clk_com = '1';
     load_ok <= '1';
     busy <= '1';

     Wait until Clk_com'EVENT AND Clk_com = '1' and load_hd = '0';
     load_ok <= '0';

     Wait until Clk_com'EVENT AND Clk_com = '1';   ----
     Wait  until Clk_com'EVENT AND Clk_com = '1' and transmit = '1';
     -- Warte bis die ganze Zelle uebertragen ist:

     For i in 1 to 12 loop
Wait until Clk_com'EVENT AND Clk_com = '1';
     End loop;

     busy <= '0';
     Wait until Clk_com'EVENT AND Clk_com = '1';
 END loop; -- main loop;
 Wait until Clk_com'EVENT AND Clk_com = '1';   ----
 END Process;
END;
```

Das Synthese-Script

```
/* bc_script fuer rc_bc.vhdl  */

bc_enable_analysis_info =  "true" /* For BC_View */

-- analyze -f vhdl utilsSRC.vhd -- nicht mehr noetig
analyze -f vhdl rc_bc.vhd

elaborate -s rc
/* reset- Setzen NACH elaborate ! */
set_behavioral_reset Reset -active high
create_clock Clk_com  -period 25
write -h -o rc_elab.db   /* save the elaborated design */

/* bc_check_design -io cycle_fixed */

bc_check_design -io superstate_fixed
/* set_operating_conditions wccom */
```

```
bc_time_design -fastest > rc_time.rep
write -h -o ht_timed.db    /*  save the timed design */

schedule -io superstate_fixed -effort medium  > rc_sched.rep

/* rename_design ahtin_scheduled */ /* keeps the names clean */
/* report_schedule -abstract > rc_fsm.rep */
report_schedule -summary > rc_sched_sum.rep

/* write -h -o ahtin_cf.db */       /* save the RTL design to a db file */
vhdlout_dont_write_types = "true"     /* prevents clashes in elaboration */
vhdlout_use_packages = { ieee.std_logic_1164,ieee.std_logic_arith }
vhdlout_equations = "true"       /* ignore gate delays */
vhdlout_levelize = true

write -hier -f vhdl -out rc_lev_reg.vhd
                        /* saves a simulatable register_level design */

quit
```

BC-Scheduling-Report

```
Warning: Design has already been timed. (HLS-117)
Loading db file '/afs/wsi/ti/synopsys/1997.8/libraries/syn/gtech.db'
Information: Scheduling 'loop_186_design' ... (HLS-152)
Information: Scheduling 'loop_201_design' ... (HLS-152)
Information: Scheduling 'loop_187_design' ... (HLS-152)
Information: Scheduling 'loop_192_design' ... (HLS-152)
Information: Scheduling 'loop_182_design' ... (HLS-152)
Information: Scheduling 'loop_198_design' ... (HLS-152)
Information: Scheduling '_L1_design' ... (HLS-152)
Information: Scheduling 'RCOUT_design' ... (HLS-152)
Information: Allocating hardware for 'RCOUT_design' ... (HLS-153)
****************************************************************************
              Date      : Wed Sep 30 16:03:45 1998
              Version   : 1997.08
              Design    : rc
****************************************************************************

****************************************
*  Operation schedule of process RCOUT:  *
****************************************

Resource types
====================================
    busy......1-bit input port
    loop......loop boundaries
    p0........1-bit registered output port transmit
    p1........1-bit registered output port load_hd
    p2........1-bit registered output port hd_reg_bsy
    p3........32-bit registered output port header
    p4........32-bit input port head_reg
    p5........1-bit input port connected
    p6........1-bit input port load_ok
    p7........1-bit input port head_ready
    p8........1-bit input port hd_r_b


                  p      p      p      p      p      p      p      p      p      p
                  o      o      o      o      o      o      o      o      o      o
                  r      r      r      r      r      r      r      r      r      r
                  t      t      t      t      t      t      t      t      t      t
-------+------+------+------+------+------+------+------+------+------+------+------
 cycle | loop |  p4  | busy |  p5  |  p6  |  p7  |  p8  |  p0  |  p1  |  p2  |  p3
-----------------------------------------------------------------------------------
   0   |..L0..|......|......|......|......|......|......|.W174.|.W175.|.W176.|......
   1   |.L16..|......|......|......|......|......|......|......|......|......|......
       |..L3..|......|......|......|......|......|......|......|......|......|......
   2   |......|......|......|......|......|......|.R182.|......|......|......|......
   3   |.L29..|......|......|......|......|......|......|......|.W183.|......|......
       |.L28..|......|......|......|......|......|......|......|......|......|......
       |.L17..|......|......|......|......|......|......|......|......|......|......
       |.L14..|......|......|......|......|......|......|......|......|......|......
   4   |......|......|......|......|.R186.|......|......|......|......|......|......
   5   |.L27..|......|......|......|......|......|......|......|......|......|......
       |.L26..|......|......|......|......|......|......|......|......|......|......
       |.L15..|......|......|......|......|......|......|......|......|......|......
       |.L12..|......|......|......|......|......|......|......|......|......|......
   6   |......|......|.R187.|......|......|......|......|......|......|......|......
   7   |.L25..|.R188.|......|......|......|......|......|......|.W189.|......|.W188.
       |.L24..|......|......|......|......|......|......|......|......|......|......
       |.L13..|......|......|......|......|......|......|......|......|......|......
   8   |.L10..|......|......|......|......|......|......|......|......|......|......
   9   |......|......|......|......|.R192.|......|......|......|......|......|......
  10   |.L23..|......|......|......|......|......|......|......|.W195.|.W194.|......
       |.L22..|......|......|......|......|......|......|......|......|......|......
       |.L11..|......|......|......|......|......|......|......|......|......|......
  11   |..L8..|......|......|......|......|......|......|......|......|......|......
  12   |......|......|......|.R198.|......|......|......|......|......|......|......
  13   |.L21..|......|......|......|......|......|......|.W199.|......|......|......
       |.L20..|......|......|......|......|......|......|......|......|......|......
```

```
       |..L9..|......|......|......|......|......|......|......|......|......|......
  14   |..L6..|......|......|......|......|......|......|......|......|......|......
  15   |......|......|.R201.|......|......|......|......|......|......|......|......
  16   |.L19..|......|......|......|......|......|.W202.|......|......|......|......
       |.L18..|......|......|......|......|......|......|......|......|......|......
       |..L7..|......|......|......|......|......|......|......|......|......|......
       |..L5..|......|......|......|......|......|......|......|......|......|......
       |..L4..|......|......|......|......|......|......|......|......|......|......
       |..L2..|......|......|......|......|......|......|......|......|......|......
       |..L1..|......|......|......|......|......|......|......|......|......|......


Operation name abbreviations
============================
    L0.......loop boundaries RCOUT_design_loop_begin
    L1.......loop boundaries RCOUT_design_loop_end
    L2.......loop boundaries RCOUT_design_loop_cont
    L3.......loop boundaries _L1/_L1_design_loop_begin
    L4.......loop boundaries _L1/_L1_design_loop_end
    L5.......loop boundaries _L1/_L1_design_loop_cont
    L6.......loop boundaries _L1/loop_201/loop_201_design_loop_begin
    L7.......loop boundaries _L1/loop_201/loop_201_design_loop_end
    L8.......loop boundaries _L1/loop_198/loop_198_design_loop_begin
    L9.......loop boundaries _L1/loop_198/loop_198_design_loop_end
    L10......loop boundaries _L1/loop_192/loop_192_design_loop_begin
    L11......loop boundaries _L1/loop_192/loop_192_design_loop_end
    L12......loop boundaries _L1/loop_187/loop_187_design_loop_begin
    L13......loop boundaries _L1/loop_187/loop_187_design_loop_end
    L14......loop boundaries _L1/loop_186/loop_186_design_loop_begin
    L15......loop boundaries _L1/loop_186/loop_186_design_loop_end
    L16......loop boundaries _L1/loop_182/loop_182_design_loop_begin
    L17......loop boundaries _L1/loop_182/loop_182_design_loop_end
    L18......loop boundaries _L1/loop_201/loop_201_design_loop_cont
    L19......loop boundaries _L1/loop_201/EXIT_L201
    L20......loop boundaries _L1/loop_198/loop_198_design_loop_cont
    L21......loop boundaries _L1/loop_198/EXIT_L198
    L22......loop boundaries _L1/loop_192/loop_192_design_loop_cont
    L23......loop boundaries _L1/loop_192/EXIT_L192
    L24......loop boundaries _L1/loop_187/loop_187_design_loop_cont
    L25......loop boundaries _L1/loop_187/EXIT_L187
    L26......loop boundaries _L1/loop_186/loop_186_design_loop_cont
    L27......loop boundaries _L1/loop_186/EXIT_L186
    L28......loop boundaries _L1/loop_182/loop_182_design_loop_cont
    L29......loop boundaries _L1/loop_182/EXIT_L182
    R182.....1-bit read _L1/loop_182/hd_r_b_182
    R186.....1-bit read _L1/loop_186/head_ready_186
    R187.....1-bit read _L1/loop_187/busy_187
    R188.....32-bit read _L1/head_reg_188
    R192.....1-bit read _L1/loop_192/load_ok_192
    R198.....1-bit read _L1/loop_198/connected_198
    R201.....1-bit read _L1/loop_201/busy_201
    W174.....1-bit write transmit_174
    W175.....1-bit write load_hd_175
    W176.....1-bit write hd_reg_bsy_176
    W183.....1-bit write _L1/hd_reg_bsy_183
    W188.....32-bit write _L1/header_188
    W189.....1-bit write _L1/load_hd_189
    W194.....1-bit write _L1/hd_reg_bsy_194
    W195.....1-bit write _L1/load_hd_195
    W199.....1-bit write _L1/transmit_199
    W202.....1-bit write _L1/transmit_202


************************************************************************
Information: control FSM status and output not registered
Control FSM has 17 states
One-Hot coding style selected, state vector is 17 bits long.
State  Code
s_0_0  1----------------
s_0_1  -1---------------
s_1_8  --1--------------
s_1_11  ---1-------------
s_1_14  ----1------------
s_2_2  -----1-----------
s_2_3  ------1----------
s_3_4  -------1---------
s_3_5  --------1--------
s_4_6  ---------1-------
s_4_7  ----------1------
s_5_9  -----------1-----
s_5_10  ------------1----
s_6_12  -------------1---
s_6_13  --------------1--
s_7_15  ---------------1-
s_7_16  ----------------1

Control unit for process RCOUT of design rc is hardwired
Control unit for process RCOUT synthesized.
Information: Scheduling 'loop_163_design' ... (HLS-152)
Information: Scheduling 'loop_91_design' ... (HLS-152)
Information: Scheduling 'loop_86_design' ... (HLS-152)
Information: Scheduling 'loop_96_design' ... (HLS-152)
Information: Scheduling 'loop_144_design' ... (HLS-152)
```

```
Information: Scheduling 'loop_129_design' ... (HLS-152)
Information: Scheduling 'loop_146_design' ... (HLS-152)
Information: Scheduling 'L1_design' ... (HLS-152)
Information: Scheduling 'loop_100_design' ... (HLS-152)
Information: Scheduling 'loop_109_design' ... (HLS-152)
Information: Scheduling 'loop_106_design' ... (HLS-152)
Information: Scheduling '_L0_design' ... (HLS-152)
Information: Scheduling 'RCIN_design' ... (HLS-152)
Information: Allocating hardware for 'RCIN_design' ... (HLS-153)
*******************************************************************************
              Date      : Wed Sep 30 16:04:28 1998
              Version   : 1997.08
              Design    : rc
*******************************************************************************

****************************************
*  Operation schedule of process RCIN:  *
****************************************

Resource types
====================================
    OA.........4-bit registered output port
    RR.........1-bit registered output port
    busy.......1-bit input port
    free.......1-bit input port
    loop.......loop boundaries
    p0.........1-bit registered output port connected
    p1.........1-bit registered output port hd_r_b
    p2.........1-bit registered output port head_ready
    p3.........1-bit registered output port Hd_read
    p4.........1-bit registered output port Hd_loaded
    p5.........1-bit registered output port xmit_bsy
    p6.........1-bit registered output port multicast
    p7.........32-bit registered output port head_reg
    p8.........32-bit input port Hd_Data
    p9.........1-bit input port hd_reg_bsy
    p10........1-bit input port WAIT_CC
    p11........1-bit input port Hd_write
    r2152......(5_5->1)-bit DW01_cmp2
    r2162......(5->5)-bit DW01_dec
```

cycle	loop	p8	p9	p10	free	busy	p11	r2152	r2162	RR	OA	p0	p1	p2	p3	p4	p5	p6	p7
0	..L0..									.W70..	.W71..	.W76..	.W72..	.W75..	.W73..	.W74..	.W80..	.W79..	
1	.L20..																		
	..L3..																		
2			.R86..																
3	.L49..														.W88..				
	.L48..																		
	.L21..																		
4	.L18..																		
5							.R91..												
6	.L47..	.R92..														.W93..			
	.L46..																		
	.L19..																		
7	.L16..																		
8							.R96..												
9	.L45..												.W98..			.W97..			
	.L44..																		
	.L17..																		
10	.L14..																		
11							.R100.												
12	.L43..	.R101.												.W104.	.W102.	.W103.			.W101.
	.L42..																		
	.L15..																		
13	.L12..																		
14							.R106.												
15	.L41..															.W107.			
	.L40..																		
	.L13..																		
16	.L10..																		
17						.R109.													
18	.L39..											.W114.					.W113.		
	.L38..																		
	.L11..																		
19																	.W118.		
20	..L8..						.o124..			.W139.	.W138.								
21	.L30..																		
	.L25..																		

```
  22   |......|......|......|......|.R129.|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
  23   |.L37..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
       |.L36..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
       |.L31..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
  24   |......|......|......|......|......|......|......|.......|.......|.W135.|.W134.|......|......|......|......|.......|.W131.|......|
  25   |.L28..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
  26   |......|......|......|.R144.|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
  27   |.L35..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
       |.L34..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
       |.L29..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
  28   |.L26..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
  29   |......|......|......|.R146.|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
  30   |.L33..|......|......|......|......|......|......|.......|.o124a.|.W147.|......|......|......|......|......|......|......|......|
       |.L32..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
       |.L27..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
  31   |.L24..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
       |..L9..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
  32   |......|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|.......|.W154.|......|
  33   |......|......|......|......|......|......|......|.......|.......|......|.W161.|......|......|......|......|......|......|......|
  34   |..L6..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
  35   |......|......|.R163.|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
  36   |.L23..|......|......|......|......|......|......|.......|.......|......|......|.W164.|.W165.|......|......|......|......|......|
       |.L22..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
       |..L7..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
  37   |..L5..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
       |..L4..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
       |..L2..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|
       |..L1..|......|......|......|......|......|......|.......|.......|......|......|......|......|......|......|......|......|......|


Operation name abbreviations
============================
    L0.........loop boundaries RCIN_design_loop_begin
    93 operations removed w.l.
    o124a......(5_1->5)-bit ADD_TC_OP _L0/L1/add_124


****************************************************************************
Information: control FSM status and output not registered
Control FSM has 55 states
One-Hot coding style selected, state vector is 55 bits long.
State  Code
s_0_0  1-----------------------------------------------------
         53 state names removed w.l.
s_12_36  -----------------------------------------------------1

Control unit for process RCIN of design rc is hardwired
Control unit for process RCIN synthesized.

Memory usage during scheduling 22585 Kbytes.
CPU usage during scheduling was 55 seconds.
Scheduled design rc.
Warning: Overwriting design file '/afs/informatik.uni-tuebingen.de/home/wlange/hilan/aht/rc_bc/rc.db'. (DDB-24)
Current design is 'rc'.
```

12.4 Schieberegister-Steuerung (SRC)

Quellcode der VHDL-Verhaltensbeschreibung

```
-- Verhaltensbeschreibung des SRC_Out fuer
-- Simulation mit Synopsys
-- Synthese mit Synopsys BC
-- src_bc.vhd hat einen Prozess
-- Umgeschrieben fuer Synopsys BC
-- Prozeduren shift und encode wieder inline eingearbeitet. - Fuer BC
-- Reset eingefuehrt
-- Das Schieberegister wird ausgelagert, da BC einen
-- eine Taktzyclus hinzufuegt.                              - Fuer BC
-- Letzte Version von srcout.vhd mit 4B/5B encoder
-- Am Anfang vor jeder Uebertragung einer Header-Zelle
-- steht das Header-Startzeichen: "11001" (X19)
-- Am Anfang vor jeder Uebertragung eines 32 - bit - Reg
-- steht das Payload-Register-Startzeichen: "10001" (X11)
-- Autor: W. Lange 3.3.98

Library IEEE;
USE IEEE.std_logic_1164.all;
--  use IEEE.std_logic_unsigned.all;
--  use IEEE.std_logic_misc.all;
    use IEEE.std_logic_arith.all;
--  use IEEE.std_logic_components.all;
--  use synopsys.attributes.all;
--  use STD.Textio.all;

ENTITY srcout IS
  PORT (Clk_xmit    : IN STD_LOGIC;
        Reset       : IN  STD_LOGIC;            -- Fuer BC
        busy        : OUT STD_LOGIC;            -- von und zu SRC_IN
        transmit    : IN  STD_LOGIC;
```

```
       load_hd      : IN  STD_LOGIC;
       header       : IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
       load_ok      : OUT STD_LOGIC;
       Cell_read    : OUT STD_LOGIC;            -- Cell_FIFO
       Cell_write   : IN  STD_LOGIC;
       Cell_word    : IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
       Cell_loaded  : OUT  STD_LOGIC;
--     Cell_data    : OUT  STD_LOGIC   -- Datenleitung entfernt
       sr_ready     : IN   STD_LOGIC;  -- neu fuer SR
       sr_strobe    : OUT  STD_LOGIC;  -- neu fuer SR - Date
       shiftreg     : OUT  STD_LOGIC_VECTOR(47 DOWNTO 0)); -- neuer Ausgang
END srcout;
-------------------------------------------------------------------------
Use work.utils_src.all;

ARCHITECTURE srcoutar OF srcout IS

BEGIN

  Shift_Proc: PROCESS

--  VARIABLE aux_reg : STD_LOGIC_VECTOR(31 DOWNTO 0);
  VARIABLE shift_reg : STD_LOGIC_VECTOR(47 DOWNTO 0);
  attribute dont_unroll : boolean;
  attribute dont_unroll of L3 : label is true;


  Begin  -- Process
      busy <= '0';
      load_ok <= '0';
      Cell_loaded  <=  '0';
      Cell_read  <=  '0';
      sr_strobe  <= '0';
      shift_reg := "000000000000000000000000000000000000000000000000";
      shiftreg  <= "000000000000000000000000000000000000000000000000";
    Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1';

    WHILE TRUE LOOP
     -- Load Header -----------------------------------------------
     Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1' and load_hd = '1';
     Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1';
       load_ok <= '1';
       busy <= '1';
     Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1'; -- f. BC(HLS-45)

     -- Lade Schieberegister und encodiere den Header
     --  load_reg_encode; encode wandelt 4 Bit in 5 Bit um

        shift_reg(42 downto 38) := encd(header(31 downto 28));
  shift_reg(37 downto 33) := encd(header(27 downto 24));
        shift_reg(32 downto 28) := encd(header(23 downto 20));
        shift_reg(27 downto 23) := encd(header(19 downto 16));
        shift_reg(22 downto 18) := encd(header(15 downto 12));
        shift_reg(17 downto 13) := encd(header(11 downto 8));
shift_reg(12 downto 8) := encd(header(7 downto 4));
  shift_reg(7 downto 3) := encd(header(3 downto 0));

     Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1' and load_hd = '0';
       load_ok <= '0';
     Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1'; -- f. BC(HLS-45)

     -- Warte auf transmit und sr_ready   ----------------------

       Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1' and transmit = '1';
       Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1' and sr_ready = '1';

       shift_reg(47 downto 43) := "11001";   -- lade das "Start_Cell" Zeichen

      -- shift;-- call Procedure shift - Schiebe den Header durch den Switch
      -- entfernt

      sr_strobe <= '1';
      shiftreg  <= shift_reg;     -- lade das Schieberegister

    Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1';
    Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1' and sr_ready ='0';
     sr_strobe <= '0';
    Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1';

      -- Schiebe die Payload durch den Switch    ----------------------
  L3: FOR k IN 1 TO 12 loop
  Cell_read <= '1';
Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1'; -- f. BC(HLS-45)
        -- Warte auf das Zell-Wort
        Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1' and Cell_write = '1';

        -- Lade Schieberegister und encodiere die Daten (load_reg_encode;)

  shift_reg(42 downto 38) := encd(Cell_word(31 downto 28));
  shift_reg(37 downto 33) := encd(Cell_word(27 downto 24));
        shift_reg(32 downto 28) := encd(Cell_word(23 downto 20));
        shift_reg(27 downto 23) := encd(Cell_word(19 downto 16));
        shift_reg(22 downto 18) := encd(Cell_word(15 downto 12));
```

```
        shift_reg(17 downto 13) := encd(Cell_word(11 downto 8));
shift_reg(12 downto 8) := encd(Cell_word(7 downto 4));
  shift_reg(7 downto 3) := encd(Cell_word(3 downto 0));

        Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1';
          Cell_loaded <= '1';
          Cell_read   <= '0';
Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1'; -- f. BC(HLS-45)
        Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1' and Cell_write = '0';
        Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1'; -- f. BC(HLS-45)
  Cell_loaded <= '0';

        Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1'; -- f. BC(HLS-45)
        Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1' and sr_ready = '1';
        Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1'; -- f. BC(HLS-45)


       shift_reg(47 downto 43) := "10001";   -- lade das "Start_Reg" Zeichen

       -- Schiebe ein Payload-Wort durch den Switch (call proc shift) -----
       -- entfernt

       sr_strobe <= '1';
       shiftreg  <= shift_reg;     -- lade das Schieberegister

       Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1';
       Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1' and sr_ready = '0';
         sr_strobe <= '0';
       Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1';

     END loop;  -- For i = 1 to 12

     busy <= '0';
     Wait UNTIL Clk_xmit'EVENT AND Clk_xmit = '1';
    END loop;
   END Process;

  END srcoutar;
```

Quellcode des SRC-Package

```
-----------------------------------------------------
Library IEEE;
USE IEEE.std_logic_1164.all;
    use IEEE.std_logic_arith.all;

package utils_src is
  subtype slv5 is STD_LOGIC_VECTOR(4 downto 0);
  function encd(v: STD_LOGIC_VECTOR(3 downto 0)) return slv5;

end;

-----------------------------------------------------

package body utils_src is


function encd(v : STD_LOGIC_VECTOR(3 downto 0)) return slv5 IS
--
-- Encapsulation of logic below according to sold BC users guide
-- Optimizing Timing and Area
-- return type has changed from std_logic_vector to slv5
--
-- synopsys preserve_function
--
  VARIABLE temp   : STD_LOGIC_VECTOR(4 downto 0);
  VARIABLE vector : STD_LOGIC_VECTOR(3 downto 0);

  begin
    vector := v;

    CASE vector IS
WHEN  "0000"  => temp := "11110";
WHEN  "0001"  => temp := "01001";
        WHEN  "0010"  => temp := "10100";
WHEN  "0011"  => temp := "10101";
WHEN  "0100"  => temp := "01010";
        WHEN  "0101"  => temp := "01011";
        WHEN  "0110"  => temp := "01110";
WHEN  "0111"  => temp := "01111";
WHEN  "1000"  => temp := "10010";
WHEN  "1001"  => temp := "10011";
WHEN  "1010"  => temp := "10110";
WHEN  "1011"  => temp := "10111";
WHEN  "1100"  => temp := "11010";
WHEN  "1101"  => temp := "11011";
WHEN  "1110"  => temp := "11100";
WHEN  "1111"  => temp := "11101";
WHEN OTHERS => temp := "00000"; -- Probleme mit v_parser
     END CASE;
```

```
   return temp;
  end encd;
end utils_src;
-------------------------------------------------
```

Quellcode des SRC-Testtreibers

```
-- Testdriver testsrcout.vhd fuer srcout.vhd
-- testet src_bc mit 2 Prozessen.
-- Der zweite Prozess nimmt die Daten wieder auf
-- Simulation mit Synopsys Simulater v1997.08
-- Autor: W. Lange. Maerz 1997
-- veraendert fuer ausgelagertes Shift-Register
-- 3. 3. 98 Fuer synthetisierte Architektur: Reset eingefuehrt.
-- takedata-Prozess entfernt

library IEEE;
library synopsys;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_unsigned.all;
  use IEEE.std_logic_misc.all;
  use IEEE.std_logic_arith.all;
--  use IEEE.std_logic_components.all;
  use synopsys.attributes.all;
  use STD.Textio.all;

Entity testsrcout IS  END;


ARCHITECTURE test_driver OF testsrcout IS


COMPONENT srcout
  PORT (Clk_xmit    : IN STD_LOGIC;
       Reset        : IN  STD_LOGIC;             -- Fuer BC
       busy         : OUT STD_LOGIC;             -- von und zu SRC_IN
       transmit     : IN  STD_LOGIC;
       load_hd      : IN  STD_LOGIC;
       header       : IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
       load_ok      : OUT STD_LOGIC;
       Cell_read    : OUT STD_LOGIC;            -- Cell_FIFO
       Cell_write   : IN  STD_LOGIC;
       Cell_word    : IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
       Cell_loaded  : OUT  STD_LOGIC;
--     Cell_data    : OUT  STD_LOGIC  -- Datenleitung entfernt
       sr_ready     : IN   STD_LOGIC;
       sr_strobe    : OUT  STD_LOGIC;  -- neu fuer SR - Date
       shiftreg     : OUT  STD_LOGIC_VECTOR(47 DOWNTO 0)); -- neuer Ausgang
END component;

-- FOR ALL : srcout  USE entity work.srcout(srcoutar);
FOR ALL : srcout  USE entity work.srcout(SYN_srcoutar);

  SIGNAL  Clk_xmit : STD_LOGIC := '0';
  SIGNAL transmit,load_hd,load_ok,Cell_read,Cell_write: STD_LOGIC;
  SIGNAL  busy,Reset,sr_strobe   : STD_LOGIC;
  SIGNAL  header,Cell_word       : STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL  Cell_loaded,sr_ready   : STD_LOGIC;
  SIGNAL  GSR                    : STD_LOGIC;
  SIGNAL  shiftreg               : STD_LOGIC_VECTOR(47 DOWNTO 0);

BEGIN

 UUT: srcout port map (Clk_xmit,Reset,busy,transmit,load_hd,header,
      load_ok,Cell_read,Cell_write,Cell_word,Cell_loaded,sr_ready,
      sr_strobe,shiftreg);

  -- Clock - Prozess ------------------------------------------------
  Clk_xmit <= not Clk_xmit after 12.5 ns;

 SRC_OT: Process

 VARIABLE init : STD_LOGIC_VECTOR(31 downto 0) := "00000000000000000000000000000000";
 VARIABLE data_init:STD_LOGIC_VECTOR(31 downto 0):="10000100110000101010001111100001";
 VARIABLE keep_data:STD_LOGIC_VECTOR(31 downto 0):="10000100110000101010001111100001";
-- VARIABLE  take_data              : STD_LOGIC_VECTOR(31 DOWNTO 0);

 BEGIN
 -- Clk_xmit sollte mit der Prozedur Clock loslaufen....

 -- Reset first..
     Reset <= '1';
     GSR <= '0';             -- Global reset
     transmit <= '0'; load_hd <= '0'; Cell_write <= '0';
     header <= init;
     Wait until Clk_xmit'EVENT AND Clk_xmit = '1'; -- Warte ein wenig
     GSR <= '1';
     Reset <= '1';
     Wait until Clk_xmit'EVENT AND Clk_xmit = '1';  -- Warte ein wenig
     Wait until Clk_xmit'EVENT AND Clk_xmit = '1';  -- Warte ein wenig
     GSR <= '0';
     Reset <= '0';
```

```
     Wait until Clk_xmit'EVENT AND Clk_xmit = '1';  -- Warte ein wenig
     Wait until Clk_xmit'EVENT AND Clk_xmit = '1';  -- Warte ein wenig

     WHILE TRUE loop
      -- bringe erst den Header
      header <= data_init;
        load_hd <= '1';

      Wait until Clk_xmit'EVENT AND Clk_xmit = '1' and load_ok = '1';
         load_hd <= '0';
      Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
      Wait until Clk_xmit'EVENT AND Clk_xmit = '1';  -- neu
      Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
         transmit <= '1';
     Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
     Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
        transmit <= '0';
    Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
    Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
    Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
      -- Gib ihm eine ganze Payload.. ------------------------------
      FOR i in 1 to 12 loop
         Wait until Clk_xmit'EVENT AND Clk_xmit = '1' and Cell_read = '1';
 Cell_write <= '1';
         keep_data := keep_data + 1;
 Cell_word <= keep_data;
         Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
         Wait until Clk_xmit'EVENT AND Clk_xmit = '1' and Cell_loaded = '1';
         Cell_write <= '0';
         Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
         Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
      End loop;
    Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
    end loop;

   END Process;


 -------------simuliert Komm mit SR
setsr_ready: Process
 -- VARIABLE  take_data  : STD_LOGIC_VECTOR(31 DOWNTO 0);

   BEGIN
     sr_ready <= '0';
     Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
     Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
     sr_ready <= '1';
     Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
     Wait until Clk_xmit'EVENT AND Clk_xmit = '1';

   WHILE TRUE loop
    Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
    Wait until Clk_xmit'EVENT AND Clk_xmit = '1' and sr_strobe = '1';
    Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
      sr_ready <= '0';
    Wait until Clk_xmit'EVENT AND Clk_xmit = '1'; -- simulate
    Wait until Clk_xmit'EVENT AND Clk_xmit = '1'; -- transmission time
    Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
    Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
    Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
     sr_ready <= '1';
    Wait until Clk_xmit'EVENT AND Clk_xmit = '1';
   End loop;
  End Process;

END;
```

Das Synthese-Script

```
/* bc_script fuer src_bc.vhdl  */
sh date >> laufzeit

bc_enable_analysis_info =  "true" /* For BC_View */
analyze -f vhdl utils_src.vhd
analyze -f vhdl src_bc.vhd
elaborate -s srcout

/* reset- Setzen NACH elaborate ! */
set_behavioral_reset Reset -active high
sh_date >> laufzeit
create_clock Clk_xmit  -period 25
write -h -o src_elab.db   /* save the elaborated design */

/* bc_check_design -io cycle_fixed */

bc_check_design -io superstate_fixed
/* set_operating_conditions wccom */
sh date >> laufzeit
bc_time_design -force -fastest > src_time.rep
write -h -o src_timed.db    /*  save the timed design */
sh date >> laufzeit
```

```
schedule -io superstate_fixed  -effort medium > src_sched.rep
/* rename_design ahtin_scheduled */ /* keeps the names clean */
report_schedule -abstract > src_fsm.rep
/* report_schedule -operations ** schon in src_sched enthalten  */
report_schedule -summary > src_sched_sum.rep

/* write -h -o ahtin_cf.db */       /* save the RTL design to a db file */
vhdlout_dont_write_types = "true"     /* prevents clashes in elaboration */
vhdlout_use_packages = { ieee.std_logic_1164,ieee.std_logic_arith }
vhdlout_equations = "true"       /* ignore gate delays */
vhdlout_levelize = true

/* write -hier -f vhdl -out src_nolev_reg.vhd */ /*schreibt Komponenten raus */
write -hier -f vhdl -out src_reg.vhd  /* saves a simulatable register_level */
sh date >> laufzeit                     /*   design */

quit
```

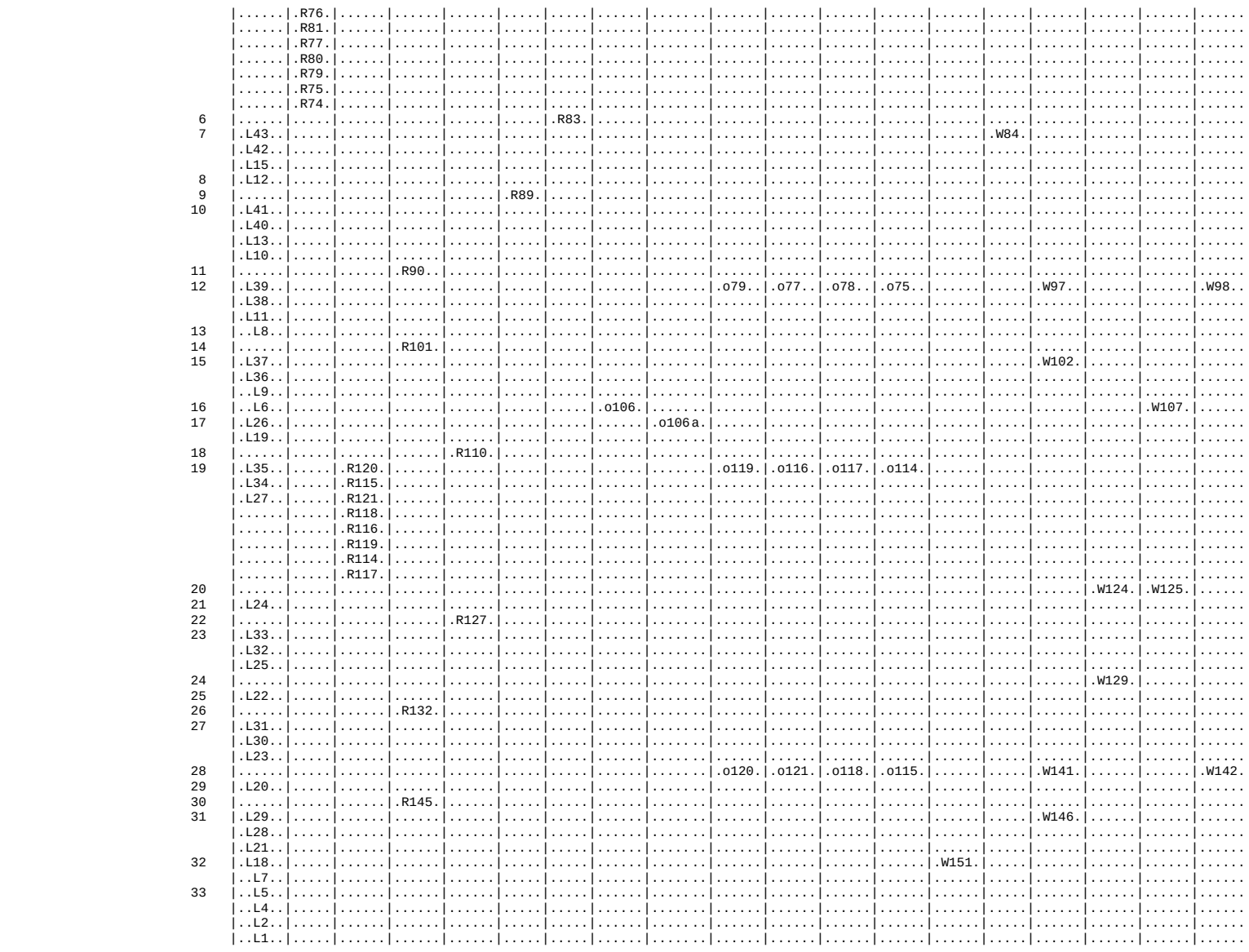
BC-Scheduling-Reports

```
Warning: Design has already been timed. (HLS-117)
Loading db file '/afs/wsi/ti/synopsys/1997.8/libraries/syn/gtech.db'
Information: Scheduling 'loop_90_design' ... (HLS-152)
Information: Scheduling 'loop_83_design' ... (HLS-152)
Information: Scheduling 'loop_145_design' ... (HLS-152)
Information: Scheduling 'loop_110_design' ... (HLS-152)
Information: Scheduling 'loop_127_design' ... (HLS-152)
Information: Scheduling 'loop_132_design' ... (HLS-152)
Information: Scheduling 'L3_design' ... (HLS-152)
Information: Scheduling 'loop_89_design' ... (HLS-152)
Information: Scheduling 'loop_65_design' ... (HLS-152)
Information: Scheduling 'loop_101_design' ... (HLS-152)
Information: Scheduling '_L0_design' ... (HLS-152)
Information: Scheduling 'Shift_Proc_design' ... (HLS-152)
Information: Allocating hardware for 'Shift_Proc_design' ... (HLS-153)
****************************************************************************
              Date      : Wed Sep 30 16:35:20 1998
              Version   : 1997.08
              Design    : srcout
****************************************************************************

*********************************************
*  Operation schedule of process Shift_Proc:  *
*********************************************

Resource types
===================================
    busy......1-bit registered output port
    loop......loop boundaries
    p0........1-bit registered output port load_ok
    p1........1-bit registered output port sr_strobe
    p2........1-bit registered output port Cell_loaded
    p3........1-bit registered output port Cell_read
    p4........48-bit registered output port shiftreg
    p5........32-bit input port header
    p6........32-bit input port Cell_word
    p7........1-bit input port sr_ready
    p8........1-bit input port Cell_write
    p9........1-bit input port transmit
    p10.......1-bit input port load_hd
    r362......(4->5)-bit encd
    r379......(4_4->1)-bit DW01_cmp2
    r388......(4->4)-bit DW01_inc
    r394......(4->5)-bit encd
    r395......(4->5)-bit encd
    r396......(4->5)-bit encd
```

cycle	loop	p5	p6	p7	p8	p9	p10	r379	r388	r395	r396	r394	r362	busy	p0	p1	p2	p3	p4
0	..L0..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.W54..	.W55.	.W58..	.W56..	.W57..	.W60..
1	.L16..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	..L3..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
2	.....	.....	.....	.....	.....	.....	.R65.	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
3	.L45..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	.L44..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	.L17..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
4	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.W68..	.W67.	.....	.....	.....	.....
5	.L14..	.R78.	.....	.....	.....	.....	.....	.....	.....	.o80..	.o76..	.o81..	.o74..	.....	.....	.....	.....	.....	.....



```
Operation name abbreviations
============================
    L0.........loop boundaries Shift_Proc_design_loop_begin
    106 ops removed w.l.
    o106a......(4_1->4)-bit ADD_UNS_OP _L0/L3/add_106


****************************************************************************
Information: control FSM status and output not registered
Control FSM has 42 states
One-Hot coding style selected, state vector is 42 bits long.
State  Code
s_0_0  1----------------------------------------
        40 state names removed w.l.
s_11_31  ----------------------------------------1

Control unit for process Shift_Proc of design srcout is hardwired
Control unit for process Shift_Proc synthesized.

