Diplomarbeit

Entwicklung eines optisch gekoppelten Caches

Jochen Preiß

Universität des Saarlandes
Fachrichtung 6.2 – Informatik
Lehrstuhl für Rechnerarchitektur und Parallelrechner
Prof. Dr. W. J. Paul

Februar 2001

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt habe. Ich habe diese Arbeit keinem anderen Prüfungsamt vorgelegt.

Saarbrücken, im Februar 2001

Jochen Preiß

Danksagung

Ich möchte den folgenden Personen danken, die mir das Erstellen dieser Arbeit ermöglicht haben:

- Professor Paul für die Vergabe dieses interessanten Themas,
- meinen Eltern für die Unterstützung in schwierigen Zeiten meiner Diplomarbeit,
- Michael Klein für gute Zusammenarbeit in dem Projekt,
- Professor Oppower, Markus Scholl, Günther Renz und Stefan Scharl für die Lösung vieler Probleme bei der Glasfaser-Anbindung,
- Thomas Drischel und Wayne McKinley für die Beantwortung meiner vielen Fragen zu ECL und der Herstellung der Dies,
- Rolf Hackenes für die Hilfe bei der Installation und Wartung von Cadence,
- Christian Jacobi für das Korrekturlesen der Arbeit und die vielen fruchtbaren Gespräche,
- Michael Bosch für die vielen interessanten Ideen, die das Design von Sequenzer und Parallelisierer immer komplizierter und besser gemacht haben,
- Peter Bach für die Hilfe bei anfänglichen Schwierigkeiten
- und dem gesamten Lehrstuhl und insbesondere dem Labor für die gute Atmosphäre.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

In den letzten Jahrzehnten hat sich die Geschwindigkeit, mit der Daten innerhalb integrierter Chips verarbeitet werden können, entsprechend dem Mooreschen Gesetz etwa alle 18 Monate verdoppelt. Die Geschwindigkeit, mit der Daten zwischen verschiedenen Chips ausgetauscht werden können, ist nicht annähernd so schnell gewachsen. Dadurch entwickelt sich der Datenaustausch zwischen den Chips zunehmend zum Flaschenhals von Systemen.
Eine wesentliche Verbesserung der Situation ist bei der derzeit üblichen elektrischen Übertragung zwischen Chips nicht abzusehen. Verschiedene physikalische Effekte begrenzen die Frequenz, mit der die Daten übertragen werden. Zusätzlich ist es nicht möglich, die Anzahl der Leitungen beliebig zu erhöhen, da die Leitungen sich gegenseitig stören, wenn sie zu nahe beieinander liegen.

Diese physikalischen Grenzen gibt es bei der optischen Übertragung nicht oder sind zumindest bei weitem noch nicht erreicht. Es sind damit höhere Frequenzen und eine höhere Anzahl von Leitungen möglich. Da optische Verbindungen allerdings bisher sehr aufwendig waren, wurden sie nur zur Verbindung entfernter Systeme verwendet (zum Beispiel für ATM und Gigabit Ethernet).
In den letzten Jahren ist es jedoch gelungen, die Größe der optischen Bauteile wie Laser und Receiver so weit zu schrumpfen, dass es möglich ist, sie zusammen mit der eigentlichen Logik in einem Chip zu integrieren. Zwar ist derzeit (Februar 2001) die optische Datenübertragung zwischen Chips noch nicht wesentlich schneller als die elektrische Übertragung, sie verspricht aber aus den genannten Gründen mehr Entwicklungspotenzial.

Hohe Geschwindigkeiten bei der Datenübertragung zwischen Chips sind zum Beispiel bei der Verbindung von Cache und Hauptspeicher wünschenswert. Um die Vorteile der Glasfaser-Übertragung auch bei kurzen Übertragungsstrecken zu demonstrieren, wird derzeit am Lehrstuhl von Professor Paul in Zusammenarbeit mit dem Lehrstuhl für Technische Physik der DLR in Stuttgart ein Prototyp eines optisch gekoppelten Caches entwickelt.
Der Prototyp besteht aus einem Cache-Chip, der über Glasfaser-Verbindungen mit vier RAM-Chips verbunden ist. Jeder dieser Chips besteht innerhalb des Gehäuses aus einem Die, der die eigentliche Logik enthält und die Glasfaser-Übertragung steuert, sowie den Dies mit den Lasern und Receivern.

Für die Übertragung der Daten zwischen Cache und Hauptspeicher stehen in diesem Projekt nur wenige Glasfaserkabel zur Verfügung. Diese können allerdings mit einer hohen Frequenz betrieben werden. Aus diesem Grund werden die Daten vor der Übertragung mit der hohen Frequenz sequenzialisiert und nach der Übertragung wieder parallelisiert.
Ziel der vorliegenden Arbeit war das Design und die Entwicklung des Logik-Dies. Einen Hauptteil dieser Arbeit stellten die Schaltkreise zur Sequenzialisierung und Parallelisierung dar. Die Herstellung des Dies ist abgeschlossen; der Die befindet sich momentan beim Fertigungstest.

Das Projekt sowie einige der Ideen aus dieser Arbeit werden demnächst im Journal Laseropto [31] veröffentlicht.

Diese Arbeit gliedert sich wie folgt. In Kapitel 2 wird die allgemeine Funktionsweise von Caches erklärt und der Aufbau des zu entwickelnden Prototypen beschrieben. Das grundlegende Design des entwickelten Dies wird in Kapitel 3 beschrieben. Dabei wird der Die in die Cache/RAM-Logik und die für die Sequenzialisierung beziehungsweise Parallelisierung notwendigen Schaltkreise aufgeteilt. Auf die Cache/RAM-Logik wird in Kapitel 4 näher eingegangen. Das Design der Schaltkreise zur Sequenzialisierung und Parallelisierung wird in Kapitel 5 behandelt. Für die kritischen Teile dieser Schaltkreise wird ein Korrektheitsbeweis geführt. Die zur Realisierung des Dies notwendigen Schritte werden in Kapitel 6 beschrieben. Kapitel 7 fasst die Arbeit zusammen und gibt einen Ausblick. Im Anhang finden sich die wichtigsten Schaltpläne, die Quellcodes des Designs sowie Implementierungsdetails.

Kapitel 2

Grundlagen

In diesem Kapitel werden die zum Verständnis der folgenden Kapitel notwendigen Grundlagen gelegt. Abschnitt 2.1 führt die verwendete Notation ein. Der grundsätzliche Aufbau von Caches wird im Abschnitt 2.2 erklärt. In Abschnitt 2.3 wird der entwickelte Prototyp des optisch gekoppelten Caches und die Testumgebung beschrieben. Die für die Entwicklung des Prototypen verwendeten Technologien werden in Abschnitt 2.4 behandelt.

2.1 Notation

2.1.1 Signale und Busse

Um die in den folgenden Kapiteln beschriebenen Schaltungen auf Korrektheit zu prüfen, wird im folgenden ein Modell eingeführt, das das Verhalten der Schaltungen möglichst einfach beschreibt.
Leitungen, die den Ausgang eines Gatters oder Registers mit einer Menge von Gatter- bzw. Registereingängen verbinden, werden als *Signale* bezeichnet. Ist einer der Eingänge der Clock-Eingang eines Registers, so wird das Signal im Zusammenhang mit dem Register auch als *Clock-Signal* bezeichnet.
Ein Signal X kann Werte aus $\mathbb{B}' := \{0, 1, \Omega\}$ annehmen. Dabei stellt Ω den nicht definierten Wert dar. Die Grundlage der Zuordnung bilden die technologieabhängigen Schwellspannungen V_l und V_h ($V_l \leq V_h$). Sei U die Spannung auf der zu X gehörenden Leitung. Für das Signal X gilt dann

$$X = \begin{cases} 1 & \text{falls } U > V_h, \\ \Omega & \text{falls } V_l \leq U \leq V_h, \\ 0 & \text{falls } U < V_l. \end{cases}$$

Der Fall $X = \Omega$ tritt bei einer azyklischen Verschaltung der Gatter nur kurzzeitig bei den Schaltvorgängen auf. Die Korrektheitsbeweise in späteren Kapiteln gehen von einem idealisierten Timing aus, Verzögerungen der Gatter und Register werden also nicht berücksichtigt. In diesem Fall nehmen die Signale nur Werte aus $\mathbb{B} := \{0, 1\}$ an.
Signale können *high-* und *low-aktiv* sein. Ein high-aktives Signal heißt *aktiv*, wenn es den Wert 1 hat; ein low-aktives heißt aktiv, wenn es den Wert 0 hat.

Low-aktive Signale werden durch einen Querstrich über dem Bezeichner gekennzeichnet (zum Beispiel $\overline{X}$).

Zusammengehörende Signale werden zu Bussen zusammengefaßt. Ein Bus aus n Signalen wird als n-Tupel behandelt. Als abkürzende Schreibweise für den Bus $(X[n-1],\ldots,X[0])$ wird die Bezeichnung $X[n-1:0]$ benutzt. Entsprechend bezeichnet $X[j:i]$ für $j \geq i$ den Teilbus $(X[j],\ldots,X[i])$. Ist die Anzahl der Signale des Busses im Zusammenhang eindeutig, so wird mit X der gesamte Bus $X[n-1:0]$ bezeichnet.
Ein Bus $X[n-1:0]$ und die von ihm dargestellte Zahl $\sum_{i=0}^{n-1} X[i] \cdot 2^i$ werden miteinander identifiziert. Um die Bedeutung deutlich zu machen, wird für die dargestellte Zahl auch die explizite Schreibweise $\langle X[n-1:0]\rangle$ benutzt.
Sollen den Signalen eines Busses $X[j:i]$ Werte aus $\mathbb{B}^{j-i+1}$ zugeordnet werden, können die folgenden abkürzenden Schreibweisen benutzt werden: für $x_j,\ldots,x_i$ aus $\mathbb{B}$ bezeichnet $x_j \ldots x_i$ den Vektor $(x_j,\ldots,x_i)$ und für $x \in \mathbb{B}$ bezeichnet x^n das n-malige Hintereinanderschreiben von x.

Beispiel: $X[n-1:0] = (\underbrace{0,\ldots,0}_{\text{(n-2)-mal}},1,0)$ wird abgekürzt zu $X = 0^{n-2}10$.

2.1.2 Gatter

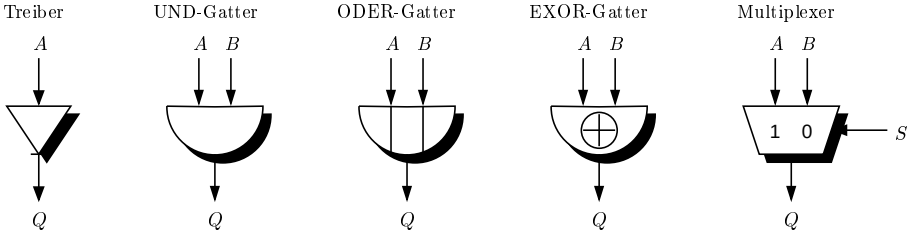

Abbildung 2.1: Schaltsymbole für Gatter

Abbildung 2.1 zeigt die in den Schaltplänen für Gatter verwendeten Symbole. Inverter werden in den Schaltplänen nicht extra gezeichnet. Invertierte Ein- und Ausgänge werden durch einen Kreis gekennzeichnet (siehe Abbildung 2.2).
Signale und Gatter eines Busses werden in den Schaltbildern zusammengefaßt. Die Breite der Busse wird durch einen Querbalken an den Signalen gekennzeichnet. Liegt ein Signal bei allen Gattern am gleichen Eingang an, so wird es nur einfach eingezeichnet. Abbildung 2.2 zeigt zum Beispiel einen 32-Bit-Multiplexer.

2.1.3 Register

Um Register modellieren zu können, ist es notwendig, für jedes Signal eine Variable einzuführen, mit der das zeitliche Verhalten des Signals beschrieben wird. Sei X der Name eines Signals oder Busses und $t \in \mathbb{Z}$. Mit X^t wird der Zustand des Signals am Ende des Taktes t bezeichnet. Ist es nicht offensichtlich, auf welches Clocksignal sich bezogen wird, so wird dies gesondert erwähnt.

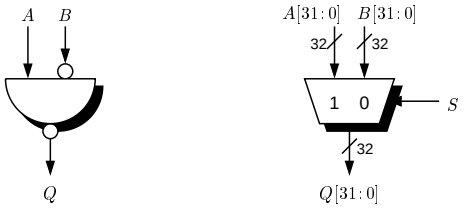

Abbildung 2.2: NAND-Gatter mit invertiertem B-Eingang, 32 Bit Multiplexer

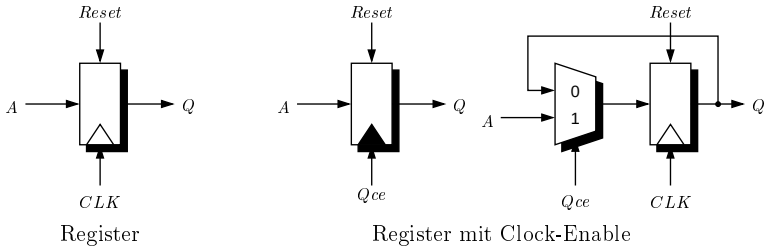

Abbildung 2.3: Schaltsymbole für Register und Register mit Clock-Enable

Abbildung 2.3 zeigt die verwendeten Schaltsymbole für Register mit und ohne Clock-Enable. Der Eingang für das asynchrone Resetsignal der Register wird, falls nicht benötigt, weggelassen. Da in den verwendetet Technologien (siehe Abschnitt 2.4) keine Register mit Clock-Enable zur Verfügung stehen, werden diese durch ein normales Register und einen Mux realisiert. Das Clocksignal, das bei Registern mit Clock-Enable verwendet wird, ist üblicherweise im Zusammenhang eindeutig. Es wird deshalb nicht eingezeichnet.

2.1.4 Speicher

Sei R ein Speicher mit $M = 2^m$ Zeilen mit jeweils N Bits. Mit $R[i][N-1:0]$ wird die Zeile i des Speichers bezeichnet. Seien $A[m-1:0]$ der Adresseingang und $\overline{Dw}$ das Schreibsignal von R. Ist $\overline{Dw}$ inaktiv, so liegt am Datenausgang $DO[N-1:0]$ von R der Inhalt der Zeile $\langle A[m-1:0]\rangle$ an:

$$DO = R[\langle A\rangle].$$

Ist $\overline{Dw}$ aktiv, wird die durch A selektierte Zeile mit dem Wert des Dateneingangs $DI[N-1:0]$ beschrieben:

$$R[\langle A\rangle] := DI.$$

Die Zeilen von R seien aufgeteilt in n einzeln beschreibbare Teile der Größe $B := N/n$ Bit. R benötigt dann n Schreibsignale $\overline{Dw}[n-1:0]$. Ist $\overline{Dw} = 1^n$, so gilt weiterhin $DO := R[\langle A\rangle]$. Sei $B_j := j \cdot B$. Ist $\overline{Dw} \neq 1^n$, so gilt für alle $j \in \{0, \ldots, n-1\}$:

$$R[\langle A\rangle][B_{j+1}-1:B_j] := \begin{cases} DI[B_{j+1}-1:B_j] & \text{falls } \overline{Dw}[j] = 0, \\ R[\langle A\rangle][B_{j+1}-1:B_j] & \text{falls } \overline{Dw}[j] = 1. \end{cases}$$

Es werden also genau die Abschnitte der Zeile überschrieben, deren zugehöriges Schreibsignal aktiv ist. Abbildung 2.4 zeigt das Symbol eines byteweise beschreibbaren RAM-Bausteins mit 64 Zeilen zu je 32 Bit.