Memory usage during scheduling 19097 Kbytes.
CPU usage during scheduling was 98 seconds.
Scheduled design srcout.
Warning: Overwriting design file '/afs/informatik.uni-tuebingen.de/home/wlange/hilan/aht/src_bc/srcout.db'. (DDB-24)
Current design is 'srcout'.


****************************************************************************
               Date      : Wed Sep 30 16:40:46 1998
               Version   : 1997.08
               Design    : srcout
```

```
****************************************************************************

*****************************************
*  Summary report for process Shift_Proc:  *
*****************************************
----------------------------------------------------------------------------
     Timing Summary
----------------------------------------------------------------------------
 Clock period 25.00
 Loop timing information:
    Shift_Proc.....................................33 cycles (cycles 0 - 33)
        _L0........................................32 cycles (cycles 1 - 33)
            loop_65................................2 cycles (cycles 1 - 3)
                (exit) EXIT_L65.............................. (cycle 3)
            loop_83................................2 cycles (cycles 5 - 7)
                (exit) EXIT_L83.............................. (cycle 7)
            loop_89................................2 cycles (cycles 8 - 10)
                (exit) EXIT_L89.............................. (cycle 10)
            loop_90................................2 cycles (cycles 10 - 12)
                (exit) EXIT_L90.............................. (cycle 12)
            loop_101...............................2 cycles (cycles 13 - 15)
                (exit) EXIT_L101............................. (cycle 15)
            L3.....................................16 cycles (cycles 16 - 32)
                (exit) EXIT_L106............................. (cycle 17)
                loop_110...........................2 cycles (cycles 17 - 19)
                    (exit) EXIT_L110......................... (cycle 19)
                loop_127...........................2 cycles (cycles 21 - 23)
                    (exit) EXIT_L127......................... (cycle 23)
                loop_132...........................2 cycles (cycles 25 - 27)
                    (exit) EXIT_L132......................... (cycle 27)
                loop_145...........................2 cycles (cycles 29 - 31)
                    (exit) EXIT_L145......................... (cycle 31)
----------------------------------------------------------------------------
     Area Summary
----------------------------------------------------------------------------
 Estimated combinational area    543
 Estimated sequential area       2230
 TOTAL                           2773

 42 control states
 56 basic transitions
 5 control inputs
 30 control outputs

----------------------------------------------------------------------------
     Resource types
----------------------------------------------------------------------------
Register Types
======================================
1-bit register.....................3
4-bit register.....................5
5-bit register.....................3
48-bit register....................1

Operator Types
======================================
(4->4)-bit DW01_inc................1
(4->5)-bit encd....................4
(4_4->1)-bit DW01_cmp2.............1

I/O Ports
======================================
1-bit input port...................4
1-bit registered output port.......5
32-bit input port..................2
48-bit registered output port......1

----------------------------------------------------------------------------
```

12.5 Verbindungs-Steuerung (CC)

Quellcode der VHDL-Verhaltensbeschreibung

```
-- Verhaltensbeschreibung des CC mit Multicast fuer
-- Simulation mit Synopsys.
-- cc_bc.vhd hat Interfaces zu 14 mal AHT_IN, und den Switch,
-- CC nimmt die Routing Requests der RC’s auf,
-- und veranlasst den Switch, eine Verbindung zwischen
-- Eingangs - und Ausgangsadresse zu schalten (load und configure)
-- Bis die Verbindung geschaltet ist, wird das Signal
-- Wait_CC aktiviert.  Es wird angenommen, dass jedes neue load
-- und configure die alten Verbindungen ueberladedt, sodass ein
-- reset (zum Passthru) zwischen den einzelnen Umschaltungen unnoetig ist.
-- Broadcast kann alle anderen RR’s blockieren. Variable "first" sorgt dafuer,
-- dass nach einem broadcast erst einmal die anderen RR’s an die Reihe kommen.
-- Die Prozedur connect-sw muss fuer die Synthese entfernt werden.
-- Autor: W. Lange, 19. 2. 98

Library IEEE;
```

```
USE IEEE.std_logic_1164.all;
    use IEEE.std_logic_arith.all;
--  use STD.Textio.all;
Use work.utilsCC.all;

ENTITY cc IS
  PORT (
    Clk_com    : IN STD_LOGIC;
    Reset_in   : IN STD_LOGIC;
    RR         : IN STD_LOGIC_VECTOR(0 to 13); -- Routing Request
    multicast  : IN STD_LOGIC_VECTOR(0 to 13);
    OA         : IN Address;
    xmit_bsy   : IN STD_LOGIC_VECTOR(0 to 13);
    Wait_CC    : OUT STD_LOGIC_VECTOR(0 to 13);
    free       : OUT  STD_LOGIC_VECTOR(0 to 13);
    S_IA       : OUT STD_LOGIC_VECTOR(3 downto 0);      -- zum switch
    S_OA       : OUT STD_LOGIC_VECTOR(3 downto 0);
    load       : OUT STD_LOGIC;
    configure  : OUT STD_LOGIC;
    reset      : OUT STD_LOGIC);
 END cc;

ARCHITECTURE ccar OF cc IS

 BEGIN
 ccin: PROCESS -- Der CCIN Process nimmt Routing Requests auf
   VARIABLE active_reg,belegt_reg  :  STD_LOGIC_VECTOR(0 TO 13);
   VARIABLE verbind                :  matrix;
   VARIABLE IAd,OAd,r,d            :  NATURAL;  -- d for delay for BC
   VARIABLE inp_addr, out_addr     :  STD_LOGIC_VECTOR(3 DOWNTO 0);
   VARIABLE vector_14    :  STD_LOGIC_VECTOR(0 to 13);
   CONSTANT zeros        :  STD_LOGIC_VECTOR(0 to 13) := "00000000000000";
   VARIABLE first        :  BOOLEAN;
   attribute dont_unroll : boolean;
   attribute dont_unroll of L1,L2 : label is true;


 BEGIN -- Process
    --  Reset: Setze alle Signale und Variablen zurueck ----------------------
    belegt_reg  := "00000000000000"; -- Zeigt belegte Ausgaenge
    active_reg  := "00000000000000"; -- zeigt aktive Eingaenge
    inp_addr    := "0000";
    out_addr    := "0000";
    first       := TRUE;
    d           := 0;                -- delay for BC
    Wait_CC    <=  "00000000000000";  -- 14 Waits auf '0'
    free       <=  "11111111111111";  -- 14 free auf '1'
    S_IA       <=  "0000";
    S_OA       <=  "0000";
    load       <= '0';
    configure  <= '1';
    reset      <= '1';
    -- Set Switch auf Passthru
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
    -- Warte einen Taktzyklus (mind. 7 ns)
    configure  <= '0';
    reset      <= '0';
    vector_14   := "00000000000000";
    -- Setze Verbindungsmatrix gesamt auf '0'
    For i in 0 to 13 loop  verbind(i) := "00000000000000";  end loop;
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1';

 WHILE  TRUE loop    -- main loop ---------------------------
    -- Pruefe ob  ein RR da ist.
    -- Broadcast Behandlung: haelt alle anderen Uebertragungen an ...

    IF RR(0) = '1' and  first THEN
       Wait UNTIL Clk_com'EVENT AND Clk_com = '1'; -- BC (HLS-45)(9)
       Wait_CC(0) <= '1';
       -- Warte bis alle Kanaele frei sind
       IF  (belegt_reg  = zeros)  THEN -- alle Ausgaenge frei?
          belegt_reg :=  "11111111111111";
          free       <=  "00000000000000";
          -- Setze den Switch auf Broadcast:
          reset <= '1';
          configure <= '0';
          active_reg(0)  := '1';
          Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
          -- Warte einen Takt lang (mind. aber 7 ns)
          reset <= '0';
          Wait_CC(0) <= '0';
          Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
          first := FALSE; -- sorge dafuer, dass auch andere Kanaele drankommen
       ELSE
          Wait UNTIL Clk_com'EVENT AND Clk_com = '1'; -- BC (HLS-47) (5)
       END IF;

    ELSE
    -- Suche die Kanaele ab nach einem aktiven RR
 L1: FOR i in 1 to 13 loop    -- 13 Kanaele-loop
      IF multicast(i) = '1' THEN
        r := 0;
        WHILE multicast(i) = '1' loop  -- multicast loop
```

```
          r := r + 1;
          IF RR(i) = '1'  THEN
             Wait UNTIL Clk_com'EVENT AND Clk_com = '1'; -- BC (HLS-47) (12)
             Wait_CC(i) <= '1';            -- Setze Wait aktiv --
             out_addr    := OA(i);
             OAd         := conv_to_nat(out_addr);
             inp_addr := convert(i);
             IF belegt_reg(OAd) = '0'  THEN  -- Der Ausgang ist frei
                S_IA      <= inp_addr;
                S_OA      <= out_addr;
                load      <= '1';
                configure <= '1';  --  man braeuchte nuer ein configure.
                vector_14 := zeros;
                vector_14(OAd) := '1';
                verbind(i) :=  verbind(i) or vector_14;
                belegt_reg(OAd) := '1';
                free(i)    <= '0';
                Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
                -- Warte einen Takt lang (mind. aber 7 ns)
                load       <= '0';
                configure  <= '0';
                Wait_CC(i) <= '0';
                active_reg(i) := '1';      -- nach Wait_CC = '0'
                Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
             ELSE
                Wait UNTIL Clk_com'EVENT AND Clk_com = '1'; -- BC(HLS-47)(1)
             END IF; -- End.. Der Ausgang ist frei
          ELSE
               Wait UNTIL Clk_com'EVENT AND Clk_com = '1'; -- BC(HLS-47)(6)
          END IF;   --  aktive RR's sind da
          EXIT when r = 12; -- 13 waere broadcast
          Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
        END loop;  -- End while loop
        Wait UNTIL Clk_com'EVENT AND Clk_com = '1'; -- BC(HLS-45)(3)
        configure <= '1';
        Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
        configure <= '0';
        Wait UNTIL Clk_com'EVENT AND Clk_com = '1';

      ELSE  -- multicast(i) = '0'
        IF (RR(i) = '1' and active_reg(i) = '0') THEN
           Wait_CC(i) <= '1';            -- Setze Wait aktiv --
           out_addr    := OA(i);
           OAd         := conv_to_nat(out_addr);
           inp_addr := convert(i);
           IF belegt_reg(OAd) = '0'  THEN  -- Der Ausgang ist frei
              S_IA      <= inp_addr;
              S_OA      <= out_addr;
              load      <= '1';
              configure <= '1';
              vector_14 := zeros;
              vector_14(OAd) := '1';
              verbind(i) :=  verbind(i) or vector_14;
              belegt_reg(OAd) := '1';
              free(i)    <= '0';
              Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
              -- Warte einen Takt lang (mind. aber 7 ns)
              load       <= '0';
              configure  <= '0';
              Wait_CC(i) <= '0';
              active_reg(i) := '1';      -- nach Wait_CC = '0'
              Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
           ELSE
              Wait UNTIL Clk_com'EVENT AND Clk_com = '1'; -- f. BC(HLS-47)(2)
           END IF; -- End.. Der Ausgang ist frei
        ELSE
              Wait UNTIL Clk_com'EVENT AND Clk_com = '1'; -- f. BC(HLS-47)(8)
        END IF;   -- Ein aktiver RR ist da
      END IF;    -- multicast IF Then Else
      Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
    end loop;  -- 13 Kanaele loop: Abfrage nach aktivem RR
--    first := TRUE;
    END IF;    -- broadcast IF

    -- Bei Uebertragungsende, setze belegt-reg und verb-Matrix zurueck
    -- Beginne bei der Broadcast-Abfrage
    IF  active_reg(0) ='1'   THEN  -- broadcast - Abfrage
      IF  xmit_bsy(0) = '0' THEN   -- broadcast - Abfrage
             belegt_reg  := "00000000000000";
             active_reg(0)  := '0';
             free       <=  "11111111111111";
             Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
      ELSE
         Wait UNTIL Clk_com'EVENT AND Clk_com = '1'; -- BC(HLS-47)(8)
      END IF;  -- xmit_bsy if ..
    ELSE
         Wait UNTIL Clk_com'EVENT AND Clk_com = '1'; -- BC(HLS-47)(11)
    END IF;    -- active_reg if ..

L2:   FOR m in 1 to 13 loop  -- kanaele loop
        IF  xmit_bsy(m) = '0' THEN
         -- Suche erst mal die aktiven Eingaenge
          IF active_reg(m) ='1' THEN
```

```
            active_reg(m) := '0';
            -- Loesche die Belegungenn im belegt-reg
            belegt_reg := belegt_reg and not verbind(m);
            verbind(m) := zeros;
            free(m)    <= '1';
            Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
          ELSE
            Wait UNTIL Clk_com'EVENT AND Clk_com = '1'; -- f. BC(HLS-47)(4)
          END IF; -- end active if
        ELSE
            Wait UNTIL Clk_com'EVENT AND Clk_com = '1'; -- f. BC(HLS-47)(10)
        END IF;    -- end xmit_bsy if
        Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
      END loop; -- kanaele-loop
    Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
    d := d + 1;                    -- (d:delay)  Bevorzugung der 14 Kanaele
    IF d = 4 THEN                  -- vor broadcast (RR(0)
    first := TRUE; d := 0; END IF; -- 23. 2. 98 wegen BC
--  first := TRUE;
  END loop; -- main loop
 END Process;
END ccar; -- End architecture
```

Quellcode des CC-Package

```
-------------------------------------------------
Library IEEE;
USE IEEE.std_logic_1164.all;
    use IEEE.std_logic_arith.all;

package utilsCC is

 TYPE Address IS ARRAY (Integer Range 0 to 13) OF STD_LOGIC_VECTOR(3 downto 0);
 TYPE matrix  IS ARRAY (Integer Range 0 to 13) OF STD_LOGIC_VECTOR(0 to 13);



  function convert(n: natural) return STD_LOGIC_VECTOR;
  function conv_to_nat(add : STD_LOGIC_VECTOR) return natural;

end;

-------------------------------------------------

package body utilsCC is

function convert(n : natural) return STD_LOGIC_VECTOR IS
  VARIABLE temp  : STD_LOGIC_VECTOR(3 downto 0);

  begin

    CASE n IS
WHEN 0  => temp := "0000";
WHEN 1  => temp := "0001";
WHEN 2  => temp := "0010";
        WHEN 3  => temp := "0011";
WHEN 4  => temp := "0100";
WHEN 5  => temp := "0101";
        WHEN 6  => temp := "0110";
        WHEN 7  => temp := "0111";
WHEN 8  => temp := "1000";
WHEN 9  => temp := "1001";
WHEN 10 => temp := "1010";
WHEN 11 => temp := "1011";
WHEN 12 => temp := "1100";
WHEN 13 => temp := "1101";
WHEN 14 => temp := "1110";
WHEN 15 => temp := "1111";
        WHEN OTHERS => temp := "0000";
     END CASE;

   return temp;
  end convert;


   function conv_to_nat(add : STD_LOGIC_VECTOR) return natural IS
     VARIABLE temp : Natural;
     VARIABLE address : STD_LOGIC_VECTOR(3 downto 0);

  begin
    address := add;
    CASE address IS
        WHEN "0000"  => temp := 0;
WHEN "0001"  => temp := 1;
WHEN "0010"  => temp := 2;
        WHEN "0011"  => temp := 3;
WHEN "0100"  => temp := 4;
WHEN "0101"  => temp := 5;
        WHEN "0110"  => temp := 6;
        WHEN "0111"  => temp := 7;
```

```
WHEN "1000"  => temp := 8;
WHEN "1001"  => temp := 9;
WHEN "1010"  => temp := 10;
WHEN "1011"  => temp := 11;
WHEN "1100"  => temp := 12;
WHEN "1101"  => temp := 13;
WHEN "1110"  => temp := 14;
WHEN "1111"  => temp := 15;
       WHEN OTHERS => temp := 0;
     END CASE;

   return temp;
  end conv_to_nat;

end utilsCC;

--------------------------------------------------
--------------------------------------------------
```

Quellcode des CC-Testtreibers

```
-- Testdriver testcc.vhd fuer cc.vhd
-- Simulation mit Synopsys Simulater v3.3b
-- Hier werden fuer drei RR's parallel gestellt...
-- Konflikt: RR(2) und RR(3) requesten denselben Ausgang
-- Autor: W. Lange. August 1997

library IEEE;
library synopsys;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_unsigned.all;
  use IEEE.std_logic_misc.all;
  use IEEE.std_logic_arith.all;
--  use IEEE.std_logic_components.all;
  use synopsys.attributes.all;
  use STD.Textio.all;
Use work.utilsCC.all;

Entity testcc IS END;

ARCHITECTURE test_driver OF testcc IS

COMPONENT cc
  PORT (
    Clk_com    : IN STD_LOGIC;
    Reset_in   : IN STD_LOGIC;
    RR         : IN STD_LOGIC_VECTOR(0 to 13); -- Routing Request
    multicast  : IN STD_LOGIC_VECTOR(0 to 13);
    OA         : IN Address;
    xmit_bsy   : IN STD_LOGIC_VECTOR(0 to 13);
    Wait_CC    : OUT STD_LOGIC_VECTOR(0 to 13);
    free       : OUT  STD_LOGIC_VECTOR(0 to 13);
    S_IA       : OUT STD_LOGIC_VECTOR(3 downto 0);      -- zum switch
    S_OA       : OUT STD_LOGIC_VECTOR(3 downto 0);
    load       : OUT STD_LOGIC;
    configure  : OUT STD_LOGIC;
    reset      : OUT STD_LOGIC);
END COMPONENT;

-- FOR ALL : cc  USE entity work.cc(ccar);
FOR ALL : cc  USE entity work.cc(SYN_ccar); -- Fuer synthetisiertes cc

  SIGNAL  Clk_com,Reset_in : STD_LOGIC := '0';
  SIGNAL  RR,multicast,xmit_bsy,Wait_CC,free  : STD_LOGIC_VECTOR(0 to 13);
  SIGNAL  OA    :  Address;
  SIGNAL  S_IA,S_OA :  STD_LOGIC_VECTOR(3 downto 0);
  SIGNAL  load,configure,reset  : STD_LOGIC;
  SIGNAL  GSR        : STD_LOGIC; -- Global Set Reset von Synopsys

BEGIN

 UUT: cc port map (Clk_com,Reset_in,RR,multicast,OA,xmit_bsy,Wait_CC,free,
                   S_IA,S_OA,load,configure,reset);

  -- Takt - "Prozess":

  Clk_com <= not Clk_com after 12.5 ns;

  Reset_all: Process

  BEGIN

   -- RR(0) .. (2) etc wird in anderen Prozessen zurueckgesetzt

   RR(3) <= '0';
   RR(4) <= '0';
   RR(5) <= '0';
   RR(6) <= '0';
   RR(7) <= '0';
   RR(8) <= '0';
   RR(9) <= '0';
   RR(10) <= '0';
```

```
  RR(11) <= '0';
  RR(12) <= '0';
  RR(13) <= '0';

   -- xmit_bsy(0) .. (2) etc wird in anderen Prozessen zurueckgesetzt
  xmit_bsy(3) <= '0';
  xmit_bsy(4) <= '0';
  xmit_bsy(5) <= '0';
  xmit_bsy(6) <= '0';
  xmit_bsy(7) <= '0';
  xmit_bsy(8) <= '0';
  xmit_bsy(9) <= '0';
  xmit_bsy(10) <= '0';
  xmit_bsy(11) <= '0';
  xmit_bsy(12) <= '0';
  xmit_bsy(13) <= '0';

  While TRUE loop

   WAIT until Clk_com'EVENT AND Clk_com = '1';

  end loop;

 end process;

cc_stim1: Process -- Bedient nur Kanal 1

BEGIN
-- Reset first..
    WAIT until Clk_com'EVENT AND Clk_com = '1';
    GSR <= '0';
    Reset_in <= '0';            -- Global reset
    WAIT until Clk_com'EVENT AND Clk_com = '1';      -- Warte ein wenig
    GSR <= '1';
    Reset_in <= '1';
    WAIT until Clk_com'EVENT AND Clk_com = '1';
    WAIT until Clk_com'EVENT AND Clk_com = '1';
    GSR <= '0';
    Reset_in <= '0';
  -- Setze alle Signale von RC zurueck
    multicast <= "00000000000000";
    xmit_bsy(1)  <= '0';
    RR(1)        <= '0';
    Wait until Clk_com'EVENT AND Clk_com = '1';
    OA(1) <= "0000";
    Wait until Clk_com'EVENT AND Clk_com = '1';

  WHILE TRUE loop

      xmit_bsy(1) <= '0';
                          -- neu
      Wait until Clk_com'EVENT AND Clk_com = '1';   -- neu

      RR(1) <= '1';
      OA(1) <= convert(7);
      Wait until Clk_com'EVENT AND Clk_com = '1' and free(1) = '1';
      xmit_bsy(1) <= '1';
      Wait until Clk_com'EVENT AND Clk_com = '1' and Wait_CC(1) = '1';
      Wait until Clk_com'EVENT AND Clk_com = '1' and Wait_CC(1) = '0';
      RR(1) <= '0';
      Wait until Clk_com'EVENT AND Clk_com = '1';
 --
      Wait until Clk_com'EVENT AND Clk_com = '1';
      Wait until Clk_com'EVENT AND Clk_com = '1';
 --     xmit_bsy(1) <= '0';
 --     Wait until Clk_com'EVENT AND Clk_com = '1';

 -- Warte ein wenig
  For i in 1 to 14 loop
      WAIT until Clk_com'EVENT AND Clk_com = '1';
    end loop;

 -- Bringe jetzt multicast, 3 Ausgangsadressen (OA): 5, 8, 11.

      xmit_bsy(1) <= '0';

      Wait until Clk_com'EVENT AND Clk_com = '1';     -- neu


      RR(1) <= '1';
      OA(1) <= convert(5);
      multicast(1) <= '1';
      Wait until Clk_com'EVENT AND Clk_com = '1' and free(1) = '1';
      xmit_bsy(1) <= '1';
     -- Kanal 1 mit multicast, 3 Ausgangsadressen (OA): 5, 8, 11.