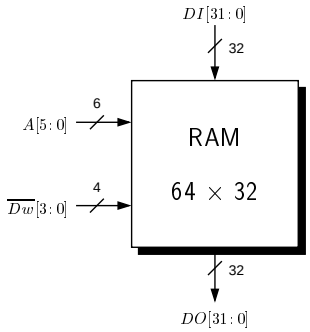

Abbildung 2.4: Symbol eines byteweise beschreibbaren 64 × 32 Bit RAM-Bausteins

2.2 Caches

2.2.1 Das Lokalitätsprinzip

Bei der Entwicklung von Computern stehen sich zwei gegensätzliche Ziele gegenüber. Einerseits wünscht man sich einen immer größeren Hauptspeicher, andererseits sollen Speicherzugriffe immer schneller werden. Hauptspeicher in der derzeit (Februar 2001) üblichen Größe von 128 MB und mehr ist momentan nur auf externen Chips in der platzsparenden, aber langsamen DRAM-Technologie (dynamic random access memory) möglich. Die schnellste Speichertechnologie, im Prozessorchip integriertes SRAM (static RAM), lässt aus Kostengründen nur Größen von wenigen MB zu.
Eine Möglichkeit, die Vorteile von kleinen, schnellen Speichern mit denen von großen, langsamen zu kombinieren, bietet das *Lokalitätsprinzip* [17]: greift ein Prozessor auf eine Speicherzelle zu, so ist die Wahrscheinlichkeit groß, dass er

- auch auf eine benachbarte Speicherzelle zugreift (*örtliche Lokalität*),
- in naher Zukunft noch einmal auf die gleiche Zelle zugreift (*zeitliche Lokalität*).

Das Lokalitätsprinzip läßt sich leicht am Programmspeicher nachvollziehen. Ist der zuletzt ausgeführte Befehl kein Sprung, so wird als nächstes der im Speicher darauffolgende Befehl ausgeführt (örtliche Lokalität). Zusätzlich enthalten die meisten Programme Schleifen, die mehrmals durchlaufen werden (zeitliche Lokalität). Als Daumenregel gilt, dass für 90% der Laufzeit nur 10% des Programmcodes benutzt werden [23, S. 38].

Das Lokalitätsprinzip macht man sich durch eine Speicher-Hierarchie zunutze. Am unteren Ende steht der Hauptspeicher, der in platzsparender Technologie aufgebaut ist. Zwischen Prozessor und Hauptspeicher wird ein kleiner, schneller

Speicher geschaltet, der eine Kopie der gerade benötigten Daten enthält. Ein solcher Zwischenspeicher wird *Cache* genannt.
Greift der Prozessor auf eine Speicherzelle zu, die bereits im Cache liegt (*Cache-Hit*), so können die Daten mit der geringen Verzögerung des Caches geliefert werden. Nur wenn sich die Daten nicht im Cache befinden (*Cache-Miss*), müssen sie aus dem Hauptspeicher nachgeladen werden. Wird in naher Zukunft noch einmal auf die gleiche Speicherzelle zugegriffen (zeitliche Lokalität), befinden sich die Daten dann bereits im Cache. Um die örtliche Lokalität auszunutzen, werden in Falle eines Cache-Misses zusätzlich benachbarte Speicherzellen in den Cache nachgeladen. Die Menge der nachgeladenen Speicherzellen wird als *Block* bezeichnet.

Moderne Architekturen benutzen meist mehrere Stufen von Caches [1, 26]. Da das prinzipielle Verhalten eines Caches in jeder Stufe gleich ist, wird im folgenden von einer einstufigen Cache-Hierarchie ausgegangen. Gibt es mehrere Caches auf gleicher Stufe – zum Beispiel bei getrenntem Instruktions- und Daten-Cache –, so muss die Kohärenz der Caches sichergestellt werden. Der in dieser Arbeit entwickelte Cache soll jedoch alleine auf den Hauptspeicher zugreifen. Auf die ansonsten benötigten Schaltkreise wird nicht eingegangen; sie werden zum Beispiel in [37] beschrieben.

2.2.2 Design

Sei 2^m Byte die Größe des Hauptspeichers und 2^b Byte die Größe der Blöcke. Abbildung 2.5 zeigt die Einteilung der Speicherzellen in Blöcke. Sei S eine Speicherzelle mit Adresse $\langle a[m-1:0]\rangle$. Der zu S gehörende Block enthält alle Speicherzellen, für deren Adresse $\langle a'[m-1{:}0]\rangle$ gilt

$$a'[m-1{:}b] = a[m-1{:}b].$$

Entsprechend wird $ba = \langle a[m-1{:}b]\rangle$ Blockadresse von S genannt.

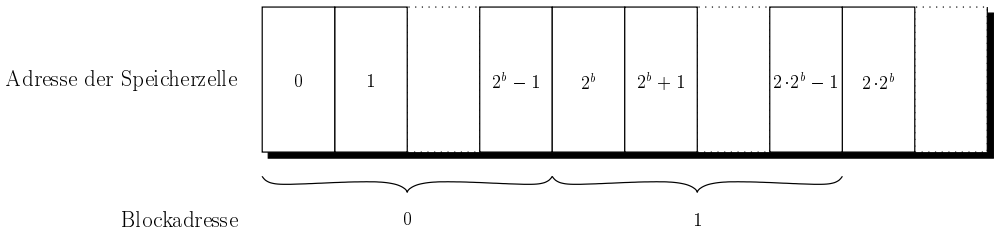

Abbildung 2.5: Einteilung der Speicherzellen in Blöcke

Das grundlegende Design eines Caches läßt sich durch die folgenden vier Fragen spezifizieren [23, S. 376ff]:

- Wo kann ein Block im Cache gespeichert werden? (*Block-Platzierung*)
- Wie kann ein Block im Cache identifiziert werden? (*Block-Identifizierung*)
- Welcher Block soll bei einem cache-miss ersetzt werden? (*Block-Ersetzung*)
- Was passiert bei einem Schreibzugriff? (*Schreibzugriffs-Strategie*)

Block-Platzierung

Der Cache-Speicher wird in sogenannte Cachelines eingeteilt. Jede Cacheline kann genau einen Block speichern. Jeweils K Cachelines werden zu einem *Set* zusammengefaßt. Man spricht dann von einem *K-way set associative* Cache. Im Spezialfall $K = 1$ redet man von einem *direct mapped* Cache. Gibt es nur einen Set, so wird der Cache *fully associative* genannt.
Für diese Arbeit wird davon ausgegangen, dass $K = 2^k$ eine Zweierpotenz ist.[1] Sei 2^s die Anzahl der Sets im Cache. Dann enthält der Cache 2^{s+k} Cachelines mit jeweils 2^b Bytes und damit insgesamt 2^{s+k+b} Bytes.
Jeder Blockadresse ba wird eine Setadresse sa zugeordnet. Ein Block aus dem Hauptspeicher kann nur in dem ihm zugeordneten Set abgespeichert werden, innerhalb des Sets jedoch in einer beliebigen Cacheline. Üblicherweise wird für die Zuordnung die folgende Relation verwendet:

$$sa = (ba \mod 2^s) = \langle a[b+s-1{:}b] \rangle.$$

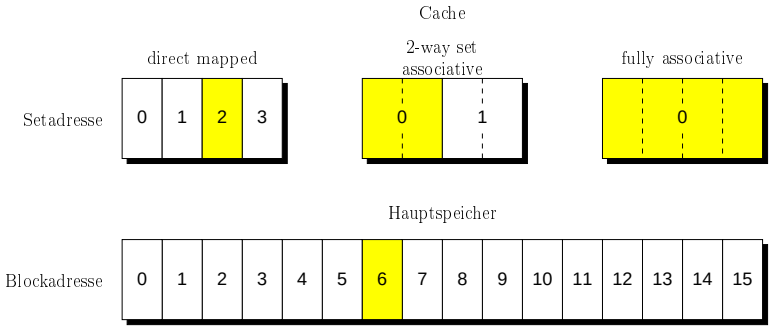

Abbildung 2.6: Beispiel für Block-Platzierung

Abbildung 2.6 gibt ein Beispiel für die verschiedenen Möglichkeiten der Block-Platzierung bei einem Cache mit 4 Cachelines und einem Hauptspeicher mit 16 Blöcken. Der Block 6 aus dem Hauptspeicher kann je nach Aufbau des Caches in einer der unterlegten Cachelines abgespeichert werden.

Block-Identifizierung

Die Zuordnung der Blöcke des Hauptspeichers auf die Sets ist nicht injektiv. Zur Identifizierung des in einer Cacheline gesicherten Blocks wird dessen Blockadresse $ba = \langle a[m-1:b] \rangle$ benötigt. Da $sa = \langle a[b+s-1:b] \rangle$ durch das Set, zu dem die Cacheline gehört, festgelegt ist, genügt es, das sogenannte *Tag* $ta = \langle a[m-1{:}b+s] \rangle$ zu kennen. Dazu wird das Tag zusammen mit dem Block in der Cacheline gespeichert.
Bei der Initialisierung des Caches, kann nicht sichergestellt werden, dass die Cachelines gültige Daten enthalten. Aus diesem Grund wird in jeder Cacheline

[1] Dies ist zwar üblich, aber nicht notwendig. Der SuperSPARC-Prozessor von Sun Microsystems beispielsweise hat einen 5-way set associative Instruktions-Cache [41].

ein *Valid-Bit* gespeichert, das angibt, ob Block-Daten und Tag gültig sind. Bei der Initialisierung des Caches müssen nur die Valid-Bits auf 0 gesetzt werden. Seien v das Valid-Bit, ta das Tag und sa die Setadresse einer Cacheline. Die Cacheline enthält genau dann die Speicherzelle mit Adresse $\langle a[m-1:0]\rangle$, wenn gilt:

$$v = 1 \ \wedge \ ta = \langle a[m-1:b+s]\rangle \ \wedge \ sa = \langle a[b+s-1:b]\rangle. \tag{2.1}$$

Cacheline-Ersetzung

Soll ein neuer Block B in den Cache geladen werden und sind alle Cachelines des zugehörigen Sets belegt, so muss eine Cacheline ersetzt werden. Im Falle eines direct mapped Cache ist die Frage, welche Cacheline ersetzt werden soll, trivial, da es nur eine Cacheline gibt, in der B gespeichert werden kann. Für set associative und fully associative Caches gibt es zwei übliche Strategien:

Zufällig Die zu ersetzende Cacheline wird zufällig ausgewählt.

Least-recently used (LRU) Die Cacheline, deren zeitlich letzter Zugriff am weitesten zurückliegt, wird ersetzt.

Das LRU-Verfahren erzielt bei den SPEC92 Benchmarks geringfügig bessere Ergebnisse [32].

Schreibzugriffs-Strategie

Wird ein Block im Cache verändert, gibt es zwei mögliche Alternativen:

Write Back Bei einem Schreibzugriff wird nur der Cache verändert. Der veränderte Block wird erst dann ins RAM geschrieben, wenn er im Cache überschrieben werden soll. Dazu ist es nötig, die Blöcke, die verändert wurden, zu markieren. Dies geschieht durch ein *Dirty-Bit*, das zusätzlich zu Valid-Bit und Tag gespeichert wird.

Write Through Bei einem Schreibzugriff werden Cache und Hauptspeicher beschrieben. Somit enthalten Cache und Hauptspeicher immer die gleichen Daten.

Bei einem Schreibzugriff werden die Daten nicht vom Prozessor benötigt. Es ist also nicht nötig, den betreffenden Block in den Cache zu laden. Dies führt zu zwei weiteren Alternativen:

Read Allocate Ein Block wird nur bei einem Lesezugriff in den Cache geladen.

Write Allocate Auch bei einem Schreibzugriff wird der Block in den Cache geladen.

Bei den SPEC92 Benchmarks erzielt im Normalfall Write Allocate bessere Ergebnisse. Nur bei Cachegrößen unter 1 KB kann die Read Allocate Strategie vorteilhaft sein [32]. Write Back und Write Through lassen sich nur vergleichen,

wenn die Dauer der Hauptspeicherzugriffe auf einzelne Zellen oder ganze Blöcke bekannt ist.

Die für diese Arbeit gewählte Implementierung wird im Abschnitt 3.1 beschrieben.

2.3 Der Prototyp

Der entwickelte Prototyp des optisch gekoppelten Caches wird auf eine Platine integriert, die momentan von Michael Klein entwickelt wird [30] (siehe Abbildung 2.7). Um nicht zusätzlich zum Cache einen ganzen Prozessor entwickeln zu müssen, wurde entschieden, einen externen – also nicht im Prozessorchip integrierten – Cache zu verwenden, auf den ein kommerzieller Prozessor zugreifen kann.

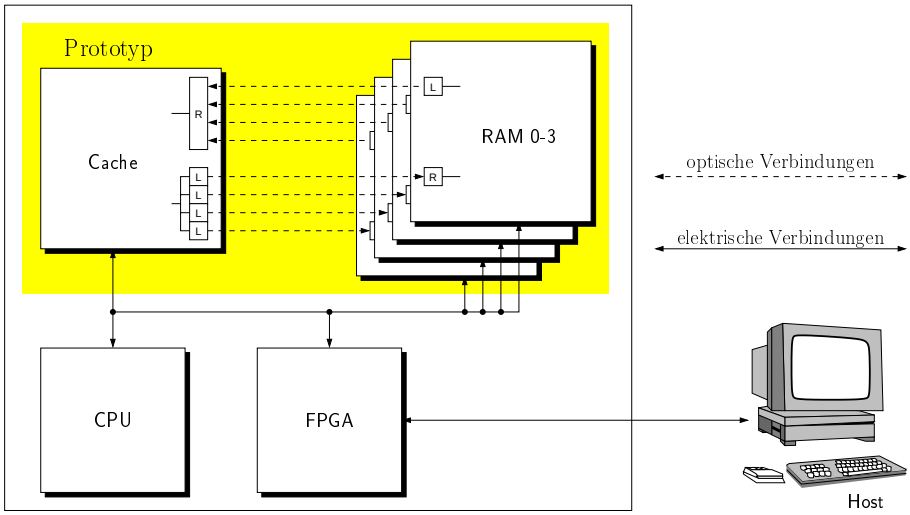

Abbildung 2.7: Die Testplatine

Der verwendete Prozessor, ein PowerPC 603e von IBM [24], ist über einen elektrischen Bus mit dem Cache verbunden. An den Cache wiederum sind über optische Verbindungen vier RAM-Chips angeschlossen. Will der Prozessor auf eine Speicherzelle zugreifen, so stellt er über den elektrischen Bus eine Anfrage an den Cache. Befinden sich die Daten nicht im Cache (Cache-Miss), lädt der Cache den entsprechenden Block über die optische Verbindung aus einem der vier RAM-Chips nach.

Damit der Cache mit den vier RAM-Chips kommunizieren kann, ist es nötig, das optische Signal vom Cache zu den RAM-Chips aufzuspalten und auf dem Rückweg wieder zusammenzuführen. Die Aufspaltung in Richtung des RAMs wird durch vier parallel geschaltete Laser realisiert, an die jeweils ein Glasfaserkabel angeschlossen ist. In Richtung Cache werden die vier Glasfaserkabel der RAM-Chips auf einen gemeinsamen Receiver gerichtet. Dabei wird ein logisches ODER der vier gesendeten Signale berechnet [40]. Damit sich die Signale nicht überlagern, darf zu jedem Zeitpunkt maximal einer der vier RAM-Chips ein

Signal ungleich Null senden. Dies entspricht dem “wired or” bei elektrischen Signalen [29, S. 283].
Jede optische Verbindung besteht aus 11 Glasfaserkabeln. Eine Glasfaser dient zur Übertragung des Clocksignals (siehe Abschnitt 3.3), die restlichen 10 Signal zur Übertragung der Daten und Adressen. Da ein Block aus mehr als 10 Bits bestehen soll, ist es nötig, mehrere Bits über eine Glasfaser zu versenden. Die Daten müssen also vor der Übertragung sequenzialisiert und danach wieder parallelisiert werden.

Um das Debugging zu erleichtern, soll es zusätzlich zum Zugriff über den Cache möglich sein, die RAM-Chips direkt über elektrische Pins zu beschreiben und auszulesen. Dazu sind die RAM-Chips an den elektrischen Bus zwischen Prozessor und Cache angeschlossen. Ob der Zugriff direkt oder über den Cache stattfindet, wird über die Adresse der Speicherzelle kodiert (siehe Abschnitt 4.7.2). Im späteren Testbetrieb soll der auf der Platine integrierte Prozessor nur auf den Adressraum des Caches zugreifen. Um den Inhalt des Caches und der RAM-Chips von außen setzten und lesen zu können, kann auf den elektrischen Bus zusätzlich von einem Host-Computer aus zugegriffen werden.

2.4 Verwendete Technologie

2.4.1 Aufbau der Glasfaser-Chips

Die zur Realisierung der optischen Verbindung benötigten Laser und Receiver werden in einer anderen Technologie (GaAs, [20]) als die interne Logik (Si, [6]) gefertigt. Es ist also nicht möglich, Glasfaseranbindung und Cachelogik auf dem gleichen Die zu integrieren. Zur Verbindung der Dies für die optische Anbindung und der internen Logik gibt es zwei Möglichkeiten: das sogenannte Flip-Chip-Bonding [21] und die Verwendung eines Multichip-Moduls.
Beim Flip-Chip-Bonding werden die beiden Dies übereinander gelegt und können durch Lötstellen auf der ganzen Fläche elektrisch miteinander verbunden werden. Diese Technologie wird allerdings nur von sehr wenigen Firmen (zum Beispiel Lucent Technologies [35]) beherrscht und stand nicht zur Verfügung.

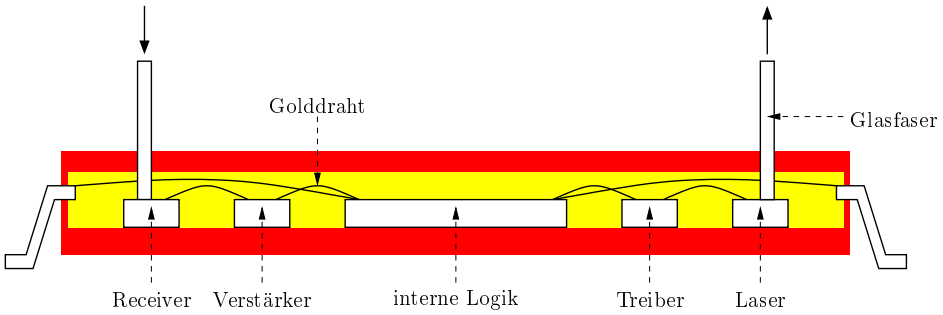

Abbildung 2.8: Schematischer Querschnitt durch einen Glasfaser-Chip

Aus diesem Grund wurde entschieden, die Glasfaser-Chips als Multichip-Modul zu realisieren (siehe Abbildung 2.8). Die interne Logik ist innerhalb des Gehäu-

ses durch feine Golddrähte mit den Dies zur Anbindung der Glasfasern und den elektrischen Pins verbunden.

2.4.2 Herstellung der Dies

Für das Design der internen Logik waren die folgenden Punkte zu beachten:

- Der Die mit der internen Logik soll innerhalb des Gehäuses mit den Lasern und Receivern verbunden werden. Dazu ist es nötig, dass der Die ungehäust zur Verfügung steht.

- Die Laser sollen mit 320 MHz betrieben werden. Der Die muss also in der Lage sein, die zu übertragenden Daten in der entsprechenden Geschwindigkeit über die Pads zu liefern.

Programmierbare Bausteine, die den zweiten Punkt erfüllen, werden momentan von der Firma Xilinx mit der Virtex FPGA-Serie hergestellt [42]. Da Xilinx diese Chips nicht ungehäust liefert, war es nicht möglich, für die interne Logik einen programmierbaren Baustein zu benutzen. Für die interne Logik musste also ein eigener Die – ein sogenannter ASIC (application specific integrated circuit) – entwickelt werden. Dies hatte zwar den Vorteil, dass der Die genau auf die Bedürfnisse angepasst werden konnte, allerdings auch mehrere schwerwiegende Nachteile:

- Bei der Entwicklung kleiner Serien von Dies sind die Stückkosten aufgrund einmaliger Kosten bei der Herstellung sehr hoch.

- Nachträgliche Änderungen am Design sind nicht möglich. Ein schwerwiegender Fehler hat zur Konsequenz, dass ein komplett neuer Die entwickelt werden muss.

- Für die Entwicklung von ASICs war am Lehrstuhl von Professor Paul keinerlei Erfahrung und Software vorhanden. Die gesamte Infrastruktur zur ASIC-Entwicklung musste also im Rahmen dieser Diplomarbeit neu aufgebaut werden. Dies führte zu einer sehr langen Entwicklungszeit von $2\frac{1}{2}$ Jahren.

Eine Möglichkeit zu verhältnismäßig günstigen Konditionen kleine Serien von ASICs herzustellen, wird von der Organisation Europractice angeboten [18]. Europractice sammelt ASIC-Designs und gibt sie zusammen an die ASIC-Hersteller weiter. Auf diese Weise können die einmaligen Kosten auf die verschiedenen Designs aufgeteilt werden.
Aufgrund der trotzdem noch hohen Kosten von circa € 25.000 wurde entschieden, nicht für Cache und RAM-Chips jeweils einen eigenen ASIC zu entwickeln. Stattdessen soll für Cache und RAM-Chip der gleiche ASIC eingesetzt werden. Ob der Chip als Cache oder RAM arbeiten soll, wird durch ein Konfigurations-Signal eingestellt (siehe Kapitel 4).

2.4.3 Gatter-Technologie

Für die ASIC-Herstellung werden mehrere Prozesse angeboten. Diese Prozesse ermöglichen Designs in den Gatter-Technologien CMOS und ECL zu erstellen. CMOS ist die momentan in der Chipentwicklung meistgenutzte Gatter-Technologie, da sie sehr platzsparend ist. Zusätzlich wird nur während der Schaltvorgänge Strom verbraucht, so dass CMOS bei niedrigen Frequenzen sehr stromsparend ist. Bei modernen Chips [1, 26] werden mit CMOS zwar innerhalb der Chips Frequenzen bis 1,5 GHz erreicht, allerdings über die Pads nur Frequenzen bis zu 133 MHz. Insbesondere war es bei keinem der angebotenen Prozesse möglich, mit CMOS die Pads mit 320 MHz zu betreiben.
In der Gatter-Technologie ECL sind prinzipiell Frequenzen im GHz-Bereich nicht nur innerhalb des Chips, sondern auch über die Pads möglich. Allerdings verbraucht ECL viel Platz und viel Strom. Insbesondere ist es mit vertretbarem Aufwand nicht möglich, die gesamte Cache-Logik inklusive dem RAM in ECL zu realisieren.

Würde man nur eine der beiden Technologien verwenden, so wäre es also nicht möglich, einen ASIC mit den gewünschten Anforderungen zu entwickeln. Allerdings bietet die Firma AMS im Rahmen von Europractice den 0.8 μm BiCMOS Prozess [6] an, der es ermöglicht, die beiden Technologien auf einem Die zu kombinieren.
Bei diesem Prozess sind allerdings in CMOS auch innerhalb des Chips nur Frequenzen bis circa 100 MHz möglich [5]. Insbesondere funktionieren die Zellen zur Konvertierung der Signale zwischen CMOS und ECL nur bis circa 80 MHz [3]. Es ist also nicht möglich, den gesamten Chip mit 320 MHz zu betreiben. Aus diesem Grund wird die eigentliche Logik mit einer niedrigen Frequenz von 10 MHz betrieben und nur die Schaltkreise zum Sequenzialisieren und Parallelisieren mit 320 MHz getaktet.

Kapitel 3

Designüberblick

Dieses Kapitel behandelt das grundlegende Design und allgemeine Probleme des entwickelten Dies. Zunächst wird in Abschnitt 3.1 der allgemeine Aufbau des Caches beschrieben. In Abschnitt 3.2 wird das Design des Dies in drei Teile aufgeteilt, auf die in den folgenden Kapiteln näher eingegangen wird. Abschnitt 3.3 behandelt das Problem der Verteilung der ECL-Clock. Auf die Probleme beim Übertragen von Daten zwischen Bereichen mit unterschiedlichen Clocksignalen wird in Abschnitt 3.4 eingegangen.

3.1 Aufbau des Caches

In der verwendeten Technologie stehen nur RAM-Bausteine zur Verfügung, die bei einem Schreibvorgang immer eine ganze Zeile verändern [4]. Es ist beispielsweise nicht möglich, einen 4 Byte breiten Baustein einzusetzen, der byteweise beschreibbar ist. Ein solches RAM muss aus vier je 1 Byte breiten RAM-Bausteinen zusammengesetzt werden. Da jeder dieser RAM-Bausteine eine eigene Adresslogik enthält, belegen sie zusammen wesentlich mehr Platz als ein einzelner 4 Byte breiter RAM-Baustein. Aus Platzgründen sollte man daher darauf achten, dass der Speicher nicht aus vielen separat zu beschreibenden Teilen besteht.

Um die Bandbreite der Glasfaserkabel auszunutzen, soll innerhalb eines Taktes ein kompletter Block zwischen Cache und Hauptspeicher übertragen werden. Dazu muss im Cache und im Hauptspeicher der komplette Block parallel aus dem Speicher-Baustein ausgelesen werden. Es ist also nicht möglich, die Blöcke über mehrere Zeilen der Speicher-Bausteine zu verteilen. Andererseits muss der Prozessor im Cache die Speicherzellen des Blocks einzeln beschreiben können. Da für Cache und Hauptspeicher der gleiche Die benutzt werden soll, gilt diese Einschränkung auch für den Hauptspeicher.
Um mit einer angemessenen Zahl von RAM-Bausteinen eine hohe Blockgröße implementieren zu können, ist der entwickelte Chip nur word-weise (ein Word entspricht vier Byte) adressierbar. Bei jedem Zugriff über die Pins wird ein komplettes Word entweder geschrieben oder gelesen. Die unteren beiden Bits der Adressen werden also nicht benötigt. Ein Block besteht aus vier Words, also

16 Bytes. Um einen Block zu speichern, werden also vier RAM-Bausteine mit einer Breite von jeweils 32 Bit benötigt.

Um eine byte-weise Adressierung des Speichers zu erlauben, würde man statt den 4 RAM-Bausteinen der Breite 32 Bit 16 RAM-Bausteine der Breite 8 Bit benötigen. Dies würde zusammen mit den zwei zusätzlich benötigten Adresspins die Kosten für den Chip um circa € 4000 erhöhen.

Das Tag einer Cacheline zu beschreiben ist nur dann nötig, wenn die Cacheline gültig ist, also das Valid-Bit auf 1 gesetzt wird. Umgekehrt darf nur dann das Valid-Bit auf 1 gesetzt werden, wenn gleichzeitig ein gültiges Tag geschrieben wird. Soll das Valid-Bit auf 0 gesetzt werden, so ist der Inhalt des Tags unerheblich, es kann also ein beliebiger Wert geschrieben werden. Tag und Valid-Bit können also immer gemeinsam geschrieben werden und damit im gleichen RAM-Baustein gespeichert werden.

Ein K-way set associative Cache besteht im wesentlichen aus K direct mapped Caches [37]. Um einen K-way set associative Cache zu implementieren, werden folglich $K \cdot (4 + 1)$ RAM-Bausteine benötigt. Ist $K > 1$, muss zusätzlich eine aufwendige Kontrolle zur Ersetzung der Cachelines implementiert werden. Wird die LRU-Strategie verwendet, so wird ein zusätzlicher RAM-Baustein benötigt, der das Zugriffsmuster auf den Cache speichert. Obwohl die Hit-Rate eines Caches mit K steigt [38], wurde entschieden, einen direct mapped Cache zu implementieren, also $K = 1$.

Die Write-Back- und Read-Allocate-Strategien führen jeweils zu einer geringeren Auslastung der Verbindung zwischen Cache und Hauptspeicher. Da diese Verbindung aber den wesentlichen Teil des Projektes darstellt, ist dies nicht erwünscht. Write Through erleichtert den Test der Verbindung, da die Daten in Cache und Hauptspeicher immer kohärent gehalten werden. Write Allocate vereinfacht die Kontrolle, da immer ganze Blöcke in den Hauptspeicher geschrieben werden. Folglich wurde eine Write-Through- und Write-Allocate-Strategie verwendet.

Mit diesen Vorgaben läßt sich in der verwendeten Technologie mit angemessenen Kosten ein Cache mit $2^s = 2^6 = 64$ Cachelines realisieren. Bei einer verwendeten Blockgröße von $2^b = 2^4 = 16$ Bytes entspricht dies einer Cachegröße von 2^{10} Bytes = 1 KB. Der Datenspeicher ist aufgeteilt in vier RAM-Bausteine der Größe 64×32 Bit.

Da der Hauptspeicher aus vier Dies mit dem gleichem Design wie der Cache besteht, ergibt sich eine Hauptspeichergröße von $2^m = 2^{12}$ Bytes = 4 KB. Für das Tag werden also $m - (b + s) = 12 - (6 + 4) = 2$ Bits benötigt (siehe Seite 8). Zusammen mit dem Valid-Bit wird für das TagValid-RAM also ein RAM-Baustein der Größe 64×3 benötigt. Die Cacheline mit Set-Adresse sa wird in der Zeile sa von Daten und TagValid-RAM gespeichert.

Die Datenpfade des implementierten Caches werden in Abschnitt 4.2 beschrieben. Zum Design der weiteren Alternativen sei auf [37] verwiesen.

3.2 Aufteilung des Designs

Das Layout des Dies läßt sich in drei Teile gliedern: Die Cache/RAM-Logik, den Sequenzer und den Parallelisierer (siehe Abbildung 3.1). Die Cache/RAM-Logik enthält neben den Schaltkreise zur Realisierung der Cache- und RAM-Funktion des Chips (inklusive des Speichers) die Anbindung an die elektrischen Pins. Die Datenpfade der Cache/RAM-Logik werden in Kapitel 4 beschrieben.