      Wait until Clk_com'EVENT AND Clk_com = '1' and Wait_CC(1) = '1';
      Wait until Clk_com'EVENT AND Clk_com = '1' and Wait_CC(1) = '0';
      RR(1) <= '0';
      Wait until Clk_com'EVENT AND Clk_com = '1';

      RR(1) <= '1';
      OA(1) <= convert(8);
```

```
      Wait until Clk_com'EVENT AND Clk_com = '1' and Wait_CC(1) = '1';
      Wait until Clk_com'EVENT AND Clk_com = '1' and Wait_CC(1) = '0';
      RR(1) <= '0';
      Wait until Clk_com'EVENT AND Clk_com = '1';

      RR(1) <= '1';
      OA(1) <= convert(11);
      Wait until Clk_com'EVENT AND Clk_com = '1' and Wait_CC(1) = '1';
      Wait until Clk_com'EVENT AND Clk_com = '1' and Wait_CC(1) = '0';
      RR(1) <= '0';

      multicast(1) <= '0';
      Wait until Clk_com'EVENT AND Clk_com = '1';
      Wait until Clk_com'EVENT AND Clk_com = '1';
       xmit_bsy(1) <= '0';
     For i in 1 to 28 loop
       Wait until Clk_com'EVENT AND Clk_com = '1';
     end loop;
 end loop; -- While - true - loop

 Wait until Clk_com'EVENT AND Clk_com = '1';

End process;

cc_stim2: Process -- Bedient nur Kanal 2

BEGIN

-- Reset first..
    WAIT until Clk_com'EVENT AND Clk_com = '1';
  -- Setze alle Signale von RC zurueck
  --  multicast <= "00000000000000";
    xmit_bsy(2)  <= '0';
    RR(2)        <= '0';
    OA(2) <= "0000";
    Wait until Clk_com'EVENT AND Clk_com = '1';

  WHILE TRUE loop

      xmit_bsy(2) <= '0';
      FOR i in 1 to 3 loop
        Wait until Clk_com'EVENT AND Clk_com = '1';
      End loop;

      WAIT until Clk_com'EVENT AND Clk_com = '1';
      RR(2) <= '1';
      OA(2) <= convert(12);
      Wait until Clk_com'EVENT AND Clk_com = '1' and free(2) = '1';
      xmit_bsy(2) <= '1';
      Wait until Clk_com'EVENT AND Clk_com = '1' and Wait_CC(2) = '1';
      Wait until Clk_com'EVENT AND Clk_com = '1' and Wait_CC(2) = '0';
      RR(2) <= '0';
      Wait until Clk_com'EVENT AND Clk_com = '1';
 --
      Wait until Clk_com'EVENT AND Clk_com = '1';
      Wait until Clk_com'EVENT AND Clk_com = '1';
      xmit_bsy(2) <= '0';
      For i in 1 to 14 loop
        Wait until Clk_com'EVENT AND Clk_com = '1';
      end loop;
  end loop;
END process;



  cc_stim3: Process -- Versuch's mal mit broadcast
                    -- nur Kanal 0 wird bedient (Broadcast-Kanal)

BEGIN

-- Reset first..
    WAIT until Clk_com'EVENT AND Clk_com = '1';
  -- Setze alle Signale von RC zurueck
    xmit_bsy(0)  <= '0';
    RR(0)        <= '0';
    Wait until Clk_com'EVENT AND Clk_com = '1';
    OA(0) <= "0000";

  WHILE TRUE loop
  --   broadcast kommt etwas spaeter (warte damit..)
 For i in 1 to 56 loop         --  23. 2. 98 ( 112 geht)
      WAIT until Clk_com'EVENT AND Clk_com = '1';
 end loop;
     xmit_bsy(0) <= '0';
     Wait until Clk_com'EVENT AND Clk_com = '1';

      RR(0) <= '1';
 --     OA(0) <= convert(9);
      Wait until Clk_com'EVENT AND Clk_com = '1' and free(0) = '1';
      xmit_bsy(0) <= '1';
      Wait until Clk_com'EVENT AND Clk_com = '1' and Wait_CC(0) = '1';
      Wait until Clk_com'EVENT AND Clk_com = '1' and Wait_CC(0) = '0';
      RR(0) <= '0';
```

```
       Wait until Clk_com'EVENT AND Clk_com = '1';
       Wait until Clk_com'EVENT AND Clk_com = '1';
       Wait until Clk_com'EVENT AND Clk_com = '1';
       Wait until Clk_com'EVENT AND Clk_com = '1';
       Wait until Clk_com'EVENT AND Clk_com = '1';
       xmit_bsy(0) <= '0';
  -- Warte noch ein wenig bis zum naechsten mal
    For i in 1 to 28 loop
       WAIT until Clk_com'EVENT AND Clk_com = '1';
     end loop;

  end loop;
 end process;

END;
```

Das Synthese-Script

```
/* bc_script fuer cc_bc.vhdl  */
sh date >> laufzeit

bc_enable_analysis_info =  "true" /* For BC_View */
analyze -f vhdl utilsCC.vhd
analyze -f vhdl cc_bc.vhd
elaborate -s cc
/* reset- Setzen NACH elaborate ! */
set_behavioral_reset Reset_in -active high
create_clock Clk_com  -period 25
write -h -o cc_elab.db   /* save the elaborated design */

/* bc_check_design -io cycle_fixed */
/* register_control -inputs */
bc_check_design -io superstate_fixed
/* set_operating_conditions wccom */
sh date >> laufzeit
bc_time_design -force -fastest > cc_time.rep
write -h -o cc_timed.db    /*  save the timed design */

schedule -io superstate_fixed -effort medium  > cc_sched.rep
/* rename_design ahtin_scheduled */ /* keeps the names clean */
report_schedule -abstract > cc_fsm.rep
report_schedule -operations > cc_sched_op.rep
report_schedule -summary > cc_sched_sum.rep
report_resource_estimates
/* write -h -o cc_sched.db */       /* save the RTL design to a db file */
vhdlout_dont_write_types = "true"     /* prevents clashes in elaboration */
vhdlout_use_packages = { ieee.std_logic_1164,ieee.std_logic_arith }
vhdlout_equations = "true"       /* ignore gate delays */
vhdlout_levelize = true

/* write -hier -f vhdl -out cc_nolev_reg.vhd */ /*schreibt Komponenten raus */
write -hier -f vhdl -out cc_lev_reg.vhd  /* saves a simulatable register_level */
sh date >> laufzeit                      /*   design */

quit
```

BC-Scheduling-Reports

```
Warning: Design has already been timed. (HLS-117)
Loading db file '/afs/wsi/ti/synopsys/1997.8/libraries/syn/gtech.db'
Information: Scheduling '_L2_design' ... (HLS-152)
Information: Scheduling 'L1_design' ... (HLS-152)
Information: Scheduling 'L2_design' ... (HLS-152)
Information: Scheduling '_L1_design' ... (HLS-152)
Information: Scheduling 'ccin_design' ... (HLS-152)
Information: Allocating hardware for 'ccin_design' ... (HLS-153)
****************************************************************************
               Date      : Wed Sep 30 17:01:13 1998
               Version   : 1997.08
               Design    : cc
****************************************************************************

****************************************
*  Operation schedule of process ccin:  *
****************************************

Resource types
==================================
    OA........56-bit input port
    RR........14-bit input port
    S_IA......4-bit registered output port
    S_OA......4-bit registered output port
    free......14-bit registered output port
    load......1-bit registered output port
    loop......loop boundaries
    p0........14-bit registered output port Wait_CC
    p1........1-bit registered output port configure
    p2........1-bit registered output port reset
    p3........14-bit input port xmit_bsy
    p4........14-bit input port multicast
```

```
    r101......(4_4->1)-bit DW01_cmp2
    r110......(4->4)-bit DW01_inc
    r466......(31->31)-bit DW01_inc
```

cycle	loop	RR	p3	p4	OA	r101	r466	r110	p0	p1	free	S_IA	S_OA	load	p2
0	..L0..	.....	.....	.....	.....	.....	.....	.....	.W62..	.W67..	.W63..	.W64..	.W65..	.W66..	.W68.
1	.....	.....	.....	.....	.....	.....	.....	.....	.....	.W72..	.....	.....	.....	.....	.W73.
2	..L3..	.R83..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
3	..L8..	.R152.	.....	.R107.	.R154.	.o106.	.....	.o106a.	.R153.	.W163.	.R168.	.W160.	.W161.	.W162.	.W91.
	.....	.....	.....	.....	.....	.....	.....	.....	.R85..	.W92..	.W168.	.....	.....	.....	.....
	.....	.....	.....	.....	.....	.....	.....	.....	.W153.	.....	.W89..	.....	.....	.....	.....
	.....	.....	.....	.....	.....	.....	.....	.....	.W85..	.....	.....	.....	.....	.....	.....
4	.L14..	.R111.	.....	.R109.	.....	.....	.o110.	.....	.R173.	.W172.	.....	.....	.....	.W171.	.W96.
	.L13..	.....	.....	.....	.....	.....	.....	.....	.R97..	.....	.....	.....	.....	.....	.....
	.....	.....	.....	.....	.....	.....	.....	.....	.W173.	.....	.....	.....	.....	.....	.....
	.....	.....	.....	.....	.....	.....	.....	.....	.W97..	.....	.....	.....	.....	.....	.....
5	.L18..	.....	.....	.....	.R114.	.....	.....	.....	.R113.	.W123.	.R128.	.W120.	.W121.	.W122.	.....
	.....	.....	.....	.....	.....	.....	.....	.....	.W113.	.....	.W128.	.....	.....	.....	.....
6	.....	.....	.....	.....	.....	.....	.....	.....	.R133.	.W132.	.....	.....	.....	.W131.	.....
	.....	.....	.....	.....	.....	.....	.....	.....	.W133.	.....	.....	.....	.....	.....	.....
7	.L17..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
8	.L16..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	.L15..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
9	.....	.....	.....	.....	.....	.....	.....	.....	.....	.W146.	.....	.....	.....	.....	.....
10	.....	.....	.....	.....	.....	.....	.....	.....	.....	.W148.	.....	.....	.....	.....	.....
11	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
12	.L12..	.....	.R191.	.....	.....	.....	.....	.....	.....	.....	.W194.	.....	.....	.....	.....
	..L9..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
13	..L6..	.....	.R204.	.....	.....	.o203.	.o222.	.....	.....	.....	.R211.	.....	.....	.....	.....
	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.W211.	.....	.....	.....	.....
14	.L11..	.....	.....	.....	.....	.....	.....	.o203a.	.....	.....	.....	.....	.....	.....	.....
15	.L10..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	..L7..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
16	..L5..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	..L4..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	..L2..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	..L1..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....

```
Operation name abbreviations
============================
    L0.........loop boundaries ccin_design_loop_begin
    78 ops removed w.l.
    o203a......(4_1->4)-bit ADD_UNS_OP _L1/L2/add_203


****************************************************************************
Information: control FSM status and output not registered
Control FSM has 30 states
One-Hot coding style selected, state vector is 30 bits long.
State  Code
s_0_0  1----------------------------
       28 states removed w.l.
s_4_15  ----------------------------1

Control unit for process ccin of design cc is hardwired
Control unit for process ccin synthesized.

Memory usage during scheduling 118105 Kbytes.
CPU usage during scheduling was 873 seconds.
Scheduled design cc.
Warning: Overwriting design file '/afs/informatik.uni-tuebingen.de/home/wlange/hilan/aht/cc_bc/cc.db'. (DDB-24)
Current design is 'cc'.


****************************************************************************
              Date      : Wed Sep 30 17:11:26 1998
              Version   : 1997.08
              Design    : cc
****************************************************************************

************************************
*  Summary report for process ccin:  *
************************************
--------------------------------------------------------------------------------
     Timing Summary
--------------------------------------------------------------------------------
 Clock period 25.00
 Loop timing information:
    ccin.....................................16 cycles (cycles 0 - 16)
        _L1..................................14 cycles (cycles 2 - 16)
```

```
            L1.....................................9 cycles (cycles 3 - 12)
                (exit) EXIT_L106.......................... (cycle 4)
                _L2.............................4 cycles (cycles 4 - 8)
                    (exit) EXIT_L142...................... (cycle 7)
                    (exit) EXIT_L109...................... (cycle 5)
            L2.....................................2 cycles (cycles 13 - 15)
                (exit) EXIT_L203.......................... (cycle 14)
--------------------------------------------------------------------------------
     Area Summary
--------------------------------------------------------------------------------
 Estimated combinational area    3634
 Estimated sequential area       9420
 TOTAL                           13054

 30 control states
 42 basic transitions
 25 control inputs
 99 control outputs

--------------------------------------------------------------------------------
     Resource types
--------------------------------------------------------------------------------
Register Types
=======================================
1-bit register.....................11
4-bit register.....................3
8-bit register.....................2
14-bit register....................7
31-bit register....................2
56-bit register....................1
196-bit register...................3

Operator Types
=======================================
(4->4)-bit DW01_inc................1
(4_4->1)-bit DW01_cmp2.............1
(31->31)-bit DW01_inc..............1

I/O Ports
=======================================
1-bit registered output port.......3
4-bit registered output port.......2
14-bit input port..................3
14-bit registered output port......2
56-bit input port..................1

--------------------------------------------------------------------------------
```

12.6 Datenaufnahme-Modul RD

Quellcode der VHDL-Verhaltensbeschreibung

```
-- Verhaltensbeschreibung des rd (receive data) fuer
-- Simulation mit Synopsys.
-- rd_bc.vhd nimmt zunaechst in einem 5-Bit Schieberegister
-- die Daten aus dem Switch auf.
-- Das Interface muss gegenueber rd.vhd geaendert werden,
-- da wegen des BC auch huier das SR ausgelagert wird.
-- das SR liefert 48 bit Daten parallel.
-- Die ersten 5 bit tragen das Startzeichen, die dekodiert werden.
-- Der decoder ist ein 5B/4B  Decoder.
-- Jede Zelle beginnt mit einem Start-Of-Cell Zeichen (’11001’)
-- Jedes der 12 folgenden Zellworte (mit 48 Bit) der Payload
-- beginnt mit einem Start-Of-Payload Zeichen (’10001’) (reg-start)
-- Die letzten 3 bits des Zellworts werden ignoriert.
-- Der 5B/4B decoder prueft auf Fehler: Wenn ein unerlaubtes
-- Zeichen kommt, wird das Signal ‘‘error’’ ungleich Null
-- In diesem Fall wird das Schreiben in den FIFO-OUT Buffer untrdrueckt..
-- Nach jeder Zelluebertragung muessen wenigstens 2 Leertakte kommen
-- rd.vhd  ist in 2 Prozesse aufgeteilt:
-- RDIN nimmt die Daten (rcell_data) auf, im Unterschied zu rd.vhd
-- aus einem 5 bit Schieberegister (hilf5),
-- dekodiert sie und setzt sie in ein 32 Bit Reg (out_reg).
-- RDOUT schreibt das out_reg in das out_fifo mit rcell_rdy, rcell_bus
-- Der Takt wird geaendert zu Clk_com
-- Autor: W. Lange 9. 3. 98

Library IEEE;
USE IEEE.std_logic_1164.all;
    use IEEE.std_logic_arith.all;
--    use STD.Textio.all;

ENTITY rd IS
  PORT (Clk_com       : IN  STD_LOGIC;
        Reset       : IN  STD_LOGIC;         -- fuer BC eingefuehrt
        sr_strobe     : IN  STD_LOGIC;
        sr_data       : STD_LOGIC_VECTOR(47 DOWNTO 0);
        srdta_taken   : OUT STD_LOGIC;
        rcell_rdy     : OUT STD_LOGIC;
        rcell_fetch   : IN  STD_LOGIC;
```

```
      rbuffer_full   : IN  BOOLEAN;
      rcell_bus      : OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
END rd;
--------------------------------
Use work.utilsrd.all;

ARCHITECTURE rd_ar OF rd IS
  SIGNAL out_reg      :  STD_LOGIC_VECTOR(31 downto 0);
  SIGNAL write_fifo   :  BOOLEAN;
  SIGNAL error_h      :  STD_LOGIC_VECTOR(4 downto 0);
  SIGNAL error_pl     :  STD_LOGIC_VECTOR(4 downto 0);
  SIGNAL wr_ack       :  BOOLEAN;

BEGIN

  RDIN: PROCESS
  -- Der RDIN Process nimmt die Zelle vom Switch auf, dekodiert sie,
  -- und gibt sie an den FIFO-OUT weiter.