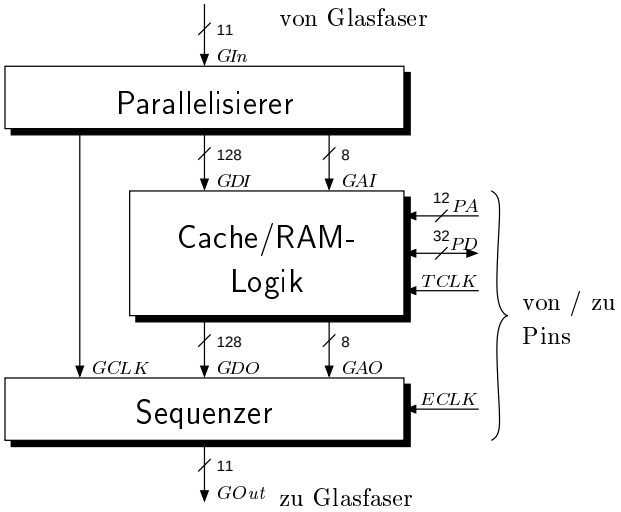

Abbildung 3.1: Schematischer Überblick

Sequenzer und Parallelisierer steuern die Glasfaser-Übertragung. Um die von der Cache/RAM-Logik gelieferten Adress- und Datenbusse (*GAO* und *GDO*) parallel zu übertragen, stehen nicht genügend Glasfaserkabel zur Verfügung. Die Hauptaufgabe des Sequenzers besteht darin, die parallelen Daten zu sequenzialisieren und an die einzelnen Glasfasern zu verteilen. Der Parallelisierer wandelt die empfangenen Signale wieder in die parallelen Busse *GAI* und *GDI* um. Eine genaue Beschreibung der Glasfaser-Übertragung sowie die Datenpfade von Sequenzer und Parallelisierer findet sich in Kapitel 5.

Die Cache/RAM-Logik wird mit der relativ niedrigen Frequenz von 10 MHz betrieben und kann deshalb in der kostengünstigen CMOS-Technologie realisiert werden (siehe Abschnitt 2.4.3). Das entsprechende Clocksignal $TCLK$ wird an allen Chips des Prototyps über die Pins angelegt. Die Verteilung an die fünf Chips erfolgt auf dem Board.
Die Busse *GOut* und *GIn* hingegen sollen mit 320 MHz getaktet werden. Dazu müssen Sequenzer und Parallelisierer zumindest teilweise in der teuren ECL-Technologie (siehe Abschnitt 2.4.3) realisiert werden.

3.3 Verteilung der ECL-Clock

Im Rahmen dieses Projektes war es nicht möglich, ein Clocksignal auf dem Die zu generieren. Das 320 MHz Clocksignal muss also von außen dem Chip zugeführt werden. Eine elektrische Clockverteilung auf der Platine ist jedoch

bei 320 MHz auch mit ECL-Technologie sehr aufwendig [27]. Deshalb wurde entschieden, nur den Cache-Chip über die Pins mit einem 320 MHz Clocksignal zu versorgen. Dadurch wird es ermöglicht, den Clockgenerator nahe am Cache-Chip zu platzieren und das Problem langer Leitungswege auf dem Board zu umgehen [30]. Die Verteilung des 320 MHz Clocksignals an die RAM-Chips erfolgt über die Glasfaser-Verbindung.

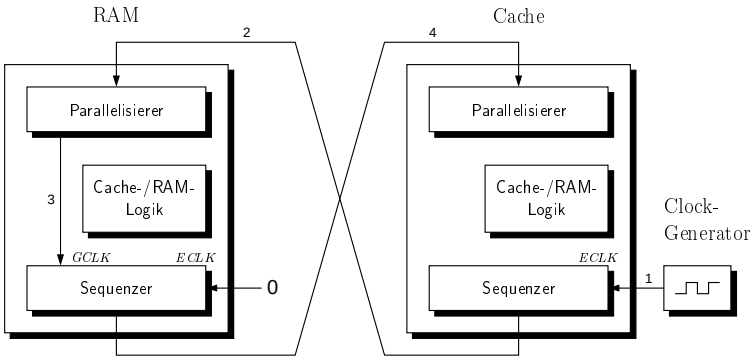

Abbildung 3.2: Verteilung der ECL-Clock

Abbildung 3.2 skizziert die Verteilung der ECL-Clock. Der Sequenzer des Caches erhält über die Pins das Clocksignal *ECLK* (1). Dieses Signal wird parallel zu den Daten vom Cache-Chip an die RAM-Chips gesendet und von den Parallelisierern der RAM-Chips benutzt (2). Die Sequenzer der RAM-Chips erhalten über den Pin-Eingang *ECLK* ein konstantes Signal. Stattdessen benutzten sie als Clocksignal das vom Parallelisierer empfangene Signal *GCLK* (3).
Das in den Sequenzern der RAM-Chips verwendete Clocksignal *GCLK* ist auf Grund der Verzögerungen der Leitungen nicht mehr synchron zum ursprünglichen Clocksignal *ECLK* im Cache-Chip. Das ursprüngliche Signal kann also nicht benutzt werden, um im Parallelisierer des Cache-Chips die von den RAM-Chips gesendeten Daten zu empfangen. Das Clocksignal muss demnach auch auf dem Rückweg vom RAM-Chip zum Cache parallel zu den Daten übertragen werden (4).
Da die Verzögerungen der einzelnen Dies produktionsbedingt nicht exakt gleich sind, kann zusätzlich nicht davon ausgegangen werden, dass die vier verschiedenen Clocksignale in den Sequenzern der RAM-Chips zueinander synchron sind. Der Parallelisierer des Cache-Chips muss also jeweils das Signal von dem RAM-Chip verwenden, der gerade Daten sendet. Überträgt ein RAM-Chip Daten, sendet er parallel das Clocksignal des Sequenzers. Die anderen RAM-Chips senden eine konstante Null. Bei der Glasfaser-Übertragung wird ein logisches ODER der vier Signale berechnet (siehe Abschnitt 2.3). Auf diese Weise verwendet der Parallelisierer des Caches immer das passende Clocksignal.

3.4 Synchronisation

Wie aus den beiden letzten Abschnitten hervorgeht, werden in den drei Teilen des Dies unabhängige Clocksignale benutzt. Dies führt vor allem bei der

Datenübertragung von der Cache/RAM-Logik zum Sequenzer zu Problemen. Abbildung 3.3 zeigt eine Daten-Übertragung, wie sie zwischen Cache/RAM-Logik und Sequenzer vorkommen kann.

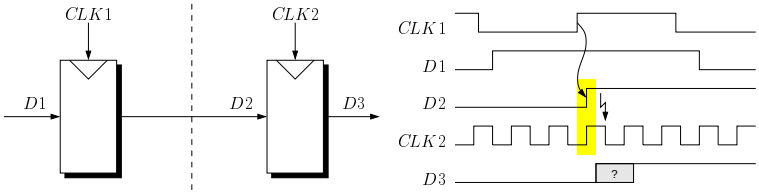

Abbildung 3.3: Übertragung zwischen Registern mit unabhängigen Clocksignalen

Das Signal $D2$ ändert sich mit der positiven Flanke des langsamen Clocksignals $CLK1$. Es ist keine Aussage über das zeitliche Verhalten von $D2$ im Verhältnis zur schnellen Clock $CLK2$ möglich. Man sagt auch, $D2$ sei *asynchron* zu $CLK2$. Insbesondere ist es möglich, dass sich $D2$ – wie im Timing angedeutet – genau zur steigenden Flanke von $CLK2$ ändert. Ist dies der Fall, kann nicht genau bestimmt werden, ob und wann sich der Ausgang $D3$ des Registers ändert. Eine Aussage über den Wert von $D3$ ist erst nach der nächsten steigenden Flanke von $CLK2$ möglich, da zu diesem Zeitpunkt der Register-Eingang $D2$ stabil ist. Das Signal $D3$ ist also auch asynchron zu $CLK2$. Da keine Aussagen über das Verhalten von $D3$ in Abhängigkeit von $CLK2$ gemacht werden kann, ist dies auch nicht für die von $D3$ abhängigen Signale möglich.

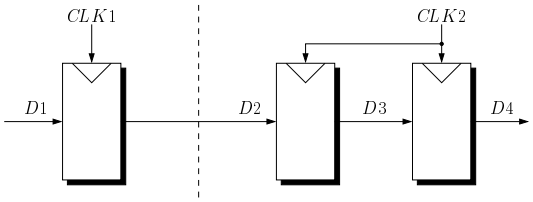

Abbildung 3.4: Synchronisation

Um dennnoch Aussagen über das zeitliche Verhalten von Schaltungen machen zu können, die asynchrone Signale verarbeiten, müssen diese Signale zunächst synchronisiert werden. Dazu wird die in Abbildung 3.4 beschriebene Schaltung benutzt [33, S. 554], [28, S. 310], [29, S. 338].

Die Wahrscheinlichkeit, dass die unbestimmte Verzögerung von $D3$ zu Problemen führt, sinkt, je niedriger die Frequenz von $CLK2$ ist. Zusätzlich steigt die zusätzliche Dauer durch die Synchronisation. Da zwischen Parallelisierer und Cache/RAM-Logik mit der langsamen Clock $TCLK$ synchronisiert werden muss, kann hier die Synchronisation abgeschaltet werden (siehe Abschnitt 5.4.2).
Zwischen Parallelisierer und Cache/RAM-Logik sowie zwischen Cache/RAM-Logik und Sequenzer werden jeweils allein 128 Datensignale übertragen. Alle Signale zu synchronisieren wäre vor allem im Sequenzer zu teuer, da die Register in ECL-Technologie realisiert werden müssten. Aus diesem Grund wird

jeweils nur ein Steuersignal synchronisiert, das die Gültigkeit der anderen Signale anzeigt.

Kapitel 4

Die Cache/RAM-Logik

In diesem Kapitel wird auf das Design der Cache/RAM-Logik näher eingegangen. Die Cache/RAM-Logik soll je nach Konfiguration als Cache oder RAM arbeiten. Abschnitt 4.1 beschreibt die Eingangs- und Ausgangs-Signale der Cache/RAM-Logik. In den Abschnitten 4.2 und 4.3 werden die zur Realisierung der Cache- beziehungsweise der RAM-Funktion notwendigen Datenpfade beschrieben. Diese Datenpfade werden in Abschnitt 4.4 zu den Datenpfaden der Cache/RAM-Logik zusammengefasst.
Auf die Protokolle zur Kommunikation über die Pins beziehungsweise mit Sequenzer und Parallelisierer wird in den Abschnitten 4.5 und 4.6 eingegangen. Die Kontrolle der Cache/RAM-Logik wird in Abschnitt 4.7 beschrieben. Neben dem Normalbetrieb unterstützt die Cache/RAM-Logik noch einen Testmodus zum Testen der Glasfaser-Verbindung. Auf den Testmodus wird in Abschnitt 4.8 näher eingegangen.

4.1 Eingangs- und Ausgangs-Signale

Die Cache/RAM-Logik beinhaltet die eigentliche Logik zur Realisierung der Cache- beziehungsweise RAM-Funktion inklusive der Speicher-Bausteine. Ob sich der Chip als Cache oder RAM verhalten soll, wird durch das über die Pins angelegte Signal *CMod* (siehe Anhang A) signalisiert. Um die Cache-Funktion einzustellen, wird *CMod* auf den Wert 1 gesetzt; für die RAM-Funktion wird *CMod* auf 0 gesetzt.
Die Cache/RAM-Logik hat nach außen drei Paare von Adress- und Datenbussen. Zugriffe über die Pins werden durch den Adressbus $PA[13:2]$ gesteuert (die Bits $PA[1:0]$ werden nicht benötigt, da immer word-weise zugegriffen wird). Die zugehörigen Daten werden über den bidirektionalen Bus $PD[31:0]$ ausgetauscht. Die obersten beiden Bits $PA[13:12]$ der Adressen werden nur von der Kontrolle der Cache/RAM-Logik (siehe Abschnitt 4.7) benutzt und tauchen deshalb in den Datenpfaden nicht auf.
Sollen Adressen beziehungsweise Daten über die Glasfasern gesendet werden, so geschieht dies über die Busse $GAO[11:4]$ und $GDO[127:0]$. Adressen oder Daten, die über die Glasfasern empfangen wurden, liegen an den Bussen $GAI[11:4]$ und $GDI[127:0]$ an. Da über die Glasfasern immer ganze Blöcke gesendet wer-

den, werden die Bits 3 und 2 der Adressen nicht benötigt.

4.2 Datenpfade Cache

Abbildung 4.1 zeigt die Datenpfade zur Realisierung der Cache-Funktion. Der Adressbus $GAI[11:4]$ wird nicht benötigt, da der Cache keine Anfragen von den RAM-Chips erhält und das RAM daher keine Adressen an den Cache sendet. Die Datenpfade lassen sich in drei Bereiche unterteilen: die Pin-Steuerung, den Block-Speicher und die Hit-Berechnung.
Die Pin-Steuerung dient zur Anbindung der Pins und wandelt den internen 128 Bit breiten Bus in den 32 Bit breiten Bus *PD* der Pins um. Der Block-Speicher steuert den Zugriff auf das Daten-RAM. In der Hit-Berechnung wird überprüft, ob sich die Speicherzelle mit der über die Pins angelegten Adresse im Cache befindet. In den folgenden Abschnitten wird genauer auf die einzelnen Bereiche eingegangen.

Bemerkung: Im Folgenden muss zwischen den beiden Begriffen *RAM-Chip* und *Daten-RAM* unterschieden werden. Mit dem Begriff Daten-RAM werden die RAM-Bausteine bezeichnet, die innerhalb der Chips die Daten speichern. Das Daten-RAM wird sowohl im Cache als auch in den RAM-Chips verwendet. Mit RAM-Chip werden die als Hauptspeicher konfigurierten Chips bezeichnet.

Alle in den Datenpfaden vorkommenden Kontrollsignale sind standardmäßig inaktiv. Generell wird für ein Signal nur näher beschrieben, wann es aktiviert wird. Insbesondere sind die Schreibsignale der Speicher-Bausteine im Normalfall inaktiv, das heißt, wenn ein Baustein nicht explizit beschrieben wird, wird er ausgelesen.

4.2.1 Pin-Steuerung

Die Pin-Steuerung trennt mit Hilfe von Registern die Datenpfade der Platine und der inneren Logik. Dem Adressbus *PA* entspricht der interne Bus *PAdr*. Der bidirektionale Datenbus *PD* wird durch die Register in den Eingangsbus *PDI* und Ausgangsbus *PDO* aufgeteilt. Da der Block-Speicher mit 128 Bit breiten Daten-Bussen arbeitet, müssen die Busse *PDI* und *PDO* zunächst an die 128 Bit breiten Busse $PDI4$ und $PDO4$ angepasst werden. Für den Eingangsbus $PDI4$ gilt:

$$PDI4[128:0] = (PDI[31:0])^4. \tag{4.1}$$

Soll also über die Pins ein Word in das Daten-RAM geschrieben werden, so liegt dies an allen vier Words des Blockes an. Auf das Beschreiben des Daten-RAMs wird in Abschnitt 4.2.2 näher eingegangen.
Sollen Daten über die Pins ausgegeben werden, so muss aus dem 128 Bit breiten Bus $PDO4$ ein Word selektiert werden. Die Position des Words wird durch die Adressbits $PAdr[3:2]$ spezifiziert. Sei $W_A = 32 \cdot \langle PAdr[3:2] \rangle$ und B der Block

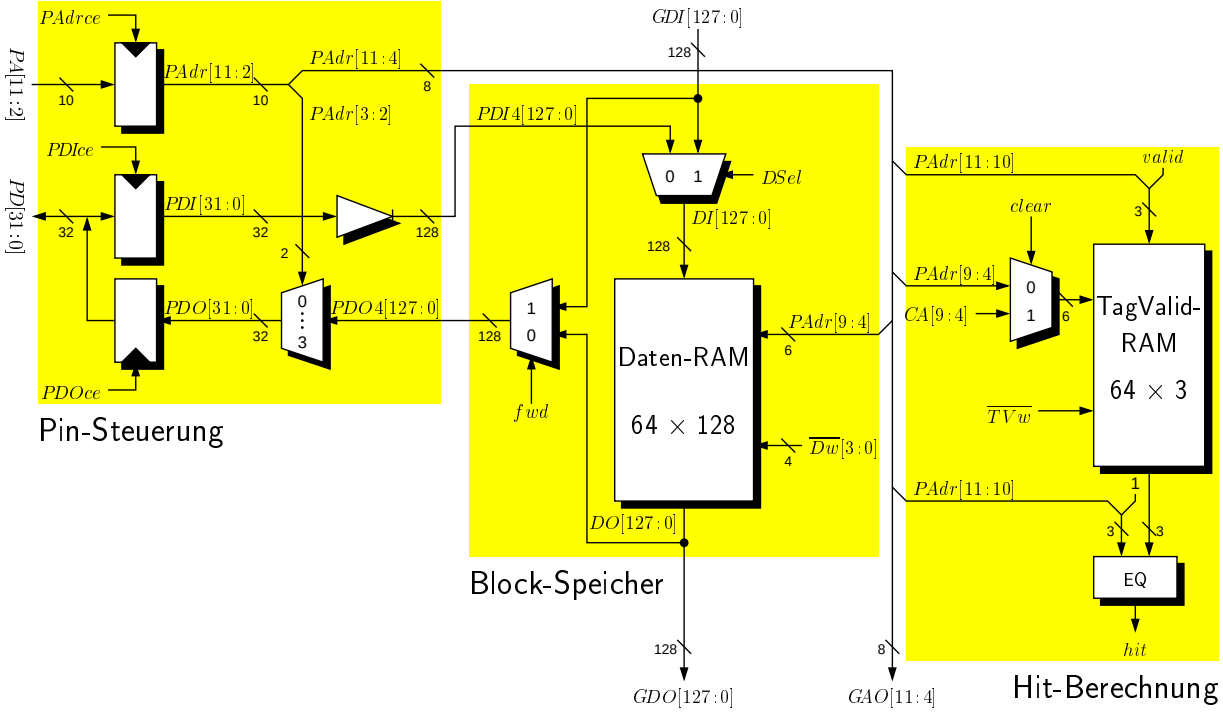

Abbildung 4.1: Datenpfade des Cache-Logik

mit Adresse $\langle PAdr[11:4]\rangle$. Dann enthalten die Bits $B[W_A+31:W_A]$ das Word mit Adresse $\langle PAdr[11:2]\rangle$. Entsprechend gilt:

$$PDO[31:0] = PDO4[W_A+31:W_A].$$

Die Auswahl wird durch einen 32 Bit breiten 4:1 Mux realisiert.

4.2.2 Block-Speicher

Der Block-Speicher muss zwei Arten von Zugriffen unterstützen: den Zugriff auf einzelne Speicherzellen über die Pins und das Nachladen oder Rückschreiben eines Blocks über die Glasfaser-Verbindung. Entsprechend hat der Blockspeicher jeweils zwei Dateneingangs- und Datenausgangs-Busse. Für Zugriffe über die Pins dienen die Busse $PDI4$ und $PDO4$. Die über die Glasfasern empfangenen Daten liegen am Bus GDI an, über den Bus GDO werden Daten an die Glasfasern gesendet. In jedem Fall wird der Adressbus $PAdr$ verwendet.
Der Blockspeicher verarbeitet die Kontrollsignale $DSel$, fwd und $Dw[3:0]$. Mit dem Signal $DSel$ kann zwischen Dateneingangs-Bussen $PDI4$ und GDI ausgewählt werden. Wird fwd aktiviert, kann ein nachgeladener Block an die Pins geforwarded werden. Tritt ein Cache-Miss bei einem Lesezugriff des Prozessors auf, muss also nicht auf das Beschreiben des Daten-RAMs gewartet werden, bevor die Daten ins Datenausgangs-Register geclockt werden. Der Bus $\overline{Dw}$ enthält die Schreibsignale für das Daten-RAM.

Sei $sa = \langle PAdr[9:4]\rangle$ die am Block-Speicher angelegte Setadresse. Im Normalfall sind die Schreibsignale $\overline{Dw}[3:0]$ alle inaktiv ($\overline{Dw}[3:0] = 1^4$), das Daten-RAM

D gibt also die Zeile $D[sa]$ aus. Die Zeile liegt direkt am Bus *GDO* an und kann zum Rückschreiben des Blockes über die Glasfasern verwendet werden. Solange fwd nicht aktiviert wird, liegt die Zeile zusätzlich an der Pin-Steuerung an und kann somit auch über die Pins ausgegeben werden.
Um beim Nachladen den vom Hauptspeicher empfangenen Block mit Setadresse sa ins Daten-RAM zu schreiben, muss der vom Parallelisierer kommende Datenbus *GDI* am Daten-RAM angelegt werden. Dazu wird das Signal *DSel* aktiviert. Durch die Aktivierung aller Schreibsignale ($\overline{Dw}[3:0] = 0^4$) folgt:

$$D[sa][127:0] := GDI[127:0].$$

Zum Schreiben einer einzelnen Speicherzelle muss genau das Schreibsignal aktiviert werden, dessen zugehöriger RAM-Baustein durch die unteren Adressbits selektiert wird:

$$\overline{Dw}[i] = \begin{cases} 0 & \text{falls } i = \langle PAdr[3:2]\rangle, \\ 1 & \text{sonst.} \end{cases}$$

Die Berechnung der $\overline{Dw}$ geschieht durch einen Decoder in der Kontrolle. Ist $DSel = 0$, so gilt mit Gleichung (4.1) $DI[127:0] = (PDI[31:0])^4$ und somit für $W_i = 32 \cdot i$:

$$D[sa][W_i + 31:W_i] := \begin{cases} PDI[31:0] & \text{falls } i = \langle PAdr[3:2]\rangle, \\ D[sa][W_i + 31:W_i] & \text{sonst.} \end{cases}$$

Die an den Pins anliegenden Daten $PD[31:0]$ werden also genau in die durch die Adresse $PA[9:2]$ selektierte Speicherzelle geschrieben.

4.2.3 Hit-Berechnung

Die Hit-Berechnung prüft mit Hilfe des TagValid-RAMs TV, ob ein zur anliegenden Adresse $PAdr[11:4]$ gehörender Block bereits im Cache gespeichert ist. Ist dies der Fall, wird das Ausgangssignal *hit* aktiviert. Die Steuerung von TV erfolgt durch das Schreibsignal $\overline{TVw}$ und das Valid-Bit *valid*. Der Adressbus $CA[9:4]$ und das Signal *clear* dienen zur Initialisierung von TV.
Bei der Initialisierung des Caches müssen die Valid-Bits von TV auf 0 gesetzt werden (siehe Seite 9). Es ist in der verwendeten Technologie nicht möglich, einen löschbaren RAM-Baustein zu verwenden. Das TagValid-RAM muss also durch zeilenweises Beschreiben gelöscht werden. Gilt $\overline{TVw} = 0$, $valid = 0$ und $clear = 1$, wird das Valid-Bit in Zeile $\langle CA[9:4]\rangle$ auf 0 gesetzt. Zum Löschen des gesamten TagValid-RAMs muss der von der Kontrolle gesteuerte Adressbus *CA* von 0 bis 63 hochgezählt werden. Dies wird von der Kontrolle während der Initialisierung gemacht (siehe Abschnitt 4.7.1). Nach der Initialisierung wird die Adresse $CA[9:4]$ nicht mehr benötigt.

Sind keine Kontrollsignale aktiv, so berechnet das TagValid-RAM das Signal *hit*. Das Signal *hit* wird aktiviert, falls sich der Block mit der anliegenden Adresse $PAdr[11:4]$ im Cache befindet. Dies ist genau dann der Fall, wenn das Tag $PAdr[11:10]$ der Adresse mit dem Tag der Cacheline mit Setadresse $PAdr[9:4]$

übereinstimmt und das Valid-Bit der Cacheline auf 1 gesetzt ist (siehe Gleichung 2.1, Seite 9). Zusammengefasst ergibt sich die folgende Bedingung:

$$hit = 1 :\Longleftrightarrow TV[\langle PAdr[9:4]\rangle] = (1, PAdr[11:10]). \tag{4.2}$$

Zum Überprüfen dieser Voraussetzung wird die Zeile $TV[\langle PAdr[9:4]\rangle]$ ausgelesen. Der Ausgang des TagValid-RAMs wird mit Hilfe eines 4-Bit Equality-Tester EQ mit dem Vektor $(1, PAdr[11:10])$ verglichen. Der Ausgang *hit* von EQ erfüllt somit die Formel (4.2).

Wird ein neuer Block in den Cache geschrieben, muss das TagValid-RAM aktualisiert werden. Dazu werden das Schreibsignal $\overline{TVw}$ und das Signal *valid* aktiviert. Es gilt dann $hit = 1$ für die anliegende Adresse $PAdr[11:2]$ des neuen Blockes.

4.3 Datenpfade RAM

Die Datenpfade für die RAM-Logik unterstützen zwei Zugriffs-Modi: neben den Zugriffen vom Cache über die Glasfaser-Verbindung kann zu Debugging-Zwecken auch direkt über die Pins auf das Daten-RAM zugegriffen werden (siehe Abschnitt 2.3). Für die Datenpfade der RAM-Logik wird der Adressausgangs-Bus *GAO* nicht benötigt, da die RAM-Chips keine Anfragen über die Glasfaser-Verbindung stellen.

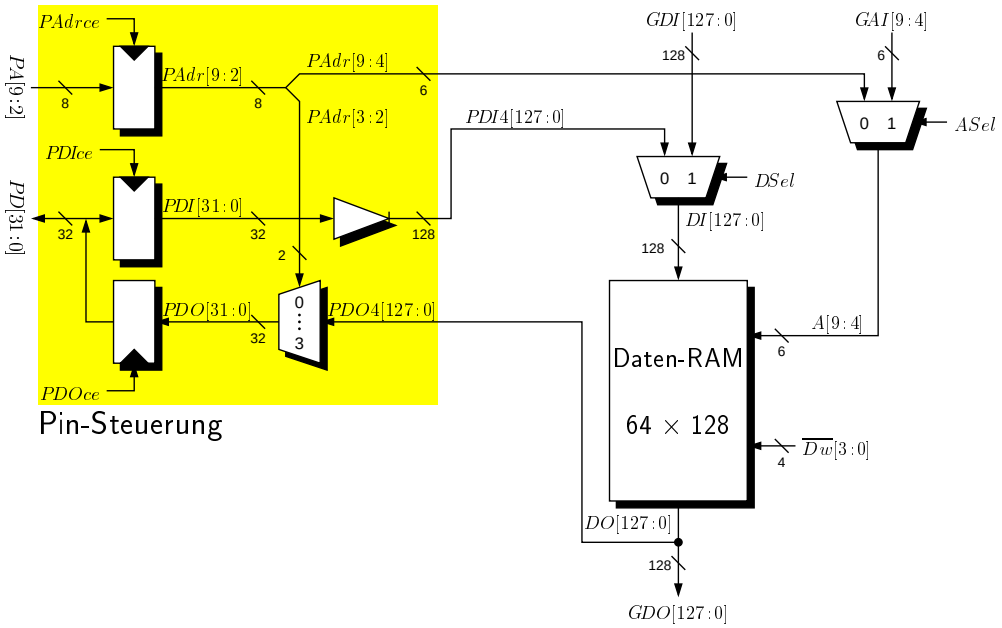

Abbildung 4.2: Datenpfade des RAMs

Abbildung 4.2 zeigt die Datenpfade der RAM-Chips. Die Bits [11 : 10] der Adressen sind in den Datenpfaden nicht eingezeichnet. Sie werden nur von der Kontrolle benutzt, um zu entscheiden, ob sich der Zugriff auf den jeweiligen

RAM-Chip bezieht (siehe Abschnitt 4.7.4). Die Datenpfade der Pin-Steuerung sind identisch zum Cache (siehe Abschnitt 4.2.1).

Die RAM-Logik wird durch *ASel*, *DSel* und die Schreibsignale $\overline{Dw}[3:0]$ gesteuert. Sind *ASel* und *DSel* aktiv, kann über die Glasfaser-Verbindung auf das Daten-RAM zugegriffen werden, andernfalls werden die über die Pins gesteuerten Busse *PAdr* und *PDI* verwendet. Die Schreibsignale $Dw[3:0]$ werden wie in Abschnitt 4.2.2 gesetzt.

4.4 Zusammenfassung

Abgesehen von dem Mux zur Auswahl der Adressbusse *PAdr* und *GAI* sind die Datenpfade der RAM-Chips eine "Teilmenge" der Datenpfade des Caches. Es ist also ohne großen Aufwand möglich, die Datenpfade für Cache und RAM zusammenzufassen (siehe Abbildung 4.3).

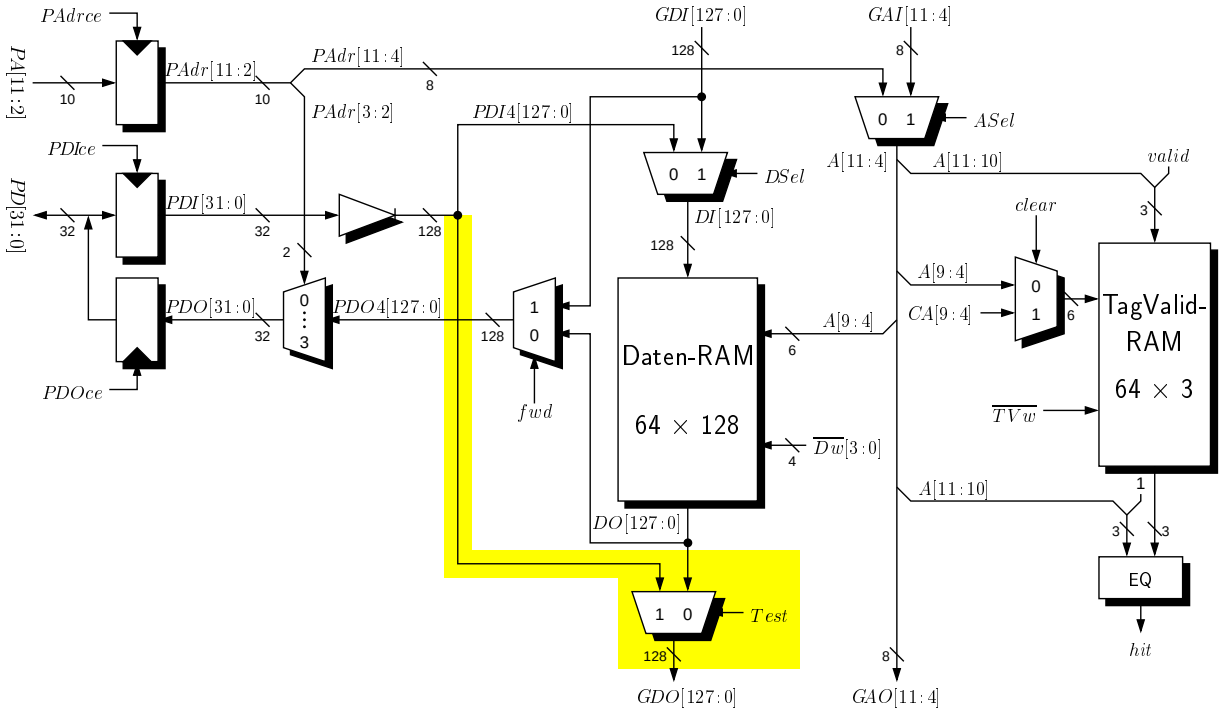

Abbildung 4.3: Datenpfade der Cache/RAM-Logik

Neben der Cache- bzw. RAM-Funktion unterstützen die zusammengefassten Datenpfade eine einfache Funktion zum Testen der Glasfaser-Verbindung (siehe Abschnitt 4.8). Dazu wurde ein zusätzlicher Mux (grau unterlegt) eingefügt. Ist das Konfigurations-Signal $Test$ aktiv, werden die Daten direkt von den Pins an den Sequenzer gesendet. Somit ist es möglich, die Glasfaser-Verbindung ohne Zugriffe auf das Daten-RAM zu testen. Ist das Signal fwd aktiv, können die vom Parallelisierer empfangenen Daten direkt über die Pins ausgegeben werden.