   VARIABLE hilf_reg     :  STD_LOGIC_VECTOR(47 downto 0);
   VARIABLE hilf5        :  STD_LOGIC_VECTOR(4 downto 0);
   VARIABLE err,err_hold :  STD_LOGIC_VECTOR(4 downto 0);
   VARIABLE hilf4        :  STD_LOGIC_VECTOR(3 downto 0);
--   VARIABLE i,k,m        :  NATURAL ;   -- i,k  fuer v_parser .. (BC)
   VARIABLE out_reg_h    :  STD_LOGIC_VECTOR(31 downto 0);
   CONSTANT start_cell   :  STD_LOGIC_VECTOR(4 downto 0):= "11001";
   CONSTANT start_payl   :  STD_LOGIC_VECTOR(4 downto 0):= "10001";
   CONSTANT header_err   :  STD_LOGIC_VECTOR(4 downto 0):= "00001";
   CONSTANT payl_err     :  STD_LOGIC_VECTOR(4 downto 0):= "00010";
   attribute dont_unroll : boolean;
   attribute dont_unroll of L3 : label is true;

-- Bitlokationen von out_reg und Zellwort
--                   31  28  27 24 23 20 19 16 15 12 11 8  7  4  3  0
-- out_reg :=          "0000  0000  0000  0000  0000  0000  0000  0000";
--              47  43 42 38 37 33 32 28 27 23 22 18 17 13 12  8 7   3 2 0
-- Zellwort  := "11001 00000 00000 00000 00000 00000 00000 00000 00000 000";


  BEGIN -- Process
  --  Reset: Setze alle Variablen und Signale zurueck ----------------------
    srdta_taken <= '0';
    hilf_reg    := "00000000000000000000000000000000000000000000000000";
    err         := "00000";
    err_hold    := "00000";
    write_fifo  <= FALSE;
    error_h     <= "00000";
    error_pl    <= "00000";
  Wait UNTIL Clk_com'EVENT AND Clk_com = '1';

  --  eine neue Zelle kommt an...
  WHILE  NOT rbuffer_full  loop    -- main loop --------------------------

    Wait UNTIL Clk_com'EVENT AND Clk_com  = '1' and sr_strobe = '1';
    -- Der Header ist da..     -----------------------------

    hilf_reg := sr_data;

    srdta_taken <= '1';

    Wait UNTIL Clk_com'EVENT AND Clk_com  = '1';
    Wait UNTIL Clk_com'EVENT AND Clk_com  = '1' and sr_strobe = '0';
    srdta_taken <= '0';

    IF hilf_reg(47 downto 43) = start_cell THEN
        -- Das Startzeichen ist da, sammle den Header auf..

    L1: FOR m in 1 to 8 loop
        CASE m IS
          WHEN 1 => hilf5 := hilf_reg(42 downto 38);
                    decode(hilf5,hilf4,err);
    out_reg_h(31 downto 28) := hilf4;
  WHEN 2 => hilf5 := hilf_reg(37 downto 33);
    decode(hilf5,hilf4,err);
                    out_reg_h(27 downto 24) := hilf4;
  WHEN 3 => hilf5 := hilf_reg(32 downto 28);
    decode(hilf5,hilf4,err);
            out_reg_h(23 downto 20) := hilf4;
     WHEN 4 => hilf5 := hilf_reg(27 downto 23);
    decode(hilf5,hilf4,err);
    out_reg_h(19 downto 16) := hilf4;
  WHEN 5 => hilf5 := hilf_reg(22 downto 18);
    decode(hilf5,hilf4,err);
    out_reg_h(15  downto 12) := hilf4;
  WHEN 6 => hilf5 := hilf_reg(17 downto 13);
    decode(hilf5,hilf4,err);
    out_reg_h(11 downto 8) := hilf4;
  WHEN 7 => hilf5 := hilf_reg(12 downto 8);
    decode(hilf5,hilf4,err);
    out_reg_h(7 downto 4) := hilf4;
  WHEN 8 => hilf5 := hilf_reg(7 downto 3);
    decode(hilf5,hilf4,err);
    out_reg_h(3 downto 0) := hilf4;
```

```
          WHEN OTHERS =>
        END CASE;
           IF err /= "00000" THEN err_hold := err; END IF;
     END loop;  -- 1 to 8 loop

--  IF err = "00000" THEN write_fifo <= TRUE; END IF; -- fuer Fehlerbehandlung
         write_fifo <= TRUE;
         error_h <= err_hold;
         out_reg <=  out_reg_h;

    ELSE error_h <= header_err;

    END IF;

      Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
      Wait UNTIL Clk_com'EVENT AND Clk_com  = '1' and wr_ack;
      write_fifo <= False;
      Wait UNTIL Clk_com'EVENT AND Clk_com = '1';

   -- Sammle die Payload auf
   L3 : FOR k in 1 to 12 loop
   Wait UNTIL Clk_com'EVENT AND Clk_com  = '1' and sr_strobe = '1';
    -- Das Zellwort ist da..   ----------------------

    hilf_reg := sr_data;

    srdta_taken <= '1';

    Wait UNTIL Clk_com'EVENT AND Clk_com  = '1';
    Wait UNTIL Clk_com'EVENT AND Clk_com  = '1' and sr_strobe = '0';
    srdta_taken <= '0';


     IF  hilf_reg(47 downto 43)  = start_payl THEN
        -- Das Startzeichen ist da, sammle das Zellwort auf..

     L2: FOR m in 1 to 8 loop
        CASE m IS
          WHEN 1 => hilf5 := hilf_reg(42 downto 38);
                    decode(hilf5,hilf4,err);
                    out_reg_h(31 downto 28) := hilf4;
          WHEN 2 => hilf5 := hilf_reg(37 downto 33);
                    decode(hilf5,hilf4,err);
                    out_reg_h(27 downto 24) := hilf4;
          WHEN 3 => hilf5 := hilf_reg(32 downto 28);
                    decode(hilf5,hilf4,err);
                    out_reg_h(23 downto 20) := hilf4;
          WHEN 4 => hilf5 := hilf_reg(27 downto 23);
                    decode(hilf5,hilf4,err);
                    out_reg_h(19 downto 16) := hilf4;
          WHEN 5 => hilf5 := hilf_reg(22 downto 18);
                    decode(hilf5,hilf4,err);
                    out_reg_h(15  downto 12) := hilf4;
          WHEN 6 => hilf5 := hilf_reg(17 downto 13);
                    decode(hilf5,hilf4,err);
                    out_reg_h(11 downto 8) := hilf4;
          WHEN 7 => hilf5 := hilf_reg(12 downto 8);
                    decode(hilf5,hilf4,err);
                    out_reg_h(7 downto 4) := hilf4;
          WHEN 8 => hilf5 := hilf_reg(7 downto 3);
                    decode(hilf5,hilf4,err);
                    out_reg_h(3 downto 0) := hilf4;
-- IF err = "00000" THEN write_fifo <= TRUE; END IF; -- Fuer Fehlerbehandlung
          WHEN OTHERS =>
        END CASE;
           IF err /= "00000" THEN err_hold := err; END IF;
     END loop;  -- 1 to 8 loop

          write_fifo <= TRUE;
          error_pl <= err_hold;
          out_reg <=  out_reg_h;

    ELSE error_pl <= payl_err;

    END IF;

     Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
     Wait UNTIL Clk_com'EVENT AND Clk_com  = '1' and wr_ack;
     write_fifo <= FALSE;
     Wait UNTIL Clk_com'EVENT AND Clk_com = '1';  -- BC(HLS-43) (1)
   End loop;   -- 1 to 12 loop

     Wait UNTIL Clk_com'EVENT AND Clk_com = '1';
     write_fifo <= FALSE;
     Wait UNTIL Clk_com'EVENT AND Clk_com = '1';  -- BC(HLS-43) (2)
   End loop;     -- Main Loop
  End Process;


  RDOUT: PROCESS
  -- Der RDOUT Process laedt das out_register in den rd-FIFO
  -- und kommuniziert mit dem rd-FIFO
  -- Achtung, Kommunikation zwischen RDIN und RDOUT
```

```
  -- ueber write_fifo ist synchron. Das kann Probleme geben, wenn
  -- rcell_fetch zu lange ausbleibt.
  -- rcell_fetch
  -- sollte innerhalb von 4 Takten nach rcell_rdy kommen.

  VARIABLE Cell_reg    : UNSIGNED(31 downto 0);  -- Reg fuer Cell Bytes

  BEGIN -- Process
  --  Reset: Setze alle Variablen und Signale zurueck ----------------------
    rcell_rdy  <= '0';
    rcell_bus <= "00000000000000000000000000000000";
    wr_ack <= FALSE;
  Wait UNTIL Clk_com'EVENT AND Clk_com  = '1';

  WHILE   NOT rbuffer_full  loop    -- main loop ---------------------------
   Wait UNTIL Clk_com'EVENT AND Clk_com  = '1' and write_fifo;
     rcell_rdy <= '1';
     rcell_bus <= out_reg;
     wr_ack    <= TRUE;
   Wait UNTIL Clk_com'EVENT AND Clk_com  = '1';   -- BC(HLS-45) (3)
   Wait UNTIL Clk_com'EVENT AND Clk_com  = '1' and  NOT write_fifo;
    wr_ack <= FALSE;
   Wait UNTIL Clk_com'EVENT AND Clk_com  = '1';
   Wait UNTIL Clk_com'EVENT AND Clk_com  = '1' and rcell_fetch = '1';
     rcell_rdy <= '0';
   Wait UNTIL Clk_com'EVENT AND Clk_com  = '1';

  END loop; -- main loop
  END Process;
 END rd_ar;
```

Quellcode des RD-Package

```
--------------------------------------------------
Library IEEE;
USE IEEE.std_logic_1164.all;
    use IEEE.std_logic_arith.all;

package utilsrd is

  procedure decode(v   : IN  STD_LOGIC_VECTOR(4 downto 0);
                   ret : OUT STD_LOGIC_VECTOR(3 downto 0);
                   err : OUT STD_LOGIC_VECTOR(4 downto 0));

end;

--------------------------------------------------

package body utilsrd is

 procedure decode( v   : IN  STD_LOGIC_VECTOR(4 downto 0);
                   ret : OUT STD_LOGIC_VECTOR(3 downto 0);
                   err : OUT STD_LOGIC_VECTOR(4 downto 0)) IS

  VARIABLE vector : STD_LOGIC_VECTOR(4 downto 0);

  begin
    vector := v;
    err := "00000";
    CASE vector IS
WHEN  "11110" => ret :=  "0000";
WHEN  "01001" => ret :=  "0001";
        WHEN  "10100" => ret :=  "0010";
WHEN  "10101" => ret :=  "0011";
WHEN  "01010" => ret :=  "0100";
        WHEN  "01011" => ret :=  "0101";
        WHEN  "01110" => ret :=  "0110";
WHEN  "01111" => ret :=  "0111";
WHEN  "10010" => ret :=  "1000";
WHEN  "10011" => ret :=  "1001";
WHEN  "10110" => ret :=  "1010";
WHEN  "10111" => ret :=  "1011";
WHEN  "11010" => ret :=  "1100";
WHEN  "11011" => ret :=  "1101";
WHEN  "11100" => ret :=  "1110" ;
WHEN  "11101" => ret :=  "1111";
WHEN OTHERS   => err :=    v; -- Probleme mit v_parser
     END CASE;

  end decode;
end utilsrd;

--------------------------------------------------
--------------------------------------------------
```

Quellcode des RD-Testtreibers

```
-- Testdriver testrd_bc.vhd  fuer rd_bc.vhd
-- Simulation mit Synopsys Simulater 1997.08
```

```
-- Aenderung gegenueber testrd.vhd:
-- Wir uebergeben Daten aus einem Schieberegister an rd_bc.vhd
-- Autor: W. Lange. Maerz 1998

library IEEE;
library synopsys;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_unsigned.all;
  use IEEE.std_logic_misc.all;
  use IEEE.std_logic_arith.all;
  use synopsys.attributes.all;
  use STD.Textio.all;

Entity testrd IS END;

Use work.utils_src.all;

ARCHITECTURE test_driver OF testrd IS


COMPONENT rd
  PORT (Clk_com       : IN  STD_LOGIC;
        Reset         : IN  STD_LOGIC;
        sr_strobe     : IN  STD_LOGIC;
        sr_data       : STD_LOGIC_VECTOR(47 DOWNTO 0);
        srdta_taken   : OUT STD_LOGIC;
        rcell_rdy     : OUT STD_LOGIC;
        rcell_fetch   : IN  STD_LOGIC;
        rbuffer_full  : IN  BOOLEAN;
        rcell_bus     : OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
END COMPONENT;

-- FOR ALL : rd  USE entity work.rd(rd_ar); -- fuer funktionale Simulation
FOR ALL : rd  USE entity work.rd(SYN_rd_ar); -- fuer Sim. nach der Synthese

  SIGNAL Clk_com    : STD_LOGIC := '0';
  SIGNAL sr_strobe,rcell_rdy,rcell_fetch,GSR   : STD_LOGIC;
  SIGNAL rcell_bus  : STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL rbuffer_full      : BOOLEAN;
  SIGNAL sr_data     : STD_LOGIC_VECTOR(47 downto 0);
  SIGNAL srdta_taken,Reset : STD_LOGIC;
  SIGNAL Cell_start  : STD_LOGIC;  -- interne Anzeige

BEGIN

 UUT: rd port map (Clk_com,Reset,sr_strobe,sr_data,srdta_taken,
      rcell_rdy,rcell_fetch,rbuffer_full,rcell_bus);

  Clk_com <= not Clk_com after 12.5 ns;   -- fuer clk_com, 40 Mhz


 Send_cell_data: Process
 VARIABLE hilf5 : STD_LOGIC_VECTOR(4 downto 0);
 VARIABLE hilf4 : STD_LOGIC_VECTOR(3 downto 0);
 VARIABLE hilf04: STD_LOGIC_VECTOR(3 downto 0);
 VARIABLE hilf_reg : STD_LOGIC_VECTOR(47 downto 0);
 VARIABLE p    : natural;
 CONSTANT start_c  : STD_LOGIC_VECTOR(4 downto 0):= "11001";
 CONSTANT start_payl  : STD_LOGIC_VECTOR(4 downto 0):= "10001";

 BEGIN

 -- Reset first..
     GSR <= '0';            -- Global reset
     Reset <= '0';
     rbuffer_full <= false;
     hilf5 := "00000";
     hilf4 := "0000";
     Cell_start <= '0';
     Wait until Clk_com'EVENT AND Clk_com = '1';

     GSR <= '1';
     Reset <= '1';
     Wait until Clk_com'EVENT AND Clk_com = '1';
     GSR <= '0';
     Reset <= '0';
     sr_data  <= "000000000000000000000000000000000000000000000000";
     hilf_reg := "000000000000000000000000000000000000000000000000";
     Wait until Clk_com'EVENT AND Clk_com = '1';

-- Uebertrage den Header (Header-Reg., 32 Bit -----
-- Bitlokationen von out_reg und Zellwort
--                    31  28  27 24 23 20 19 16 15 12 11 8  7  4  3  0
-- out_reg :=          "0000  0000  0000  0000  0000  0000  0000  0000";
--              47  43 42 38 37 33 32 28 27 23 22 18 17 13 12  8 7   3 2 0
-- Zellwort  := "11001 00000 00000 00000 00000 00000 00000 00000 00000 000";


 WHILE TRUE loop     -- main loop

 -- Bringe Zell-Daten mit header:

    hilf_reg(47 downto 43) := start_c;
```

```
   hilf4 := "0000";

    FOR i in 1 to 8 loop
     hilf4 := hilf4 + "0001";
     hilf5 := encd(hilf4);
       CASE i IS
         WHEN 1 => hilf_reg(42 downto 38) := hilf5;
         WHEN 2 => hilf_reg(37 downto 33) := hilf5;
         WHEN 3 => hilf_reg(32 downto 28) := hilf5;
         WHEN 4 => hilf_reg(27 downto 23) := hilf5;
         WHEN 5 => hilf_reg(22 downto 18) := hilf5;
         WHEN 6 => hilf_reg(17 downto 13) := hilf5;
         WHEN 7 => hilf_reg(12 downto 8)  := hilf5;
         WHEN 8 => hilf_reg(7 downto 3)   := hilf5;
        END CASE;
    END loop;  -- 1 to 8 loop
    -- Schicke den trailer (3 Nullen)
    hilf_reg(2 downto 0)   := "000";

      sr_data <= hilf_reg;
      sr_strobe <= '1';
      Cell_start <= '1';  -- nur eine interne Anzeige
    Wait until Clk_com'EVENT AND Clk_com = '1' and srdta_taken ='1';
       sr_strobe <= '0';
      Cell_start <= '0';
  -- Verzoegere 4 Takte, das Schiebe-Register bekommt die Payload
   For i in 1 to 4 loop
     Wait until Clk_com'EVENT AND Clk_com = '1';
   end loop;


  -- Bringe die 12 Payload Worte:
  -- erst das payload - Sonderzeichen

  Wait until Clk_com'EVENT AND Clk_com = '1';

  FOR k in 1 to 12 loop
    hilf5 := start_payl;
    -- Bringe die 12 Payload Worte:
    hilf_reg(47 downto 43) := hilf5;

    FOR i in 1 to 8 loop
     hilf4 := hilf4 + "0001";
     hilf5 := encd(hilf4);
       CASE i IS
         WHEN 1 => hilf_reg(42 downto 38) := hilf5;
         WHEN 2 => hilf_reg(37 downto 33) := hilf5;
         WHEN 3 => hilf_reg(32 downto 28) := hilf5;
         WHEN 4 => hilf_reg(27 downto 23) := hilf5;
         WHEN 5 => hilf_reg(22 downto 18) := hilf5;
         WHEN 6 => hilf_reg(17 downto 13) := hilf5;
         WHEN 7 => hilf_reg(12 downto 8)  := hilf5;
         WHEN 8 => hilf_reg(7 downto 3)   := hilf5;
        END CASE;
    END loop;  -- 1 to 8 loop
    -- Schicke den trailer (3 Nullen)
    hilf_reg(2 downto 0)   := "000";

      sr_data <= hilf_reg;
      sr_strobe <= '1';
    Wait until Clk_com'EVENT AND Clk_com = '1' and srdta_taken ='1';
       sr_strobe <= '0';
    For i in 1 to 4 loop
      Wait until Clk_com'EVENT AND Clk_com = '1';  -- Warte  4 Takte
    end loop;                                      -- bis das neue Zellwort kommt
  end loop;  -- 12 Payload Worte

    For i in 1 to 8 loop
      Wait until Clk_com'EVENT AND Clk_com = '1';  -- Warte 8 Takte
    end loop;                                      -- bis die neue Zelle kommt

 END loop; -- main loop;
 END Process;


SIMULATE_FIFO: Process

BEGIN

-- Reset first..
 rcell_fetch <= '0';
 FOR i in 1 to 4 loop
   Wait until Clk_com'EVENT AND Clk_com = '1';
 end loop;

WHILE TRUE loop

-- Warte bis ein rcell_rdy kommt:

   Wait until Clk_com'EVENT AND Clk_com = '1' and rcell_rdy ='1';
--   Wait until Clk_com'EVENT AND Clk_com = '1';
     rcell_fetch <= '1';
   Wait until Clk_com'EVENT AND Clk_com = '1' and rcell_rdy ='0';
```

```
     rcell_fetch <= '0';

 --    FOR i in 1 to 5 loop
     Wait until Clk_com'EVENT AND Clk_com = '1';
 --  End loop;