4.5 Kommunikation über die Pins

Der Cache soll mit einem PowerPC 603e von IBM [24] zusammenarbeiten. Deshalb wurde für Zugriffe über die Pins das Busprotokoll des PowerPCs übernommen. Das Busprotokoll konnte jedoch an einigen Stellen vereinfacht werden [30]. Da nur der Prozessor Anfragen auf den Bus stellt, konnten die Kontrollsignale für Mehrprozessorsysteme unberücksichtigt bleiben. Als weitere Vereinfachung wurde die Unterstützung von Zugriffen auf mehrere aufeinanderfolgende Speicherzellen innerhalb einer Anfrage (Burst-Zugriffe) abgeschaltet.

Ein Zugriff unterteilt sich in Adress- und Datenphase. Die beiden Phasen können sich überschneiden, die Datenphase beginnt allerdings frühstens einen Takt nach dem Beginn der Adressphase. Der Prozessor beginnt eine Phase durch Aktivieren der Valid-Signale $\overline{AV}$ beziehungsweise $\overline{DV}$. Der Cache beendet die Phasen indem er das entsprechende Acknowledge-Signal $\overline{AACK}$ oder $\overline{DACK}$ für einen Takt aktiviert.
Solange $\overline{AV}$ aktiv ist, garantiert der Prozessor, dass das Modussignal $\overline{write}$ und die Adressen PA gültig sind. Ist $\overline{write}$ aktiv, so greift der Prozessor schreibend auf den Cache zu, im anderen Fall lesend.
Bei einem Schreibzugriff signalisiert der Prozessor durch Aktivieren von $\overline{DV}$, dass gültige Daten auf dem Datenbus PD liegen. Greift der Prozessor lesend zu, so gibt er durch Aktivieren von $\overline{DV}$ an, dass er die Daten empfangen kann. Aktiviert der Cache $\overline{DACK}$, so muss er bei einem Lesezugriff gleichzeitig die entsprechenden Daten auf den Datenbus legen.

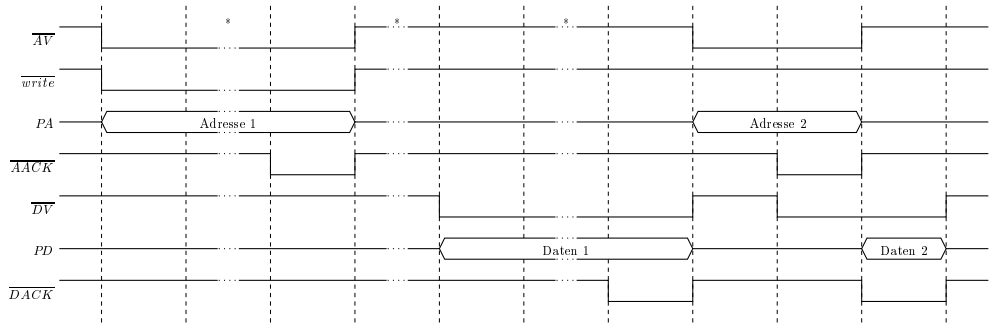

Abbildung 4.4: Langer Schreibzugriff gefolgt von kurzem Lesezugriff

Abbildung 4.4 zeigt zwei mögliche Speicherzugriffe mit dem vereinfachten Busprotokoll. Die mit $*$ gekennzeichneten Bereiche können aus mehreren Takten bestehen oder auch ganz wegfallen.

4.6 Kommunikation mit Sequenzer und Parallelisierer

Um Daten über die Glasfasern zu senden, muss die Cache/RAM-Logik den Sequenzer veranlassen, eine Übertragung zu starten. Hat der Parallelisierer neue Daten empfangen, muss er dies der Cache/RAM-Logik mitteilen. In diesem Abschnitt werden die dazu verwendeten Kontrollsignale und deren Bedeutung beschrieben.

Der Sequenzer erhält als Eingaben von der Kontrolle der Cache/RAM-Logik das Startsignal *DOReady* (Data Out Ready), das Resetsignal $\overline{SRST}$ und das zu übertragende Modusbit *ErwOut*. Von den Datenpfaden der Cache/RAM-Logik kommen die zu übertragenden Daten *GDO* und Adressen *GAO*.
Die Kontrolle der Cache/RAM-Logik veranlasst den Sequenzer, eine Übertragung über die Glasfaser zu starten, indem sie das Signal *DOReady* aktiviert. Ist *DOReady* aktiv, so müssen die zu übertragenden Daten und Adressen stabil bleiben (siehe Abschnitt 3.4). Eine Übertragung dauert so lange, bis *DOReady* wieder deaktiviert wird. Gegebenenfalls werden die Daten mehrfach gesendet. Der Parallelisierer verarbeitet jedoch nur die erste Übertragung der Daten (siehe Abschnitt 5.2). Bevor der Sequenzer eine neue Übertragung beginnen kann, muss er wieder in den Grundzustand gebracht werden; dazu wird das Resetsignal $\overline{SRST}$ aktiviert (siehe Abschnitt 5.3.5).
Die Art der Übertragung wird durch das Modus-Bit *ErwOut* (Error/Write Out) angezeigt. Die Bedeutung von *ErwOut* hängt vom Sender ab. Bei der Übertragung vom Cache zu den RAM-Chips gibt *ErwOut* an, ob es sich um einen Schreib- ($ErwOut = 1$) oder Lesezugriff ($ErwOut = 0$) handelt. Auf dem Rückweg von den RAM-Chips zum Cache wird durch $ErwOut = 1$ angezeigt, dass es auf dem Hinweg zu einem Übertragungsfehler gekommen ist. Um Übertragungsfehler erkennen zu können, werden die zu übertragenden Daten mit einem Hammingcode versehen (siehe Abschnitt 5.1).

Der Parallelisierer sendet die beiden Kontrollsignale $NewData$ und *Error* sowie die über die Glasfaser-Verbindung empfangenen Signale *GDI*, *GAI* und *ErwIn* an die Cache/RAM-Logik. Die empfangenen Signale entsprechen den gesendeten Signalen *GDO*, *GAO* und *ErwOut*.
Um zu signalisieren, dass neue Daten empfangen wurden, aktiviert der Parallelisierer das Signal $NewData$. Ist $NewData$ aktiv, garantiert der Parallelisierer, dass die übertragenen Daten stabil sind. Kam es bei der Übertragung zu einem Fehler, so wird das Signal *Error* aktiviert. Wird $NewData$ aktiviert, so muss *Error* noch nicht berechnet sein, spätestens aber einen Takt später.
Damit der Parallelisierer neue Daten empfangen kann, muss auch er von der Cache/RAM-Logik ein Reset erhalten. Ist $NewData$ aktiv, so wird das Resetsignal $\overline{PRST}$ aktiviert. Dieses Signal setzt die Kontrolle des Parallelisierers in den Grundzustand zurück. Die Ausgangs-Register des Parallelisierers werden von $\overline{PRST}$ nicht beeinflußt. Nur das Signal $NewData$ wird wieder auf 0 gesetzt.

4.7 Kontrolle

4.7.1 Aufbau und Initialisierung

Die Kontrolle der Cache- und der RAM-Logik wird durch einen gemeinsamen Automaten realisiert. Wird das globale Resetsignal $\overline{Reset}$ aktiviert, so geht die gemeinsame Kontrolle in den Zustand Clear. In diesem Zustand sind die Resetsignale für Sequenzer und Parallelisierer $\overline{SRST}$ und $\overline{PRST}$ aktiv, so dass sich der gesamte Chip im Grundzustand befindet.

Wird $\overline{Reset}$ wieder deaktiviert, geht die Kontrolle je nach Wert des Konfigurations-Signals *CMod* in den Startzustand der Kontrolle für den Cache C_Start oder das RAM R_Start. Ist der Chip als Cache konfiguriert ($CMod = 1$), so werden vorher noch die Valid-Bits des TagValid-RAM auf 0 gesetzt (siehe Abschnitt 4.2.3). Dazu zählt ein Zähler die Clear-Adresse $CA[9:0]$ hoch. Erst wenn der Zähler den Wert 63 erreicht, geht die Kontrolle in den Zustand C_Start über. In diesen 64 Takten reagiert die Kontrolle nicht auf Anfragen über die Pins.

Die Kontrolle kann durch die drei Signale *ETest*, *opt* und *SDD* konfiguriert werden (siehe Anhang A). Im Folgenden wird nur der Fall $ETest = 1$, $opt = 0$ und $SDD = 0$ beschrieben.

Zur Implementierung der Kontrolle wird ein Mealy-Automat mit binär kodiertem Zustand benutzt [36]. Die Ausgangssignale hängen sowohl vom aktuellen Zustand als auch von den Eingangssignalen ab. In der Regel werden die Ausgangssignale für den nächsten Takt vorberechnet und in einem Register gespeichert. Die Vorberechnung ist allerdings bei einigen Signale wie zum Beispiel *PAce* (siehe nächster Abschnitt) nicht sinnvoll. Diese Signale werden direkt berechnet.

4.7.2 Verarbeitung der Adressen

Um schnell auf eine Anfrage des Prozessors reagieren zu können, sollten die an den Pins anliegenden Adressen mit der nächsten steigenden Flanke der Clock nach dem Aktivieren von $\overline{AV}$ in den Adressregistern der Pinsteuerung gespeichert werden. Das Clock-Enable-Signal für das Adressregister *PAce* muss deshalb direkt aus $\overline{AV}$ berechnet werden. Es gilt $PAce = \neg\overline{AV}$.

Warnung: Hier ist ein Fehler in der Kontrolle! Neue Adressen dürfen nur in den Startzuständen der Kontrolle für Cache und RAM übernommen werden. Wird ein neuer Zugriff gestartet bevor der Chip $\overline{DACK}$ aktiviert, wird eine neue Adresse ins Adressregister geschrieben. Der noch nicht beendete Zugriff wird dann mit falscher Adresse fortgesetzt.
Um dies zu verhindern, darf die Adressphase eines neuen Zugriffes erst gestartet werden, wenn die Datenphase des letzten beendet wurde. Dies kann erreicht werden, indem auf der Platine das Signal $\overline{AACK}$ an den Pin $\overline{DACK}$ angeschlossen wird. Der Pin $\overline{AACK}$ wird ignoriert. Ein neuer Zugriff kann dann erst gestartet werden, wenn der vorherige abgeschlossen ist.

Neben der eigentlichen Adresse $PAdr[11:2]$ der Speicherzellen werden bei Zugriffen über die Pins noch die Adressbits $PAdr[13:12]$ benötigt, um die Art des Zugriffs zu bestimmen. Das Signal $PAdr[12]$ gibt an, ob auf den Hauptspeicher direkt oder über den Cache zugegriffen wird. Das Bit $PAdr[13]$ gibt an, ob der Zugriff den Prototypen betrifft. Auf diese Weise ist es möglich, auf der Platine neben dem Prototypen weiteren Speicher zu integrieren, der zum Beispiel den Programm-Code enthält [30]. Die Bedeutung der Werte von $PAdr[13:12]$ wird in Tabelle 4.1 aufgeschlüsselt. Bei einem Zugriff überprüft die Kontrolle

zunächst die Adressbits $PAdr[13:12]$, um zu entscheiden, ob sie auf den Zugriff reagieren muss.

$PAdr[13:12]$	Bedeutung
$0*$	Kein Zugriff auf den Prototyp
10	Direkter Zugriff auf den Hauptspeicher
11	Zugriff auf den Hauptspeicher über den Cache

Tabelle 4.1: Bedeutung der Adressbits $PAdr[13:12]$

Bei Zugriffen über die Glasfaser-Verbindung ist die Art des Zugriffes eindeutig. Die Bits $PAdr[13:12]$ müssen also nicht übermittelt werden.

Die Kontrolle benötigt also gültige Adressen, um zu entscheiden, ob sie auf einen Zugriff reagieren soll. Der interne Adressbus $PAdr$ hat allerdings erst einen Takt nach Aktivierung von $\overline{AV}$ gültige Werte. Die Kontrolle darf entsprechend erst einen Takt nach Aktivierung von $\overline{AV}$ auf die Adressen zugreifen. Die Kontrolle verwendet daher das wie folgt definierte Signale $\overline{AV2}$:

$$\overline{AV2}^{t+1} := \overline{AV}^t \vee \neg\overline{AACK}^t \ \forall t \geq 0.$$

Dieses Signal wird einen Takt nach $\overline{AV}$ aktiviert und gleichzeitig mit dem Ende der Adressphase ($\overline{AACK}$) wieder deaktiviert.

4.7.3 Kontrolle des Caches

Die Kontrolle des Caches verarbeitet die Busprotokoll-Signale $\overline{AV2}$, $\overline{write}$ und $\overline{DV}$ (siehe Abschnitt 4.5) sowie die obersten Adressbits $PAdr[13:12]$. Von der Hit-Berechnung erhält die Kontrolle das Signal hit (siehe Abschnitt 4.2.3) und vom Parallelisierer die Signale $NewData$, $Error$ und $ErwIn$ (siehe Abschnitt 4.6).

Abbildung 4.5 zeigt das Zustands-Diagramm der Kontrolle des Caches. Alle Anfragen an den Cache beginnen und enden im Zustand C_Start. Die Kontrolle muss Lese- und Schreibzugriffe über die Pins bearbeiten. Diese unterteilen sich jeweils in Cache-Hits ($hit = 1$) und Cache-Misses ($hit = 0$). Der Cache hat genau dann eine neuen Zugriff zu bearbeiten, wenn $\overline{AV2}$ aktiv ist und die Adressbits $PAdr[13:12]$ einen Zugriff auf den Cache signalisieren (siehe Tabelle 4.1). Dies wird durch die Aktivierung des Signal Acc angezeigt:

$$Acc = 1 :\Leftrightarrow (\overline{AV2} = 0) \wedge (PAdr[13:12] = 11).$$

Ist Acc aktiv, so aktiviert die Kontrolle noch im gleichen Takt das Acknowledge-Signal für die Adressphase $\overline{AACK}$.

Lesezugriffe

Ein Lesezugriff wird durch $\overline{write} = 1$ signalisiert. Lesezugriffe unterteilen sich in drei Abschnitte. Zunächst wird überprüft, ob sich die gesuchte Speicherzelle im Cache befindet. Ist sie nicht im Cache, so muss der zugehörige Block aus

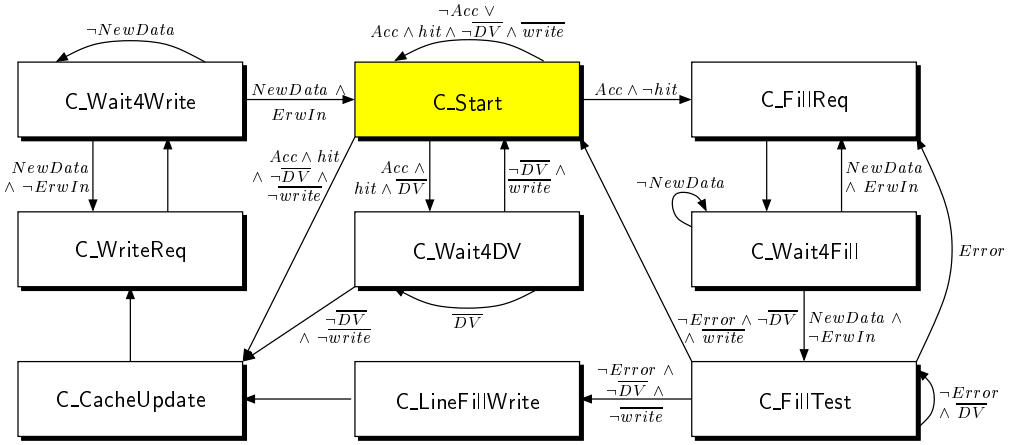

Abbildung 4.5: Zustands-Diagramm der Cache-Kontrolle

dem Hauptspeicher nachgeladen werden. Schließlich werden die Daten über die Pins ausgegeben.
Ist *Acc* aktiv, so sind die internen Adressen $PAdr[13:2]$ seit einem Takt gültig. Das Signal *hit* wurde demnach schon vom Schaltkreis Hit-Berechnung berechnet. Der erste Abschnitt des Zugriffs ist also bereits abgeschlossen.

Ist $hit = 1$, kann der zweite Abschnitt übersprungen werden. Die gesuchten Daten befinden sich also bereits im Daten-RAM. Da am Daten-RAM ebenfalls seit einem Takt gültige Adressen anliegen, liegen die gesuchten Daten bereits am Bus $PDO[31:0]$ an.
Damit der Cache den Zugriff abschließen kann, muss der Prozessor durch Aktivieren des Signals $\overline{DV}$ anzeigen, dass er bereit ist, die Daten zu empfangen. Ist $\overline{DV}$ bereits aktiv, so kann der Cache die Daten im nächsten Takt ausgeben. Die Kontrolle bleibt im Zustand C_Start und wartet auf den nächsten Zugriff. Ist $\overline{DV}$ inaktiv, so geht die Kontrolle in den Zustand C_Wait4DV bis der Prozessor $\overline{DV}$ aktiviert.
Das Ausgeben der Daten geschieht in zwei Schritten. Sobald $\overline{DV}$ aktiv ist, wird noch im gleichen Takt das Signal *PDOce* aktiviert. Die geforderten Daten stehen also zu Beginn des nächsten Taktes im Daten-Ausgangsregister. Im nächsten Takt aktiviert die Kontrolle das Output-Enable-Signals der Datenpins $\overline{Doe}$, um die Daten auf den Bus *PD* zu legen und beendet die Datenphase des Zugriffs durch Aktivieren des Signals $\overline{DACK}$.

Ist $hit = 0$, muss zunächst der Block mit der vom Prozessor angeforderten Speicherzelle aus den RAM-Chips nachgeladen werden. Da der Cache eine Write-Back Strategie verwendet, muss keine Cacheline verdrängt werden. Der Cache geht in den Zustand C_FillReq und startet einen Lesezugriff über die Glasfaser. Dazu wird das Signal *DOReady* aktiviert und das Signal *ErwOut* auf 0 gesetzt, um einen Lesezugriff zu signalisieren. Der Sequenzer sendet dann innerhalb eines Taktes (siehe Kapitel 5) die Adresse $PAdr[11:4]$ und das Signal *ErwOut* an die RAM-Chips.
Der Cache geht im nächsten Takt in den Zustand C_Wait4Fill und aktiviert das

Resetsignal für den Sequenzer $\overline{SRST}$. In diesem Zustand bleibt die Kontrolle, bis der Parallelisierer das Signal $NewData$ aktiviert und damit anzeigt, dass einer der RAM-Chips auf die Anfrage geantwortet hat. Sobald $NewData = 1$ gilt, aktiviert die Kontrolle des Caches im nächsten Takt das Signal $\overline{PRST}$, um den Parallelisierer in den Grundzustand zu bringen. Die empfangenen Daten *GDI* ändern sich dadurch nicht (siehe Abschnitt 5.4).
Aktiviert der Parallelisierer zusätzlich zu $NewData$ das Signal $ErwIn$, so wird signalisiert, dass es bei der Übertragung der Adressen an die RAM-Chips zu einem Übertragungsfehler gekommen ist. Die Kontrolle des Caches geht dann in den Zustand C_FillReq zurück, um die Übertragung zu wiederholen.
Falls $ErwIn$ inaktiv ist, so muss noch überprüft werden, ob es bei der Übertragung der Daten vom RAM-Chip an den Cache zu Fehlern gekommen ist, also $Error = 1$ gilt. Die Berechnung von $Error$ erfolgt im Zustand C_FillTest. Ist $Error = 1$, so muss die Übertragung ebenfalls wiederholt werden. Die Kontrolle geht also in den Zustand C_FillReq zurück.
Wurde die Übertragung erfolgreich beendet, ist der Cache bereit, die Daten auf den Bus PD zu legen. Dazu muss allerdings zunächst der Prozessor das Signal $\overline{DV}$ aktivieren. Die Kontrolle des Caches bleibt deshalb so lange im Zustand C_FillTest, bis der Prozessor $\overline{DV}$ aktiviert.
Ist $\overline{DV}$ aktiv, so werden noch im gleichen Takt die Signale fwd und $PDOce$ aktiviert, um die vom Prozessor geforderten Daten ins Daten-Ausgangsregister zu schreiben. Wie im Fall $hit = 1$ geht der Kontrolle im nächsten Takt in den Zustand C_Start zurück und aktiviert die Signale $\overline{Doe}$ und $\overline{DACK}$. Zusätzlich müssen allerdings noch das Daten-RAM und das TagValid-RAM aktualisiert werden. Um den vom Parallelisierer empfangenen Block am Daten-RAM anzulegen, wird das Signal $DSel$ aktiviert. Zusätzlich werden die Schreibsignale $\overline{Dw}[3:0]$ und $\overline{TVw}$ aktiviert und das Signal $valid$ auf 1 gesetzt.

Schreibzugriffe

Ist das Signal $\overline{write} = 0$, so ist der aktuelle Zugriff ein Schreibzugriff. Ein Schreibzugriff unterteilt sich in vier Abschnitte. Wie bei den Lesezugriffen muss zunächst überprüft werden, ob sich eine Speicherzelle bereits im Cache befindet und gegebenenfalls der entsprechende Block aus den RAM-Chips nachgeladen werden. Danach werden die Daten vom Prozessor in den sich nun im Cache befindenden Block geschrieben. Da eine Write-Through Strategie verwendet wird (siehe Abschnitt 3.1), muss der veränderte Block noch in den Hauptspeicher zurückgeschrieben werden. Der erste Abschnitt ist wie beim Lesezugriff bereits abgeschlossen, wenn Acc aktiviert wird.

Ist $hit = 0$, so wird wie beim Lesezugriff der gesuchte Block aus dem Hauptspeicher geladen. Die Kontrolle des Caches befindet sich dann im Zustand C_FillTest. Bevor die zu schreibenden Daten in das Daten-Eingangsregister der Pin-Steuerung geschrieben werden können, muss die Kontrolle in diesem Zustand warten, bis der Prozessor das Signal $\overline{DV}$ aktiviert.
Ist $\overline{DV}$ schließlich aktiv, so wird im gleichen Takt das Signal $PDIce$ aktiviert. Die vom Prozessor gesendeten Daten stehen also zu Beginn des nächsten Taktes im Daten-Eingangsregister. Der Cache geht in den Zustand C_LineFillWrite. In

diesem Zustand schreibt der Cache den aus dem Hauptspeicher nachgeladenen Block in das Daten-RAM und aktualisiert das TagValid-RAM.
Im darauffolgenden Takt geht die Kontrolle in den Zustand C_CacheUpdate und schreibt die vom Prozessor empfangene Speicherzelle in den Block. Dazu wird das entsprechende Schreibsignal $\overline{Dw}[\langle PAdr[3:2]\rangle]$ aktiviert (siehe Abschnitt 4.2.2). Damit ist der dritte Abschnitt des Schreibzugriffs abgeschlossen.
Zum Abschließen des Schreibzugriffs muss der Cache den veränderten Block in den Hauptspeicher zurückschreiben. Dazu geht der Cache in den Zustand C_WriteReq und startet einen Schreibzugriff über die Glasfaser-Verbindung, indem er die Signale $DOReady$ und $ErwOut$ aktiviert. Analog zum Lesezugriff geht die Kontrolle in den Zustand C_Wait4Write und wartet, bis der Parallelisierer das Signal $NewData$ aktiviert. Aktiviert der Parallelisierer zusätzlich das Signal $ErwIn$, so wird die Übertagung wiederholt. Andernfalls kann der Schreibzugriff abgeschlossen werden. Dazu geht die Kontrolle in den Zustand C_Start zurück und aktiviert das Signal $\overline{DACK}$.

Im Fall $hit = 1$ kann der zweite Abschnitt des Zugriffs wie beim Lesezugriff übersprungen werden. Der Cache wartet gegebenenfalls im Zustand C_Wait4DV bis der Prozessor $\overline{DV}$ aktiviert. Ist dies der Fall wird im gleichen Takt das Signal $PDIce$ aktiviert und der Cache kann direkt in den Zustand C_CacheUpdate übergehen. Der Rest des Zugriffs entspricht dem Fall $hit = 0$.

4.7.4 Kontrolle der RAM-Chips

Die RAM-Chips müssen zwei verschiedene Arten von Zugriffen bearbeiten: Zugriffe vom Cache über die Glasfaser-Verbindung und direkte Zugriffe über die Pins.
Der Hauptspeicher besteht aus vier RAM-Chips. Deshalb muss bei einem Zugriff auf den Hauptspeicher erkannt werden auf welchen der vier RAM-Chips sich der Zugriff bezieht. Dazu werden die RAM-Chips durchnummeriert. Die Nummer eines RAM-Chips wird durch die Konfigurations-Signale $Range[1:0]$ angegeben (siehe Anhang A). Die Nummer des RAM-Chips, auf den sich ein Zugriff bezieht, wird durch die Adressbits 11 und 10 angegeben. Deshalb benötigt die Kontrolle der RAM-Chips zum Bearbeiten der Pin-Zugriffe zusätzlich zu den Adressbits $PAdr[13:12]$ die Bits $PAdr[11:10]$ und zum Bearbeiten der Zugriffe über die Glasfasern die Bits $GAI[11:10]$. Weiterhin verarbeitet die Kontrolle die Signale des Busprotokolls $\overline{AV2}$, $\overline{write}$ und $\overline{DV}$ sowie die Signale $NewData$, $Error$ und $ErwIn$ vom Parallelisierer.
Abbildung 4.6 zeigt das Zustands-Diagramm der RAM-Kontrolle. Analog zur Cache-Kontrolle beginnen und enden alle Zugriffe im Zustand R_Start. Ein Zugriff über die Pins auf den RAM-Chip findet genau dann statt, wenn $\overline{AV2}$ aktiv ist, die Adressbits $PAdr[13:12]$ einen direkten Zugriff auf die RAM-Chips signalisieren und die Bits $PAdr[11:10]$ der Nummer des RAM-Chips entsprechen. Diese Bedingung wird zu dem Signal $PAcc$ zusammengefasst:

$$PAcc = 1 :\Leftrightarrow (\overline{AV2} = 0) \wedge (PAdr[13:10] = (1, 0, Range[1:0])).$$

Über Glasfaser-Verbindung wird auf den RAM-Chip zugegriffen, wenn der Parallelisierer das Signal $NewData$ aktiviert und die vom Cache empfangenen

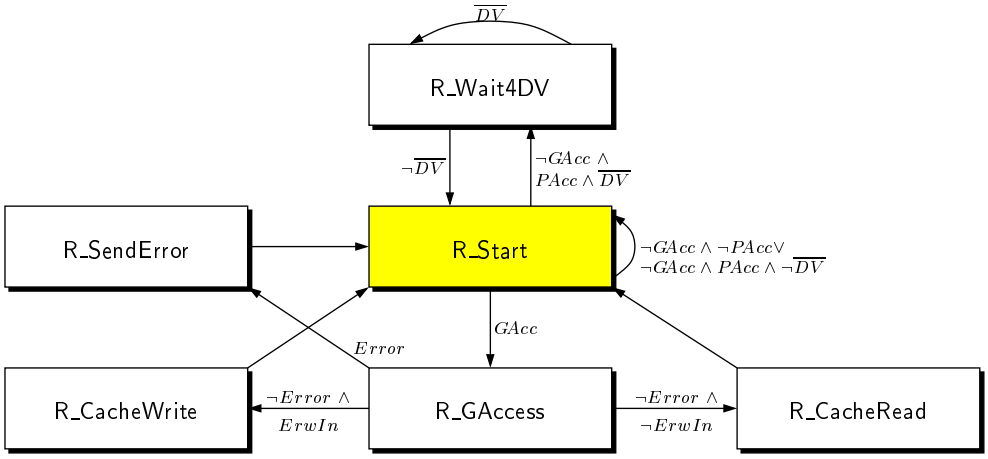

Abbildung 4.6: Zustands-Diagramm der RAM-Kontrolle

Adressbits $GAI[11:10]$ der Nummer des RAM-Chips entsprechen. Das Signal $GAcc$ wird wie folgt definiert:

$$GAcc = 1 :\Leftrightarrow (NewData = 1) \wedge (GAI[11:10] = Range[1:0]).$$

Solange die eingestellten Nummern $Range[1:0]$ für jeden RAM-Chip unterschiedlich sind, antwortet also höchstens ein RAM-Chip auf jeden Zugriff des Caches. Da der Cache immer auf eine Antwort eines RAM-Chips wartet, bevor er den nächsten Zugriff startet, kann es nicht dazu kommen, dass sich die Übertragungen von zwei RAM-Chips überschneiden. Sind $PAcc$ und $GAcc$ gleichzeitig aktiv, so wird zunächst der Glasfaser-Zugriff bearbeitet.

Sei $GAcc = 1$. Die RAM-Kontrolle geht dann in den Zustand R_GAccess über. In diesem Zustand wird überprüft, ob die Daten fehlerfrei übermittelt wurden. Ist dies nicht der Fall ($Error = 1$), wird im Zustand R_SendError das Signal $ErwOut = 1$ an den Cache übertragen. Danach geht die Kontrolle wieder in den Zustand R_Start. Der Cache würde in diesem Fall den Zugriff wiederholen, die RAM-Kontrolle interpretiert die Wiederholung jedoch als neuen Zugriff.
War die Übertragung fehlerfrei, wird bei einem Schreibzugriff ($ErwIn = 1$) im Zustand R_CacheWrite das RAM aktualisiert und eine Bestätigung zum Cache zurückgeschickt. Bei einem Lesezugriff werden im Zustand R_CacheRead die geforderten Daten über die Glasfaser-Verbindung gesendet.

Ist $PAcc = 1$, muss der Prozessor wie in der Cache-Kontrolle das Signal $\overline{DV}$ aktivieren, damit der RAM-Chip den Zugriff abschließen kann. Ist $\overline{DV}$ noch nicht aktiv, geht die RAM-Kontrolle in den Zustand R_Wait4DV. Wird $\overline{DV}$ schließlich vom Prozessor aktiviert, so wird entsprechen der Zugriffsart das Daten-RAM beschrieben oder die geforderten Daten über die Pins ausgegeben.