  END loop; -- main loop;
  END Process;
END;
```

Das Synthese-Script

```
/* bc_script fuer rd_bc.vhdl  */
sh date >> laufzeit

bc_enable_analysis_info =  "true" /* For BC_View */
analyze -f vhdl utilsrd.vhd
analyze -f vhdl rd_bc.vhd
elaborate -s rd

/* reset- Setzen NACH elaborate ! */
set_behavioral_reset Reset -active high
sh_date >> laufzeit
create_clock Clk_com  -period 25
write -h -o rd_elab.db   /* save the elaborated design */

/* bc_check_design -io cycle_fixed */

bc_check_design -io superstate_fixed
/* set_operating_conditions wccom */
sh date >> laufzeit
bc_time_design -fastest > rd_time.rep
write -h -o rd_timed.db    /*  save the timed design */
sh date >> laufzeit
schedule -io superstate_fixed -effort medium   > rd_sched.rep
/* rename_design ahtin_scheduled */ /* keeps the names clean */
report_schedule -abstract > rd_fsm.rep
/* report_schedule -operations ** schon in rd_sched enthalten  */
report_schedule -summary > rd_sched_sum.rep

/* write -h -o ahtin_cf.db */       /* save the RTL design to a db file */
vhdlout_dont_write_types = "true"     /* prevents clashes in elaboration */
vhdlout_use_packages = { ieee.std_logic_1164,ieee.std_logic_arith }
vhdlout_equations = "true"       /* ignore gate delays */
vhdlout_levelize = true

/* write -hier -f vhdl -out rd_nolev_reg.vhd */ /*schreibt Komponenten raus */
write -hier -f vhdl -out rd_lev_reg.vhd  /* saves a simulatable register_level */
sh date >> laufzeit                     /*   design */

quit
```

BC-Scheduling-Reports

```
Warning: Design has already been timed. (HLS-117)
Loading db file '/afs/wsi/ti/synopsys/1997.8/libraries/syn/gtech.db'
Information: Scheduling 'loop_144_design' ... (HLS-152)
Information: Scheduling 'loop_102_design' ... (HLS-152)
Information: Scheduling 'loop_158_design' ... (HLS-152)
Information: Scheduling 'loop_200_design' ... (HLS-152)
Information: Scheduling 'loop_150_design' ... (HLS-152)
Information: Scheduling 'L3_design' ... (HLS-152)
Information: Scheduling 'loop_94_design' ... (HLS-152)
Information: Scheduling '_L0_design' ... (HLS-152)
Information: Scheduling 'RDIN_design' ... (HLS-152)
Information: Allocating hardware for 'RDIN_design' ... (HLS-153)
****************************************************************************
               Date      : Wed Sep 30 19:33:08 1998
               Version   : 1997.08
               Design    : rd
****************************************************************************

****************************************
*  Operation schedule of process RDIN:  *
****************************************

Resource types
==================================
    loop......loop boundaries
    p0........1-bit registered output port srdta_taken
    p1........1-bit registered output port write_fifo
    p2........5-bit registered output port error_h
    p3........5-bit registered output port error_pl
    p4........32-bit registered output port out_reg
    p5........1-bit input port rbuffer_full
    p6........48-bit input port sr_data
    p7........1-bit input port wr_ack
```

```
    p8........1-bit input port sr_strobe
    r658......(4_4->1)-bit DW01_cmp2
    r667......(4->4)-bit DW01_inc
```

cycle	loop	p5	p6	p7	p8	r658	r667	p0	p1	p2	p3	p4
0	..L0..	.....	.....	.....	.....	.....	.....	.W82..	.W86..	.W87..	.W88..	.....
1	..L3..	.R92.	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
2	.L13..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	..L6..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
3	.....	.....	.....	.....	.R94..	.....	.....	.....	.....	.....	.....	.....
4	.L34..	.....	.R97..	.....	.....	.....	.....	.W99..	.....	.....	.....	.....
	.L33..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	.L14..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
5	.L11..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
6	.....	.....	.....	.....	.R102.	.....	.....	.....	.....	.....	.....	.....
7	.L32..	.....	.....	.....	.....	.....	.....	.W103.	.W135.	.W136.	.....	.W137.
	.L31..	.....	.....	.....	.....	.....	.....	.....	.....	.W139.	.....	.....
	.L12..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
8	..L9..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
9	.....	.....	.....	.R144.	.....	.....	.....	.....	.....	.....	.....	.....
10	.L30..	.....	.....	.....	.....	.....	.....	.....	.W145.	.....	.....	.....
	.L29..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	.L10..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
11	..L7..	.....	.....	.....	.....	.o149.	.....	.....	.....	.....	.....	.....
12	.L21..	.....	.....	.....	.....	.....	.o149a.	.....	.....	.....	.....	.....
	.L16..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
13	.....	.....	.....	.....	.R150.	.....	.....	.....	.....	.....	.....	.....
14	.L28..	.....	.R153.	.....	.....	.....	.....	.W155.	.....	.....	.....	.....
	.L27..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	.L22..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
15	.L19..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
16	.....	.....	.....	.....	.R158.	.....	.....	.....	.....	.....	.....	.....
17	.L26..	.....	.....	.....	.....	.....	.....	.W159.	.W191.	.....	.W192.	.W193.
	.L25..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.W195.	.....
	.L20..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
18	.L17..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
19	.....	.....	.....	.R200.	.....	.....	.....	.....	.....	.....	.....	.....
20	.L24..	.....	.....	.....	.....	.....	.....	.....	.W201.	.....	.....	.....
	.L23..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	.L18..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
21	.L15..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	..L8..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
22	..L5..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	..L4..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	..L2..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	..L1..	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....

```
Operation name abbreviations
============================
    L0.........loop boundaries RDIN_design_loop_begin
    62 ops removed w.l.
    o149a......(4_1->4)-bit ADD_UNS_OP _L0/L3/add_149


*************************************************************************
Information: control FSM status and output not registered
Control FSM has 45 states
One-Hot coding style selected, state vector is 45 bits long.
State  Code
s_0_0  1-------------------------------------------
         43 states removed w.l.
s_8_20  -------------------------------------------1

Control unit for process RDIN of design rd is hardwired
Control unit for process RDIN synthesized.
Information: Scheduling 'loop_239_design' ... (HLS-152)
Information: Scheduling 'loop_231_design' ... (HLS-152)
Information: Scheduling 'loop_236_design' ... (HLS-152)
Information: Scheduling '_L1_design' ... (HLS-152)
Information: Scheduling 'RDOUT_design' ... (HLS-152)
Information: Allocating hardware for 'RDOUT_design' ... (HLS-153)
*************************************************************************
               Date      : Wed Sep 30 19:33:31 1998
               Version   : 1997.08
               Design    : rd
*************************************************************************

****************************************
*  Operation schedule of process RDOUT:  *
```

```
****************************************

Resource types
===================================
    loop......loop boundaries
    p0........1-bit registered output port rcell_rdy
    p1........1-bit registered output port wr_ack
    p2........32-bit registered output port rcell_bus
    p3........1-bit input port rbuffer_full
    p4........32-bit input port out_reg
    p5........1-bit input port rcell_fetch
    p6........1-bit input port write_fifo



                  p      p      p      p      p      p      p
                  o      o      o      o      o      o      o
                  r      r      r      r      r      r      r
                  t      t      t      t      t      t      t
-------+------+------+------+------+------+------+------+------
 cycle | loop |  p3  |  p4  |  p5  |  p6  |  p0  |  p1  |  p2
-----------------------------------------------------------------
   0   |..L0..|......|......|......|......|.W225.|.W227.|.W226.
   1   |..L3..|.R230.|......|......|......|......|......|......
   2   |.L11..|......|......|......|......|......|......|......
       |..L6..|......|......|......|......|......|......|......
   3   |......|......|......|......|.R231.|......|......|......
   4   |.L18..|......|.R233.|......|......|.W232.|.W234.|.W233.
       |.L17..|......|......|......|......|......|......|......
       |.L12..|......|......|......|......|......|......|......
   5   |..L9..|......|......|......|......|......|......|......
   6   |......|......|......|......|.R236.|......|......|......
   7   |.L16..|......|......|......|......|......|.W237.|......
       |.L15..|......|......|......|......|......|......|......
       |.L10..|......|......|......|......|......|......|......
   8   |..L7..|......|......|......|......|......|......|......
   9   |......|......|......|.R239.|......|......|......|......
  10   |.L14..|......|......|......|......|.W240.|......|......
       |.L13..|......|......|......|......|......|......|......
       |..L8..|......|......|......|......|......|......|......
  11   |..L5..|......|......|......|......|......|......|......
       |..L4..|......|......|......|......|......|......|......
       |..L2..|......|......|......|......|......|......|......
       |..L1..|......|......|......|......|......|......|......


Operation name abbreviations
============================
    L0........loop boundaries RDOUT_design_loop_begin
    L1........loop boundaries RDOUT_design_loop_end
    L2........loop boundaries RDOUT_design_loop_cont
    L3........loop boundaries _L1/_L1_design_loop_begin
    L4........loop boundaries _L1/_L1_design_loop_end
    L5........loop boundaries _L1/_L1_design_loop_cont
    L6........loop boundaries _L1/EXIT_L230
    L7........loop boundaries _L1/loop_239/loop_239_design_loop_begin
    L8........loop boundaries _L1/loop_239/loop_239_design_loop_end
    L9........loop boundaries _L1/loop_236/loop_236_design_loop_begin
    L10.......loop boundaries _L1/loop_236/loop_236_design_loop_end
    L11.......loop boundaries _L1/loop_231/loop_231_design_loop_begin
    L12.......loop boundaries _L1/loop_231/loop_231_design_loop_end
    L13.......loop boundaries _L1/loop_239/loop_239_design_loop_cont
    L14.......loop boundaries _L1/loop_239/EXIT_L239
    L15.......loop boundaries _L1/loop_236/loop_236_design_loop_cont
    L16.......loop boundaries _L1/loop_236/EXIT_L236
    L17.......loop boundaries _L1/loop_231/loop_231_design_loop_cont
    L18.......loop boundaries _L1/loop_231/EXIT_L231
    R230......1-bit read _L1/rbuffer_full_230
    R231......1-bit read _L1/loop_231/write_fifo_231
    R233......32-bit read _L1/out_reg_233
    R236......1-bit read _L1/loop_236/write_fifo_236
    R239......1-bit read _L1/loop_239/rcell_fetch_239
    W225......1-bit write rcell_rdy_225
    W226......32-bit write rcell_bus_226
    W227......1-bit write wr_ack_227
    W232......1-bit write _L1/rcell_rdy_232
    W233......32-bit write _L1/rcell_bus_233
    W234......1-bit write _L1/wr_ack_234
    W237......1-bit write _L1/wr_ack_237
    W240......1-bit write _L1/rcell_rdy_240


************************************************************************
Information: control FSM status and output not registered
Control FSM has 18 states
One-Hot coding style selected, state vector is 18 bits long.
State  Code
s_0_0  1-----------------
s_0_1  -1----------------
s_1_2  --1---------------
s_1_3  ---1--------------
s_1_4  ----1-------------
s_1_5  -----1------------
```

```
s_1_6  ------1-----------
s_1_7  -------1----------
s_1_8  --------1---------
s_1_9  ---------1--------
s_1_10  ----------1-------
s_1_11  -----------1------
s_2_3  ------------1-----
s_2_4  -------------1----
s_3_6  --------------1---
s_3_7  ---------------1--
s_4_9  ----------------1-
s_4_10  -----------------1

Control unit for process RDOUT of design rd is hardwired
Control unit for process RDOUT synthesized.

Memory usage during scheduling 603641 Kbytes.
CPU usage during scheduling was 7258 seconds.
Scheduled design rd.
Warning: Overwriting design file '/afs/informatik.uni-tuebingen.de/home/wlange/hilan/aht/rd_bc/rd.db'. (DDB-24)
Current design is 'rd'.


****************************************************************************
               Date      : Thu Oct  1 10:35:57 1998
               Version   : 1997.08
               Design    : rd
****************************************************************************

************************************
*  Summary report for process RDIN:  *
************************************
----------------------------------------------------------------------------
     Timing Summary
----------------------------------------------------------------------------
 Clock period 25.00
 Loop timing information:
    RDIN.....................................22 cycles (cycles 0 - 22)
        _L0..................................21 cycles (cycles 1 - 22)
            (exit) EXIT_L92.............................. (cycle 2)
            loop_94..............................2 cycles (cycles 2 - 4)
                (exit) EXIT_L94.......................... (cycle 4)
            loop_102.............................2 cycles (cycles 5 - 7)
                (exit) EXIT_L102......................... (cycle 7)
            loop_144.............................2 cycles (cycles 8 - 10)
                (exit) EXIT_L144......................... (cycle 10)
            L3...................................10 cycles (cycles 11 - 21)
                (exit) EXIT_L149......................... (cycle 12)
                loop_150.........................2 cycles (cycles 12 - 14)
                    (exit) EXIT_L150..................... (cycle 14)
                loop_158.........................2 cycles (cycles 15 - 17)
                    (exit) EXIT_L158..................... (cycle 17)
                loop_200.........................2 cycles (cycles 18 - 20)
                    (exit) EXIT_L200..................... (cycle 20)
----------------------------------------------------------------------------
     Area Summary
----------------------------------------------------------------------------
 Estimated combinational area    319
 Estimated sequential area       3110
 TOTAL                           3429

 45 control states
 60 basic transitions
 239 control inputs
 306 control outputs

----------------------------------------------------------------------------
     Resource types
----------------------------------------------------------------------------
Register Types
======================================
1-bit register.....................116
4-bit register.....................2
5-bit register.....................3
48-bit register....................1

Operator Types
======================================
(4->4)-bit DW01_inc................1
(4_4->1)-bit DW01_cmp2.............1

I/O Ports
======================================
1-bit input port...................3
1-bit registered output port.......2
5-bit registered output port.......2
32-bit registered output port......1
48-bit input port..................1

----------------------------------------------------------------------------
************************************
*  Summary report for process RDOUT:  *
************************************
----------------------------------------------------------------------------
```

```
     Timing Summary
--------------------------------------------------------------------------------
 Clock period 25.00
 Loop timing information:
    RDOUT.....................................11 cycles (cycles 0 - 11)
        _L1.....................................10 cycles (cycles 1 - 11)
            (exit) EXIT_L230.............................. (cycle 2)
            loop_231.............................2 cycles (cycles 2 - 4)
                (exit) EXIT_L231.......................... (cycle 4)
            loop_236.............................2 cycles (cycles 5 - 7)
                (exit) EXIT_L236.......................... (cycle 7)
            loop_239.............................2 cycles (cycles 8 - 10)
                (exit) EXIT_L239.......................... (cycle 10)
--------------------------------------------------------------------------------
     Area Summary
--------------------------------------------------------------------------------
 Estimated combinational area    46
 Estimated sequential area       2930
 TOTAL                           2976

 18 control states
 25 basic transitions
 3 control inputs
 8 control outputs

--------------------------------------------------------------------------------
     Resource types
--------------------------------------------------------------------------------
Register Types
=======================================
1-bit register.....................2

Operator Types
=======================================

I/O Ports
=======================================
1-bit input port...................3
1-bit registered output port.......2
32-bit input port..................1
32-bit registered output port......1

--------------------------------------------------------------------------------
```

12.7 AHT-Ausgangs-Modul

Quellcode der VHDL-Verhaltensbeschreibung mit Schleifen

```
-- Verhaltensbeschreibung des aht_out fuer
-- Simulation mit Synopsys. Compilierung mit Synopsys analyze/elaborate
-- Synthese mit BC
-- aht_out besteht aus 2 Prozessen:
-- AHTOUT holt Zell-Daten aus dem FIFO_out-Speicher und
-- und gibt sie an den PROCESS AHTOUT_Transmit weiter
-- AHTOUT_Transmit gibt die Daten in 8-Bit Paketen (an die PHYS-Schicht)
-- auf die Leitung.
-- Es werden nur 52 Byte uebertragen (4 Byte fuer den Header, ohne HEC.
-- Es wird davon ausgegengen, dass der HEC in der PHYS-Schicht
-- eingefuegt wird.
-- Aenderungen: 1. reset wird eingefuegt
-- 2. 'Dont unroll' fuer die loops L1,L2,L3
-- Autor: W. Lange Maerz 1998


Library IEEE;
USE IEEE.std_logic_1164.all;
    use IEEE.std_logic_arith.all;
--  use STD.Textio.all;

ENTITY ahtout IS
  PORT (Clk_aht_out    : IN  STD_LOGIC;
        Cell_Sync_out  : OUT STD_LOGIC;
        Cell_preSync   : OUT STD_LOGIC;
        Chan_bsy       : IN  STD_LOGIC;
        Reset          : IN STD_LOGIC;
        Data_out       : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        rnew_cell      : OUT STD_LOGIC;
        rcell_data     : IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
        rcell_read     : OUT STD_LOGIC;
        rcell_write    : IN  STD_LOGIC);
END ahtout;
---------------------------------

ARCHITECTURE ahtoutar OF ahtout IS
  SIGNAL transmit      : BOOLEAN;
  SIGNAL transmit_new  : BOOLEAN;
  SIGNAL reg_taken     : BOOLEAN;
  SIGNAL cell_reg      : STD_LOGIC_VECTOR(31 downto 0);  --Reg fuer Cellwort
```

```
BEGIN

  AHTOUT: PROCESS
  -- Der AHTOUT Process holt Zell-Daten aus dem FIFO_out-Speicher und
  -- und gibt sie an den Process AHTOUT_TRANSMIT weiter
   VARIABLE i,k       :  NATURAL;
   attribute dont_unroll : boolean;
   attribute dont_unroll of L1 : label is true;

  BEGIN -- Process
  --  Reset: Setze alle Variablen und Signale zurueck ----------------------
    rcell_read   <= '0';
    rnew_cell    <= '0';
    transmit     <= FALSE;
    transmit_new <= FALSE;
   Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';

   WHILE  TRUE  loop    -- main loop ---------------------------

     Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and Chan_bsy = '0';
     -- Der Ausgangskanal ist frei. Hole eine Zelle aus dem FIFO-Speicher.
     -- Wiederhole 13 mal fuer 32 - Bit-Register (52 Byte)
      rnew_cell <= '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';  -- BC
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write='1';
      cell_reg <= rcell_data; -- Der Header
      rnew_cell  <= '0';
      transmit_new <= TRUE;
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and reg_taken; -- BC01
      transmit_new <= FALSE;
--    Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write ='0';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   --BC02(HLS43)

     L1: FOR i in 1 to 12 loop
      rcell_read <= '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   --BC03(HLS45)
      -- Warte auf die Antwort des Speicher-Mgmt's
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write='1';
      cell_reg <= rcell_data;
      rcell_read <= '0';
      transmit <= TRUE;
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   -- BC
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and reg_taken; --BC01
      transmit <= FALSE;
--    Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write ='0';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';     --BC02(HLS43)
     End loop;
  End loop;  -- main loop

  End Process;


  AHTOUT_Transmit: PROCESS
     VARIABLE hilf         : STD_LOGIC_VECTOR(31 downto 0);
     attribute dont_unroll : boolean;
     attribute dont_unroll of L2 : label is true;


  BEGIN   -- Process
    -- Reset Part
    reg_taken <= FALSE;
    Cell_Sync_out  <= '0';
    Cell_preSync   <= '0';
    Data_out       <= "00000000";
    Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';

    WHILE Chan_bsy  = '0' loop  -- main loop

     Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and transmit_new;
     Cell_preSync <= '1';
     Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
     Cell_Sync_out <= '1';

     hilf := cell_reg;    -- Header
     reg_taken <= TRUE;

data_out <= hilf(31 downto 24);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(23 downto 16);
        reg_taken <= FALSE;
        Cell_Sync_out <= '0';
        Cell_preSync <= '0';
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(15 downto 8);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(7 downto 0);
        -- Das HEC Byte bleibt unberuecksichtigt.
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1'; -- BC

     -- Transmit Bytewise die Payload

     L2: FOR k IN 1 TO 12 loop  -- 12 * 32 Bit register
```

```
       Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1'and transmit;
       hilf := cell_reg;
       reg_taken <= TRUE;

        -- 4 * 8 Bit = 32 Bit gehen auf die Leitung

data_out <= hilf(31 downto 24);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(23 downto 16);
        reg_taken <= FALSE;
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(15 downto 8);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(7 downto 0);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';  -- BC

    end loop; -- 1 to 12 loop

    Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';

   End loop; -- Main Loop
  End Process;

 END ahtoutar;
```

Quellcode der VHDL-Verhaltensbeschreibung ohne Schleifen

```
-- Verhaltensbeschreibung des aht_out fuer
-- Simulation mit Synopsys. Compilierung mit Synopsys analyze/elaborate
-- Synthese mit BC
-- ahaout_bcnew besteht aus 2 Prozessen:
-- AHTOUT holt Zell-Daten aus dem FIFO_out-Speicher und
-- und gibt sie an den PROCESS AHTOUT_Transmit weiter
-- AHTOUT_Transmit gibt die Daten in 8-Bit Paketen (an die PHYS-Schicht)
-- auf die Leitung.
-- Es werden nur 52 Byte uebertragen (4 Byte fuer den Header, ohne HEC.
-- Es wird davon ausgegengen, dass der HEC in der PHYS-Schicht
-- eingefuegt wird.
-- Aenderungen: 1. reset wird eingefuegt
-- Alle loops sind von Hand aufgerollt
-- Arbeitet praktisch schon wie 'cycle fixed'
-- Autor: W. Lange Maerz 1998


Library IEEE;
USE IEEE.std_logic_1164.all;
    use IEEE.std_logic_arith.all;
--  use STD.Textio.all;

ENTITY ahtout IS
  PORT (Clk_aht_out    : IN  STD_LOGIC;
        Cell_Sync_out  : OUT STD_LOGIC;
        Cell_preSync   : OUT STD_LOGIC;
        Chan_bsy       : IN  STD_LOGIC;
        Reset          : IN STD_LOGIC;
        Data_out       : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        rnew_cell      : OUT STD_LOGIC;
        rcell_data     : IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
        rcell_read     : OUT STD_LOGIC;
        rcell_write    : IN  STD_LOGIC);
END ahtout;
---------------------------------

ARCHITECTURE ahtoutar OF ahtout IS
  SIGNAL transmit      : BOOLEAN;
  SIGNAL transmit_new  : BOOLEAN;
  SIGNAL reg_taken     : BOOLEAN;
  SIGNAL cell_reg      : STD_LOGIC_VECTOR(31 downto 0);  --Reg fuer Cellwort
  SIGNAL cell_reg2     : STD_LOGIC_VECTOR(31 downto 0);  --Reg fuer Cellwort

BEGIN

  AHTOUT: PROCESS
  -- Der AHTOUT Process holt Zell-Daten aus dem FIFO_out-Speicher und
  -- und gibt sie an den Process AHTOUT_TRANSMIT weiter
   VARIABLE i,k       :  NATURAL;
--   attribute dont_unroll : boolean;
--   attribute dont_unroll of L1 : label is true;


  BEGIN -- Process
  --  Reset: Setze alle Variablen und Signale zurueck ----------------------
    rcell_read   <= '0';
    rnew_cell    <= '0';
    transmit     <= FALSE;
    transmit_new <= FALSE;
   Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';

   WHILE  TRUE  loop    -- main loop --------------------------

     Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and Chan_bsy = '0';
```

```
     Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
     -- Der Ausgangskanal ist frei. Hole eine Zelle aus dem FIFO-Speicher.
     -- Wiederhole 13 mal fuer 32 - Bit-Register (52 Byte)
      rnew_cell <= '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';    -- BC(HLS45)
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write='1';
      cell_reg <= rcell_data; -- Der Header    (1)
      rnew_cell  <= '0';
      transmit_new <= TRUE;
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
--      transmit_new <= FALSE; verlegt nach unten
--      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   --BC02(HLS43)

      rcell_read <= '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   --BC03(HLS45)
      -- Warte auf die Antwort des Speicher-Mgmt's
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write='1';
      cell_reg2 <= rcell_data;   --       (2)
      transmit_new <= false;
      rcell_read <= '0';
      transmit <= TRUE;
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
      transmit <= FALSE;
--      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';     --BC02(HLS43)

      rcell_read <= '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   --BC03(HLS45)
      -- Warte auf die Antwort des Speicher-Mgmt's
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write='1';
      cell_reg  <= rcell_data;     --   (3)
      rcell_read <= '0';
      transmit <= TRUE;
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
      transmit <= FALSE;
--      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';     --BC02(HLS43)

      rcell_read <= '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   --BC03(HLS45)
      -- Warte auf die Antwort des Speicher-Mgmt's
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write='1';
      cell_reg2  <= rcell_data;      --  (4)
      rcell_read <= '0';
      transmit <= TRUE;
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
      transmit <= FALSE;
--      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';     --BC02(HLS43)

      rcell_read <= '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   --BC03(HLS45)
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write='1';
      cell_reg  <= rcell_data;      --  (5)
      rcell_read <= '0';
      transmit <= TRUE;
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
      transmit <= FALSE;
--      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';     --BC02(HLS43)

      rcell_read <= '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   --BC03(HLS45)
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write='1';
      cell_reg2  <= rcell_data;      --  (6)
      rcell_read <= '0';
      transmit <= TRUE;
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
      transmit <= FALSE;
--      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';     --BC02(HLS43)

      rcell_read <= '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   --BC03(HLS45)
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write='1';
      cell_reg  <= rcell_data;      --  (7)
      rcell_read <= '0';
      transmit <= TRUE;
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
      transmit <= FALSE;
--      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';     --BC02(HLS43)

      rcell_read <= '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   --BC03(HLS45)
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write='1';
      cell_reg2  <= rcell_data;       -- (8)
      rcell_read <= '0';
      transmit <= TRUE;
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
      transmit <= FALSE;
--      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';     --BC02(HLS43)

      rcell_read <= '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   --BC03(HLS45)
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write='1';
      cell_reg  <= rcell_data;      --  (9)
      rcell_read <= '0';
      transmit <= TRUE;
```

```
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
      transmit <= FALSE;
--      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';     --BC02(HLS43)

      rcell_read <= '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   --BC03(HLS45)
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write='1';
      cell_reg2  <= rcell_data;      --  (10)
      rcell_read <= '0';
      transmit <= TRUE;
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
      transmit <= FALSE;
--      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';     --BC02(HLS43)

      rcell_read <= '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   --BC03(HLS45)
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write='1';
      cell_reg  <= rcell_data;      --  (11)
      rcell_read <= '0';
      transmit <= TRUE;
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
      transmit <= FALSE;
--      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';     --BC02(HLS43)

      rcell_read <= '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   --BC03(HLS45)
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write='1';
      cell_reg2  <= rcell_data;      --  (12)
      rcell_read <= '0';
      transmit <= TRUE;
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
      transmit <= FALSE;
--      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';     --BC02(HLS43)

      rcell_read <= '1';
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';   --BC03(HLS45)
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_write='1';
      cell_reg  <= rcell_data;       -- (13)
      rcell_read <= '0';
      transmit <= TRUE;
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
      transmit <= FALSE;
      Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';     --BC02(HLS43)

  End loop;  -- main loop
  End Process;


  AHTOUT_Transmit: PROCESS
     VARIABLE hilf         : STD_LOGIC_VECTOR(31 downto 0);

  BEGIN   -- Process
    -- Reset Part
    reg_taken <= FALSE;
    Cell_Sync_out  <= '0';
    Cell_preSync   <= '0';
    Data_out       <= "00000000";
    Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';

    WHILE Chan_bsy  = '0' loop  -- main loop

     Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1' and transmit_new;
     Cell_preSync <= '1';
     Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
     Cell_Sync_out <= '1';

     hilf := cell_reg;    -- Header
     reg_taken <= TRUE;

data_out <= hilf(31 downto 24);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(23 downto 16);
        reg_taken <= FALSE;
        Cell_Sync_out <= '0';
        Cell_preSync <= '0';
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(15 downto 8);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(7 downto 0);
        -- Das HEC Byte bleibt unberuecksichtigt.
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1'; -- BC

     -- Transmit Bytewise die Payload

--     Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1'and transmit;
       hilf := cell_reg2;  -- (2)
       reg_taken <= TRUE;

        -- 4 * 8 Bit = 32 Bit gehen auf die Leitung

data_out <= hilf(31 downto 24);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(23 downto 16);
```

```
        reg_taken <= FALSE;
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(15 downto 8);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(7 downto 0);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';  -- BC

--     Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1'and transmit;

       hilf := cell_reg;  -- (3)
       reg_taken <= TRUE;

data_out <= hilf(31 downto 24);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(23 downto 16);
        reg_taken <= FALSE;
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(15 downto 8);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(7 downto 0);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';  -- BC

--     Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1'and transmit;
       hilf := cell_reg2;   -- (4)
       reg_taken <= TRUE;

data_out <= hilf(31 downto 24);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(23 downto 16);
        reg_taken <= FALSE;
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(15 downto 8);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(7 downto 0);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';  -- BC

        hilf := cell_reg;   -- (5)
        reg_taken <= TRUE;

data_out <= hilf(31 downto 24);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(23 downto 16);
        reg_taken <= FALSE;
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(15 downto 8);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(7 downto 0);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';  -- BC

--     Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1'and transmit;
       hilf := cell_reg2;   -- (6)
       reg_taken <= TRUE;

data_out <= hilf(31 downto 24);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(23 downto 16);
        reg_taken <= FALSE;
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(15 downto 8);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(7 downto 0);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';  -- BC

        hilf := cell_reg;   -- (7)
        reg_taken <= TRUE;

data_out <= hilf(31 downto 24);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(23 downto 16);
        reg_taken <= FALSE;
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(15 downto 8);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(7 downto 0);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';  -- BC

--     Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1'and transmit;
       hilf := cell_reg2;   -- (8)
       reg_taken <= TRUE;

data_out <= hilf(31 downto 24);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(23 downto 16);
        reg_taken <= FALSE;
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(15 downto 8);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(7 downto 0);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';  -- BC

        hilf := cell_reg;   -- (9)
        reg_taken <= TRUE;
```

```
data_out <= hilf(31 downto 24);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(23 downto 16);
        reg_taken <= FALSE;
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(15 downto 8);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(7 downto 0);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';  -- BC

--     Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1'and transmit;
        hilf := cell_reg2;  -- (10)
        reg_taken <= TRUE;

data_out <= hilf(31 downto 24);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(23 downto 16);
        reg_taken <= FALSE;
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(15 downto 8);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(7 downto 0);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';  -- BC

        hilf := cell_reg;   -- (11)
        reg_taken <= TRUE;

data_out <= hilf(31 downto 24);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(23 downto 16);
        reg_taken <= FALSE;
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(15 downto 8);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(7 downto 0);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';  -- BC

--     Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1'and transmit;
        hilf := cell_reg2;   -- (12)
        reg_taken <= TRUE;

data_out <= hilf(31 downto 24);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(23 downto 16);
        reg_taken <= FALSE;
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(15 downto 8);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(7 downto 0);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';  -- BC

        hilf := cell_reg;   -- (13)
        reg_taken <= TRUE;

data_out <= hilf(31 downto 24);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(23 downto 16);
        reg_taken <= FALSE;
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(15 downto 8);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';
data_out <= hilf(7 downto 0);
        Wait UNTIL Clk_aht_out'EVENT AND Clk_aht_out = '1';  -- BC

   End loop; -- Main Loop
  End Process;

 END ahtoutar;
```

Quellcode des VHDL-Testtreibers

```
-- Testdriver testaht_out.vhd  fuer ahtout.vhd
-- Simulation mit Synopsys Simulater v3.3b
-- Autor: W. Lange. Maerz 1998

library IEEE;
library synopsys;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_unsigned.all;
   use IEEE.std_logic_misc.all;
  use IEEE.std_logic_arith.all;
--  use IEEE.std_logic_components.all;
  use synopsys.attributes.all;
--  use STD.Textio.all;

Entity testaht_out IS END;

ARCHITECTURE test_driver OF testaht_out IS

COMPONENT ahtout
  PORT (Clk_aht_out    : IN  STD_LOGIC;
```

```
        Cell_Sync_out  : OUT STD_LOGIC;
        Cell_preSync   : OUT STD_LOGIC;
        Chan_bsy       : IN  STD_LOGIC;
        Reset          : IN STD_LOGIC;
        Data_out       : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        rnew_cell      : OUT STD_LOGIC;
        rcell_data     : IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
        rcell_read     : OUT STD_LOGIC;
        rcell_write    : IN  STD_LOGIC);
END COMPONENT;
--------------------------------

--FOR ALL : ahtout  USE entity WORK.ahtout(ahtoutar);
FOR ALL : ahtout  USE entity WORK.ahtout(SYN_ahtoutar);

  SIGNAL Cell_Sync_out,Cell_preSync,Chan_bsy : STD_LOGIC;
  SIGNAL Reset,rcell_read,rcell_write,rnew_cell : STD_LOGIC;
  SIGNAL GSR : STD_LOGIC;
  SIGNAL Clk_aht_out : STD_LOGIC:= '0';
  SIGNAL rcell_data  : STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL Data_out    : STD_LOGIC_VECTOR(7 DOWNTO 0);


BEGIN

 UUT: ahtout port map (Clk_aht_out,Cell_Sync_out,Cell_preSync,Chan_bsy,
      Reset,Data_out,rnew_cell,rcell_data,rcell_read,rcell_write);

 Clk_aht_out <= NOT Clk_aht_out after 12.5 ns;

 Stimulus: Process

 CONSTANT rcell_data_start :
    STD_LOGIC_VECTOR(31 downto 0) :="00010010001101000101011001110111";

 BEGIN
 -- Clk_com sollte mit der Prozedur Clock loslaufen....

 -- Reset first..
     GSR <= '0';            -- Global reset
     Reset <= '0';
     Chan_bsy <= '1';
     rcell_write <= '0';
     rcell_data <= rcell_data_start;
     rcell_data <= "00000000000000000000000000000000";
     WAIT FOR 20 ns;        -- Warte ein wenig
     Reset <= '1';
     GSR <= '1';
     WAIT FOR 40 ns;
     GSR <= '0';
     Reset <= '0';
     Chan_bsy <= '0';
--     WAIT FOR 20 ns;
     Wait until Clk_aht_out'EVENT AND Clk_aht_out = '1';

 WHILE TRUE loop  -- Simuliere den FIFO
  -- Warte auf einen read - request
     Wait until Clk_aht_out'EVENT AND Clk_aht_out = '1' and rnew_cell = '1';
  --  Wait until Clk_aht_out'EVENT AND Clk_aht_out = '1';
        rcell_write <= '1';
        rcell_data <= rcell_data_start + "00000000000000000000000000000001";
    Wait until Clk_aht_out'EVENT AND Clk_aht_out = '1';
    Wait until Clk_aht_out'EVENT AND Clk_aht_out = '1'and rnew_cell = '0';
         rcell_write <='0';

  FOR i in 1 to 12 loop
    Wait until Clk_aht_out'EVENT AND Clk_aht_out = '1' and rcell_read = '1';
        rcell_write <= '1';
        rcell_data <= rcell_data + "00000000000000000000000000000001" ;
    Wait until Clk_aht_out'EVENT AND Clk_aht_out = '1'and rcell_read = '0';
        rcell_write <='0';
  end loop;

     Wait until Clk_aht_out'EVENT AND Clk_aht_out = '1';
  END loop; -- main loop;
  END Process;
END;
```

Das Synthese-Script

```
/* bc_script fuer ahtout.vhdl  */
sh date >> laufzeit

bc_enable_analysis_info =  "true" /* For BC_View */
analyze -f vhdl ahtout_bc.vhd
elaborate -s ahtout
/* reset- Setzen NACH elaborate ! */
set_behavioral_reset Reset -active high
create_clock Clk_aht_out  -period 25
write -h -o ahtout_elab.db   /* save the elaborated design */
```

```
/* bc_check_design -io cycle_fixed */
/* register_control -inputs */
bc_check_design -io superstate_fixed
/* set_operating_conditions wahtoutom */
sh date >> laufzeit

bc_time_design > ahtout_time.rep
/* bc_time_design braucht nicht mit -fastest ausgef"uhrt zu werden
   da Delay = 0 ist. */
write -h -o ahtout_timed.db    /*  save the timed design */

schedule -io superstate_fixed -effort medium  > ahtout_sched.rep
/* rename_design ahtin_scheduled */ /* keeps the names clean */
report_schedule -abstract > ahtout_fsmloop.rep
report_schedule -operations > ahtout_sched_oploop.rep
report_schedule -summary > ahtout_sched_sumloop.rep

/* write -h -o ahtin_cf.db */       /* save the RTL design to a db file */
vhdlout_dont_write_types = "true"     /* prevents clashes in elaboration */
vhdlout_use_packages = { ieee.std_logic_1164,ieee.std_logic_arith }
vhdlout_equations = "true"       /* ignore gate delays */
vhdlout_levelize = true

/* write -hier -f vhdl -out ahtout_nolev_reg.vhd */ /*schreibt Komponenten raus */
write -hier -f vhdl -out ahtout_rtl.vhd  /* saves a simulatable register_level */
sh date >> laufzeit                     /*   design */

quit
```

BC-Scheduling-Report des AHT_OUT ohne Schleifen

```
Warning: Design has already been timed. (HLS-117)
Loading db file '/afs/wsi/ti/synopsys/1997.8/libraries/syn/gtech.db'
Information: Scheduling 'loop_164_design' ... (HLS-152)
Information: Scheduling 'loop_124_design' ... (HLS-152)
Information: Scheduling 'loop_104_design' ... (HLS-152)
Information: Scheduling 'loop_194_design' ... (HLS-152)
Information: Scheduling 'loop_144_design' ... (HLS-152)
Information: Scheduling 'loop_93_design' ... (HLS-152)
Information: Scheduling 'loop_64_design' ... (HLS-152)
Information: Scheduling 'loop_81_design' ... (HLS-152)
Information: Scheduling 'loop_154_design' ... (HLS-152)
Information: Scheduling 'loop_114_design' ... (HLS-152)
Information: Scheduling 'loop_184_design' ... (HLS-152)
Information: Scheduling 'loop_134_design' ... (HLS-152)
Information: Scheduling 'loop_70_design' ... (HLS-152)
Information: Scheduling 'loop_174_design' ... (HLS-152)
Information: Scheduling '_L0_design' ... (HLS-152)
Information: Scheduling 'AHTOUT_design' ... (HLS-152)
Information: Allocating hardware for 'AHTOUT_design' ... (HLS-153)
****************************************************************************
               Date      : Tue Sep 22 14:52:14 1998
               Version   : 1997.08
               Design    : ahtout
****************************************************************************

****************************************
*  Operation schedule of process AHTOUT:  *
****************************************

Resource types
====================================
    loop......loop boundaries
    p0........1-bit registered output port rcell_read
    p1........1-bit registered output port transmit
    p2........1-bit registered output port rnew_cell
    p3........1-bit registered output port transmit_new
    p4........32-bit registered output port cell_reg
    p5........32-bit registered output port cell_reg2
    p6........32-bit input port rcell_data
    p7........1-bit input port rcell_write
    p8........1-bit input port Chan_bsy


                 p      p     p      p      p     p     p      p      p
                 o      o     o      o      o     o     o      o      o
                 r      r     r      r      r     r     r      r      r
                 t      t     t      t      t     t     t      t      t
-------+------+------+------+-----+------+------+-----+-----+------+------
 cycle | loop |  p6  |  p7  | p8  |  p0  |  p1  | p2  | p3  |  p4  |  p5
----------------------------------------------------------------------------
   0   |..L0..|......|......|.....|.W56..|.W58..|.W57.|.W59.|......|......
   1   |.L32..|......|......|.....|......|......|.....|.....|......|......
       |..L3..|......|......|.....|......|......|.....|.....|......|......
   2   |......|......|......|.R64.|......|......|.....|.....|......|......
   3   |.L61..|......|......|.....|......|......|.....|.....|......|......
       |.L60..|......|......|.....|......|......|.....|.....|......|......
       |.L33..|......|......|.....|......|......|.....|.....|......|......
   4   |......|......|......|.....|......|......|.W68.|.....|......|......
   5   |.L30..|......|......|.....|......|......|.....|.....|......|......
```

```
   6   |......|......|.R70..|......|......|......|.....|.....|......|......
   7   |.L59..|.R71..|......|......|......|......|.W72.|.W73.|.W71..|......
       |.L58..|......|......|......|......|......|.....|.....|......|......
       |.L31..|......|......|......|......|......|.....|.....|......|......
   8   |......|......|......|......|.W78..|......|.....|.....|......|......
   9   |.L28..|......|......|......|......|......|.....|.....|......|......
  10   |......|......|.R81..|......|......|......|.....|.....|......|......
  11   |.L57..|.R82..|......|......|.W84..|.W85..|.....|.W83.|......|.W82..
       |.L56..|......|......|......|......|......|.....|.....|......|......
       |.L29..|......|......|......|......|......|.....|.....|......|......
  12   |......|......|......|......|.W90..|.W87..|.....|.....|......|......
  13   |.L26..|......|......|......|......|......|.....|.....|......|......
  14   |......|......|.R93..|......|......|......|.....|.....|......|......
  15   |.L55..|.R94..|......|......|.W95..|.W96..|.....|.....|.W94..|......
       |.L54..|......|......|......|......|......|.....|.....|......|......
       |.L27..|......|......|......|......|......|.....|.....|......|......
  16   |......|......|......|......|.W101.|.W98..|.....|.....|......|......
  17   |.L24..|......|......|......|......|......|.....|.....|......|......
  18   |......|......|.R104.|......|......|......|.....|.....|......|......
  19   |.L53..|.R105.|......|......|.W106.|.W107.|.....|.....|......|.W105.
       |.L52..|......|......|......|......|......|.....|.....|......|......
       |.L25..|......|......|......|......|......|.....|.....|......|......
  20   |......|......|......|......|.W112.|.W109.|.....|.....|......|......
  21   |.L22..|......|......|......|......|......|.....|.....|......|......
  22   |......|......|.R114.|......|......|......|.....|.....|......|......
  23   |.L51..|.R115.|......|......|.W116.|.W117.|.....|.....|.W115.|......
       |.L50..|......|......|......|......|......|.....|.....|......|......
       |.L23..|......|......|......|......|......|.....|.....|......|......
  24   |......|......|......|......|.W122.|.W119.|.....|.....|......|......
  25   |.L20..|......|......|......|......|......|.....|.....|......|......
  26   |......|......|.R124.|......|......|......|.....|.....|......|......
  27   |.L49..|.R125.|......|......|.W126.|.W127.|.....|.....|......|.W125.
       |.L48..|......|......|......|......|......|.....|.....|......|......
       |.L21..|......|......|......|......|......|.....|.....|......|......
  28   |......|......|......|......|.W132.|.W129.|.....|.....|......|......
  29   |.L18..|......|......|......|......|......|.....|.....|......|......
  30   |......|......|.R134.|......|......|......|.....|.....|......|......
  31   |.L47..|.R135.|......|......|.W136.|.W137.|.....|.....|.W135.|......
       |.L46..|......|......|......|......|......|.....|.....|......|......
       |.L19..|......|......|......|......|......|.....|.....|......|......
  32   |......|......|......|......|.W142.|.W139.|.....|.....|......|......
  33   |.L16..|......|......|......|......|......|.....|.....|......|......
  34   |......|......|.R144.|......|......|......|.....|.....|......|......
  35   |.L45..|.R145.|......|......|.W146.|.W147.|.....|.....|......|.W145.
       |.L44..|......|......|......|......|......|.....|.....|......|......
       |.L17..|......|......|......|......|......|.....|.....|......|......
  36   |......|......|......|......|.W152.|.W149.|.....|.....|......|......
  37   |.L14..|......|......|......|......|......|.....|.....|......|......
  38   |......|......|.R154.|......|......|......|.....|.....|......|......
  39   |.L43..|.R155.|......|......|.W156.|.W157.|.....|.....|.W155.|......
       |.L42..|......|......|......|......|......|.....|.....|......|......
       |.L15..|......|......|......|......|......|.....|.....|......|......
  40   |......|......|......|......|.W162.|.W159.|.....|.....|......|......
  41   |.L12..|......|......|......|......|......|.....|.....|......|......
  42   |......|......|.R164.|......|......|......|.....|.....|......|......
  43   |.L41..|.R165.|......|......|.W166.|.W167.|.....|.....|......|.W165.
       |.L40..|......|......|......|......|......|.....|.....|......|......
       |.L13..|......|......|......|......|......|.....|.....|......|......
  44   |......|......|......|......|.W172.|.W169.|.....|.....|......|......
  45   |.L10..|......|......|......|......|......|.....|.....|......|......
  46   |......|......|.R174.|......|......|......|.....|.....|......|......
  47   |.L39..|.R175.|......|......|.W176.|.W177.|.....|.....|.W175.|......
       |.L38..|......|......|......|......|......|.....|.....|......|......
       |.L11..|......|......|......|......|......|.....|.....|......|......
  48   |......|......|......|......|.W182.|.W179.|.....|.....|......|......
  49   |..L8..|......|......|......|......|......|.....|.....|......|......
  50   |......|......|.R184.|......|......|......|.....|.....|......|......
  51   |.L37..|.R185.|......|......|.W186.|.W187.|.....|.....|......|.W185.
       |.L36..|......|......|......|......|......|.....|.....|......|......
       |..L9..|......|......|......|......|......|.....|.....|......|......
  52   |......|......|......|......|.W192.|.W189.|.....|.....|......|......
  53   |..L6..|......|......|......|......|......|.....|.....|......|......
  54   |......|......|.R194.|......|......|......|.....|.....|......|......
  55   |.L35..|.R195.|......|......|.W196.|.W197.|.....|.....|.W195.|......
       |.L34..|......|......|......|......|......|.....|.....|......|......
       |..L7..|......|......|......|......|......|.....|.....|......|......
  56   |......|......|......|......|......|.W199.|.....|.....|......|......
  57   |..L5..|......|......|......|......|......|.....|.....|......|......
       |..L4..|......|......|......|......|......|.....|.....|......|......
       |..L2..|......|......|......|......|......|.....|.....|......|......
       |..L1..|......|......|......|......|......|.....|.....|......|......


Operation name abbreviations
============================
    L0........loop boundaries AHTOUT_design_loop_begin
    156 ops removed w.l.
    W199......1-bit write _L0/transmit_199


*************************************************************************
Information: control FSM status and output not registered
Control FSM has 58 states
```

```
One-Hot coding style selected, state vector is 58 bits long.
State  Code
s_0_0  1----------------------------------------------------------
        56 states removed w.l.
s_15_55  ----------------------------------------------------------1

Control unit for process AHTOUT of design ahtout is hardwired
Control unit for process AHTOUT synthesized.
Information: Scheduling 'loop_219_design' ... (HLS-152)
Information: Scheduling '_L1_design' ... (HLS-152)
Information: Scheduling 'AHTOUT_Transmit_design' ... (HLS-152)
Information: Allocating hardware for 'AHTOUT_Transmit_design' ... (HLS-153)
******************************************************************************
               Date      : Tue Sep 22 14:52:47 1998
               Version   : 1997.08
               Design    : ahtout
******************************************************************************

************************************************
*  Operation schedule of process AHTOUT_Transmit:  *
************************************************

Resource types
====================================
    loop......loop boundaries
    p0........1-bit registered output port reg_taken
    p1........1-bit registered output port Cell_Sync_out
    p2........8-bit registered output port Data_out
    p3........1-bit registered output port Cell_preSync
    p4........1-bit input port Chan_bsy
    p5........32-bit input port cell_reg2
    p6........32-bit input port cell_reg
    p7........1-bit input port transmit_new


                  p      p      p      p      p      p      p      p
                  o      o      o      o      o      o      o      o
                  r      r      r      r      r      r      r      r
                  t      t      t      t      t      t      t      t
-------+------+------+------+------+------+------+------+------+------
 cycle | loop |  p4  |  p5  |  p6  |  p7  |  p0  |  p1  |  p2  |  p3
-------------------------------------------------------------------------
   0   |..L0..|......|......|......|......|.W211.|.W212.|.W214.|.W213.
   1   |..L3..|.R217.|......|......|......|......|......|......|......
   2   |..L7..|......|......|......|......|......|......|......|......
       |..L6..|......|......|......|......|......|......|......|......
   3   |......|......|......|......|.R219.|......|......|......|......
   4   |.L10..|......|......|......|......|......|......|......|.W220.
       |..L9..|......|......|......|......|......|......|......|......
       |..L8..|......|......|......|......|......|......|......|......
   5   |......|......|......|.R224.|......|.W225.|.W222.|.W227.|......
   6   |......|......|......|......|......|.W230.|.W231.|.W229.|.W232.
   7   |......|......|......|......|......|......|......|.W234.|......
   8   |......|......|......|......|......|......|......|.W236.|......
   9   |......|......|.R243.|......|......|.W244.|......|.W248.|......
  10   |......|......|......|......|......|.W251.|......|.W250.|......
  11   |......|......|......|......|......|......|......|.W253.|......
  12   |......|......|......|......|......|......|......|.W255.|......
  13   |......|......|......|.R260.|......|.W261.|......|.W263.|......
  14   |......|......|......|......|......|.W266.|......|.W265.|......
  15   |......|......|......|......|......|......|......|.W268.|......
  16   |......|......|......|......|......|......|......|.W270.|......
  17   |......|......|.R274.|......|......|.W275.|......|.W277.|......
  18   |......|......|......|......|......|.W280.|......|.W279.|......
  19   |......|......|......|......|......|......|......|.W282.|......
  20   |......|......|......|......|......|......|......|.W284.|......
  21   |......|......|......|.R287.|......|.W288.|......|.W290.|......
  22   |......|......|......|......|......|.W293.|......|.W292.|......
  23   |......|......|......|......|......|......|......|.W295.|......
  24   |......|......|......|......|......|......|......|.W297.|......
  25   |......|......|.R301.|......|......|.W302.|......|.W304.|......
  26   |......|......|......|......|......|.W307.|......|.W306.|......
  27   |......|......|......|......|......|......|......|.W309.|......
  28   |......|......|......|......|......|......|......|.W311.|......
  29   |......|......|......|.R314.|......|.W315.|......|.W317.|......
  30   |......|......|......|......|......|.W320.|......|.W319.|......
  31   |......|......|......|......|......|......|......|.W322.|......
  32   |......|......|......|......|......|......|......|.W324.|......
  33   |......|......|.R328.|......|......|.W329.|......|.W331.|......
  34   |......|......|......|......|......|.W334.|......|.W333.|......
  35   |......|......|......|......|......|......|......|.W336.|......
  36   |......|......|......|......|......|......|......|.W338.|......
  37   |......|......|......|.R341.|......|.W342.|......|.W344.|......
  38   |......|......|......|......|......|.W347.|......|.W346.|......
  39   |......|......|......|......|......|......|......|.W349.|......
  40   |......|......|......|......|......|......|......|.W351.|......
  41   |......|......|.R355.|......|......|.W356.|......|.W358.|......
  42   |......|......|......|......|......|.W361.|......|.W360.|......
  43   |......|......|......|......|......|......|......|.W363.|......
  44   |......|......|......|......|......|......|......|.W365.|......
  45   |......|......|......|.R368.|......|.W369.|......|.W371.|......
  46   |......|......|......|......|......|.W374.|......|.W373.|......
```

```
  47   |......|......|......|......|......|......|......|.W376.|......
  48   |......|......|......|......|......|......|......|.W378.|......
  49   |......|......|.R382.|......|......|.W383.|......|.W385.|......
  50   |......|......|......|......|......|.W388.|......|.W387.|......
  51   |......|......|......|......|......|......|......|.W390.|......
  52   |......|......|......|......|......|......|......|.W392.|......
  53   |......|......|......|.R395.|......|.W396.|......|.W398.|......
  54   |......|......|......|......|......|.W401.|......|.W400.|......
  55   |......|......|......|......|......|......|......|.W403.|......
  56   |......|......|......|......|......|......|......|.W405.|......
  57   |..L5..|......|......|......|......|......|......|......|......
       |..L4..|......|......|......|......|......|......|......|......
       |..L2..|......|......|......|......|......|......|......|......
       |..L1..|......|......|......|......|......|......|......|......


Operation name abbreviations
============================
    L0.......loop boundaries AHTOUT_Transmit_design_loop_begin
    110 ops removed
    W405......8-bit write _L1/Data_out_405


************************************************************************
Information: control FSM status and output not registered
Control FSM has 60 states
One-Hot coding style selected, state vector is 60 bits long.
State  Code
s_0_0  1-----------------------------------------------------------
        58 states removed w.l.
s_2_4  -----------------------------------------------------------1

Control unit for process AHTOUT_Transmit of design ahtout is hardwired
Control unit for process AHTOUT_Transmit synthesized.

Memory usage during scheduling 21337 Kbytes.
CPU usage during scheduling was 86 seconds.
Scheduled design ahtout.
Warning: Overwriting design file '/afs/informatik.uni-tuebingen.de/home/wlange/hilan/aht/ahtout_bc/ahtout.db'. (DDB-24)
Current design is 'ahtout'.


************************************************************************
              Date      : Tue Sep 22 14:55:55 1998
              Version   : 1997.08
              Design    : ahtout
************************************************************************

****************************************
*  Summary report for process AHTOUT:  *
****************************************
------------------------------------------------------------------------------
     Timing Summary
------------------------------------------------------------------------------
 Clock period 25.00
 Loop timing information:
    AHTOUT.....................................57 cycles (cycles 0 - 57)
        _L0....................................56 cycles (cycles 1 - 57)
            loop_64.............................2 cycles (cycles 1 - 3)
                (exit) EXIT_L64..................... (cycle 3)
            loop_70.............................2 cycles (cycles 5 - 7)
                (exit) EXIT_L70..................... (cycle 7)
            loop_81.............................2 cycles (cycles 9 - 11)
                (exit) EXIT_L81..................... (cycle 11)
            loop_93.............................2 cycles (cycles 13 - 15)
                (exit) EXIT_L93..................... (cycle 15)
            loop_104............................2 cycles (cycles 17 - 19)
                (exit) EXIT_L104.................... (cycle 19)
            loop_114............................2 cycles (cycles 21 - 23)
                (exit) EXIT_L114.................... (cycle 23)
            loop_124............................2 cycles (cycles 25 - 27)
                (exit) EXIT_L124.................... (cycle 27)
            loop_134............................2 cycles (cycles 29 - 31)
                (exit) EXIT_L134.................... (cycle 31)
            loop_144............................2 cycles (cycles 33 - 35)
                (exit) EXIT_L144.................... (cycle 35)
            loop_154............................2 cycles (cycles 37 - 39)
                (exit) EXIT_L154.................... (cycle 39)
            loop_164............................2 cycles (cycles 41 - 43)
                (exit) EXIT_L164.................... (cycle 43)
            loop_174............................2 cycles (cycles 45 - 47)
                (exit) EXIT_L174.................... (cycle 47)
            loop_184............................2 cycles (cycles 49 - 51)
                (exit) EXIT_L184.................... (cycle 51)
            loop_194............................2 cycles (cycles 53 - 55)
                (exit) EXIT_L194.................... (cycle 55)
------------------------------------------------------------------------------
     Area Summary
------------------------------------------------------------------------------
 Estimated combinational area    83
 Estimated sequential area       1850
 TOTAL                           1933

 58 control states
```

```
 72 basic transitions
 3 control inputs
 42 control outputs

------------------------------------------------------------------------------
     Resource types
------------------------------------------------------------------------------
Register Types
=======================================
1-bit register.....................2

Operator Types
=======================================

I/O Ports
=======================================
1-bit input port...................2
1-bit registered output port.......4
32-bit input port..................1
32-bit registered output port......2

------------------------------------------------------------------------------
************************************************
*  Summary report for process AHTOUT_Transmit:  *
************************************************
------------------------------------------------------------------------------
     Timing Summary
------------------------------------------------------------------------------
 Clock period 25.00
 Loop timing information:
    AHTOUT_Transmit.............................57 cycles (cycles 0 - 57)
        _L1.....................................56 cycles (cycles 1 - 57)
            (exit) EXIT_L217.............................. (cycle 2)
            loop_219...............................2 cycles (cycles 2 - 4)
                (exit) EXIT_L219.......................... (cycle 4)
------------------------------------------------------------------------------
     Area Summary
------------------------------------------------------------------------------
 Estimated combinational area    213
 Estimated sequential area       2330
 TOTAL                           2543

 60 control states
 63 basic transitions
 3 control inputs
 56 control outputs

------------------------------------------------------------------------------
     Resource types
------------------------------------------------------------------------------
Register Types
=======================================
1-bit register.....................2
32-bit register....................1

Operator Types
=======================================

I/O Ports
=======================================
1-bit input port...................2
1-bit registered output port.......3
8-bit registered output port.......1
32-bit input port..................2

------------------------------------------------------------------------------
```

13 Literatur

Literatur

[LaRo97] W.Lange, W. Rosenstiel, Modellierung einer ATM-Switch-Steuerung. WSI 97-6 Wilhelm-Schickard-Institut Universität Tübingen (Bericht) WSI 1997 ISSN 0964-3852

[DeMi94] G. De Micheli, Synthesis and Optimization of Digital Circuits, McGraw-Hill (1994)

[Gaj92] Daniel D. Gajski, Nikil D. Dutt, Allen C-H Wu, Steve Y-L Lin, High-Level Synthesis Introduction to Chip and System Design, Kluwer Academic Publishers (1992)

[Syn97] Synopsys Behavioral Compiler Methodology Guide. Version 1997.08

[SynUg97] Synopsys Behavioral Compiler User Guide. Version 1997.08