4.8 Test-Modus

Zusätzlich zum Standardbetrieb unterstützt die Cache/RAM-Logik einen Testmodus zum Testen der Glasfaser-Verbindung. Dieser wird durch das Konfigura-

tions-Signal $Test$ aktiviert. Ist $Test$ aktiv und ist der Chip als Cache konfiguriert ($CMod = 1$), werden bei einem Schreibzugriff auf den Cache die Daten direkt über die Glasfaser-Verbindung gesendet (siehe Abbildung 4.3). Es wird nicht auf eine Antwort der RAM-Chips gewartet.
Ist der Chip als RAM-Chip konfiguriert, so ist das Signal fwd dauerhaft aktiv, das heißt die vom Parallelisierer gelieferten Daten *GDI* liegen am Datenausgangs-Register an. Wird über die Pins lesend auf den RAM-Chip zugegriffen, so gibt der RAM-Chip das durch die Adresse selektierte Word von *GDI* aus. Dabei wird nicht überprüft, ob bereits eine Glasfaser-Übertragung stattgefunden hat.

Auf diese Weise kann die Glasfaser-Verbindung getestet werden, während nur ein kleiner Teil der Datenpfade und der Kontrolle der Cache/RAM-Logik benutzt wird. Insbesondere muss nicht auf die RAM-Bausteine zugegriffen werden.

Kapitel 5

Die Glasfaser-Übertragung

Dieses Kapitel widmet sich der Glasfaser-Übertragung und den dafür notwendigen Schaltkreisen. Zunächst wird in Abschnitt 5.1 die Wahrscheinlichkeit für das Auftreten eines Fehlers bei der Übertragung untersucht und die implementierte Fehlererkennung beschrieben. Das Protokoll für die Datenübertragung über die Glasfasern wird in Abschnitt 5.2 erklärt. In den Abschnitten 5.3 und 5.4 wird auf die Designs für Sequenzer und Parallelisierer eingegangen. Zusätzlich zur Beschreibung der Designs wird in diesen Abschnitten auch die Korrektheit der Schaltkreise bewiesen.

5.1 Fehlererkennung

Die Wahrscheinlichkeit p, dass eine von den Lasern über die Glasfaser-Verbindung gesendetes Eins vom Receiver als Null erkannt wird und umgekehrt liegt bei $p = 10^{-15}$ [39]. Wie die folgende Rechnung zeigt, ist dieser Wert zu hoch, um davon ausgehen zu können, dass der Betrieb mit hoher Wahrscheinlichkeit fehlerfrei verläuft.
Unter der Annahme, dass die Fehler unabhängig auftreten, ergibt sich mit $p = 10^{-15}$ ein Erwartungswert für das Auftreten des ersten Fehlers nach

$$\sum_{i=1}^{\infty} i \cdot (1-p)^{i-1} \cdot p = \frac{1}{p} \; = \; 10^{15}$$

übertragenen Bits [15, S. 116]. Bei jeder Übertragung über die Glasfaserkabel werden 137 Bits (128 Bit Daten, 8 Bit Adressen und das Modus-Bit $ErwOut$) gesendet. Angenommen, es kommt in jedem dritten Takt des langsamen Clocksignals $TCLK$ (10 MHz) zu einer Glasfaser-Übertragung, so ergibt sich eine erwartete meantime between failures (MTBF) von

$$\frac{10^{15} \cdot 3}{137 \cdot 10^{7} Hz} \approx 25,3 \text{ Tagen.}$$

Es ist also nötig, Fehler in der Übertragung zu erkennen. Dazu werden die Daten vor der Übertragung mit einem Hammingcode [22] kodiert. Der Hammingcode besteht neben den 137 Bit der ursprünglichen Nachricht aus 8 zusätzlichen

Parity-Bits $GPO[7:0]$. Er ermöglicht es, bis zu zwei Fehler in der Übertragung zu entdecken. Damit ein Fehler nicht erkannt wird, müssen von den $n := 137 + 8 = 145$ übertragenen Bits also mindestens drei Bits falsch empfangen werden. Für die Wahrscheinlichkeit P dieses Ereignisses gilt:

$$P \leq 1 - \sum_{i=0}^{2} \binom{n}{i} \cdot p^i \cdot (1-p)^{n-i} < 5 \cdot 10^{-40}.$$

Damit errechnet sich mit den obigen Angaben eine MTBF von

$$\frac{10^{40} \cdot 3}{5 \cdot 10^7 Hz} \approx 2 \cdot 10^{25} \text{ Jahren.}$$

Wird in einer Übertragung ein Fehler erkannt, so wird die Übertragung wiederholt (siehe Abschnitt 4.7.3).

Leider sind nach Beginn der Fertigung der Dies zwei Fehler im Design entdeckt worden, die die MTBF senken. Der erste Fehler befindet sich in dem Schaltkreis zur Berechnung des Hammingcodes (siehe Abbildung C.10). Durch ein fehlendes EXOR-Gatter wird die MTBF auf circa 10^{16} Jahre gesenkt. Da dies nicht weiter tragisch ist,[1] wird auf diesen Fehler nicht näher eingegangen.
Der zweite Fehler ist kritischer. Dieser Fehler kann bewirken, dass der Parallelisierer den Beginn einer Glasfaser-Übertragung nicht erkennt. Der Erwartungswert für das Auftreten dieses Fehlers beträgt circa 4,75 Jahre (siehe Seite 53). Die Häufigkeit, mit der dieser Fehler auftritt, wäre für einen kommerziellen Chip zu hoch, für den Test des Prototypen ist sie jedoch gering genug.

5.2 Übertragungs-Protokoll

Die Laser zur Steuerung der Glasfaser-Übertragung werden über Kondensatoren angesprochen. Wird eine lange Folge von Einsen übertragen, entläd sich der Kondensator und der Laser überträgt Nullen anstatt der gewünschten Einsen. Aus diesem Grund dürfen nicht mehr als vier Einsen in Folge gesendet werden. Dazu wird nach jedem vierten übertragenen Bit eine Null eingefügt. Geht man davon aus, dass Fehler unabhängig sind, wird durch das Einfügen von Nullen die Fehlerkennung nicht beeinflußt.

Da wegen der Veroderung der Signale am Receiver des Caches die RAM-Chips nicht dauerhaft senden dürfen (siehe Abschnitt 2.3), muss signalisiert werden, wann eine Übertragung beginnt. Vor der Übertragung der eigentlichen Daten wird daher eine Eins gesendet, anhand welcher der Parallelisierer auf der Empfängerseite den Beginn der Übertragung erkennt. Insgesamt ergibt sich für die Datenübertragung das in Abbildung 5.1 abgebildete Timing. Dieses Protokoll wird auch auf dem Weg vom Cache zu den RAM-Chips verwendet.
Inklusive der Parity-Bits des Hammingcodes müssen 145 Bits übertragen werden. Für die Übertragung ist eine Unterscheidung der zu übertragenen Bits in Daten, Adressen, Parity-Bits oder Kontrollsignale nicht notwendig. Sie können

[1] Das Alter des Universums wird auf circa 10^{10} Jahre geschätzt.

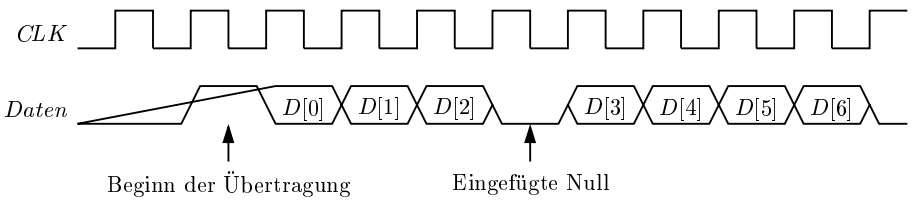

Abbildung 5.1: Protokoll der Glasfaser-Übertragung

also beliebig auf die 10 zur Verfügung stehenden Glasfaserkabel verteilt werden. Über jedes Glasfaserkabel werden 14 oder 15 Bits übertragen. Auf Glasfasern, die nur 14 Bit übertragen, wird der Einfachheit halber das 14. Bit doppelt übertragen. Zur Verteilung der Bits auf die Glasfaserkabeln siehe Anhang B.2.

Um 15 Bit über ein Glasfaserkabel zu senden, werden mit dem verwendeten Protokoll $\lceil (15+1) \cdot (1+\frac{1}{4}) \rceil = 20$ Takte der schnellen Clock benötigt. Bei dem angestrebten Verhältnis zwischen schneller und langsamer Clock von 32:1 würde die Übertragung aller 145 Bit also weniger als einen Takt der langsamen Clock dauern. Würden beispielsweise bei einer Leseanfrage des Caches an den Speicher nur die Adressen und das Modusbit übertragen, so würde man dadurch keine Zeit gewinnen. Um die Kontrolle und die Berechnung der Parity-Bits zu vereinfachen, werden aus diesem Grund bei jeder Übertragung alle 145 Bit gesendet. Die bei einem Lesezugriff vom Cache zusätzlich gesendeten Daten werden von der Cache/RAM-Logik des empfangenden Chips ignoriert.
Da immer gleich viele Daten über die Glasfaser-Verbindung gesendet werden, kann der Parallelisierer das Ende einer Übertragung durch Zählen der bereits gesendeten Daten berechnen. Der Sequenzer muss das Ende einer Übertragung also nicht signalisieren und muss insbesondere nicht nach dem Senden der Daten sofort die Übertragung beenden.

5.3 Sequenzer

5.3.1 Aufbau

Der Sequenzer berechnet zunächst aus den von der Cache/RAM-Logik übertragenen Daten $GDO[127:0]$, Adressen $GAO[11:4]$ und dem Modusbit $ErwOut$ die Parity-Bits $GPO[7:0]$ des Hammingcodes. Aus allen 145 Bit werden danach entsprechend dem Übertragungsprotokoll die Signale $GOut[9:0]$ berechnet, die über die Glasfaserkabeln gesendet werden. Das Ausgangssignal $GOut[10]$ des Sequenzers ist das zu den übertragenen Daten gehörende Clocksignal.
Solange der Sequenzer Daten übertragen soll, aktiviert die Kontrolle der Cache/RAM–Logik das Startsignal *DOReady* (Data Out Ready). Durch dieses Signal wird gleichzeitig garantiert, dass GDO, GAO und $ErwOut$ stabil sind.
Bei der Initialisierung des Chips und nach jeder Übertragung wird das Resetsignal für den Sequenzer $\overline{SRST}$ aktiviert, um ihn in den Grundzustand zu versetzen (siehe Abschnitt 4.7).
Abbildung 5.2 zeigt einen Überblick über die Datenpfade des Sequenzers. Je nach Wert des Signals *CMod* wird als Clocksignal CLK das über die Pins erhal-

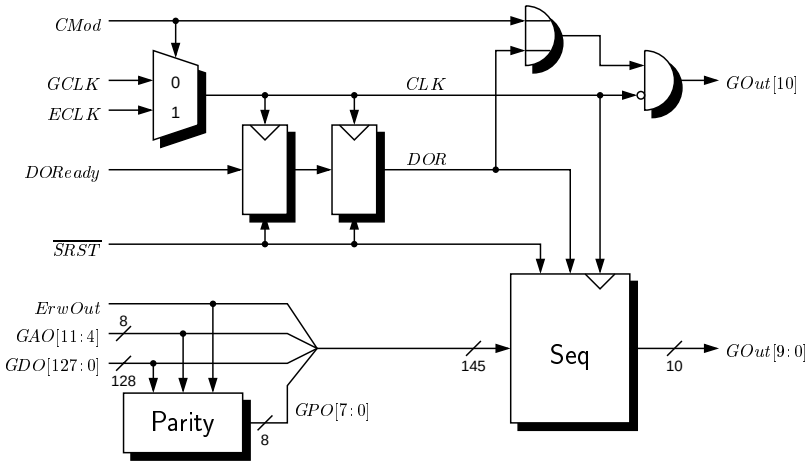

Abbildung 5.2: Überblick über den Sequenzer

tene Clocksignal *ECLK* oder das vom Parallelisierer gesendete Signal *GCLK* verwendet (siehe Abschnitt 3.3). Mit diesem Clocksignal wird zunächst das Startsignal *DOReady* synchronisiert (siehe Abschnitt 3.4). Die Schaltkreise des Sequenzers benutzen nur das synchronisierte Signale *DOR*.
Ist der Chip als Cache konfiguriert ($CMod = 1$), so wird dauerhaft ein Clocksignal über die Glasfaser-Verbindung an die RAM-Chips gesendet. Es gilt:

$$GOut[10] = \neg CLK = \neg ECLK.$$

Die RAM-Chips dürfen nur dann ein Clocksignal übertragen, wenn sie auch Daten übertragen, also wenn gilt $DOR = 1$. Ist $CMod = 0$ gilt folglich:

$$GOut[10] = \begin{cases} \neg CLK = \neg GCLK & \text{falls } DOR = 1, \\ 0 & \text{falls } DOR = 0. \end{cases}$$

Das Signal *DOR* ändert sich nach der positiven Flanke von *CLK*. Zum Zeitpunkt der Aktivierung von *DOR*, gilt also $CLK = 1$. Würde man für die Glasfaser-Clock $GOut[10]$ nicht die invertierte Clock benutzen, so wäre die erste steigende Flanke von $GOut[10]$ durch das UND-Gatter verzögert und damit der erste Takt kürzer. Diese Verzögerung tritt nicht auf, wenn man die invertierte Clock über die Glasfaser überträgt. Diese ist kein Problem, da sowieso auf den verschiedenen Chips von einer asynchronen Clock ausgegangen wird.

Die Parity-Bits $GPO[7:0]$ werden im Schaltkreis Parity berechnet. Der Schaltkreis besteht aus EXOR-Bäumen, die sich direkt aus der Definition des Hammingcodes [22] ableiten. Die Berechnung der Parity-Bits kann vor der Übertragung geschehen und wird deshalb in platzsparender CMOS-Technologie realisiert. In dieser Arbeit wird nicht näher auf diesen Schaltkreis eingegangen.

Wird von der Kontrolle der Cache/RAM-Logik das Signal *DOReady* aktiviert, sind die Adressen *GAO* bereits seit einem Takt stabil. Da Schreibzugriffe den Inhalt der RAM-Bausteine immer zur negativen Flanke der Clock verändern [4], sind auch die Daten *GDO* seit mindestens einem halben Takt stabil. Die Synchronisierung dieser Signale ist also unproblematisch.

Das einzige kritische Eingangssignal ist *ErwOut*. Dieses Signal wird möglicherweise gleichzeitig mit *DOReady* aktiviert. Eine genaue Timinganalyse ergibt jedoch, dass dies kein Problem darstellt. Das liegt daran, dass *ErwOut* erst zwei Takte der schnellen Clock nach Aktivieren von *DOReady* vom Sequenzer verarbeitet wird (siehe Abschnitt 5.3.5 und Anhang B.2).
Dies gilt analog auch für die Parity-Signale. Diese hängen entweder nicht von *ErwOut* ab oder werden mit der Verzögerung von nur einem EXOR-Gatter aus *ErwOut* berechnet (siehe Abbildung C.10). Der Sequenzer verwendet die Parity-Bits frühestens zwölf Takte nach Aktivierung von *DOReady*. Im Folgenden kann also der Einfachheit halber davon ausgegangen werden, dass alle Signale bereits stabil sind, wenn *DOR* aktiviert wird.

Aus den Bussen *GDO*, *GAO* und *GPO* sowie dem Signal *ErwOut* werden im Schaltkreis Seq die Signale $GOut[9:0]$ berechnet. Da die Schaltkreise zur Berechnung jedes der Signale $GOut[9:0]$ identisch sind, wird in den folgenden Abschnitten nur die Berechnung des Signals $GOut[0]$ aus den Signalen $GDO[14:0]$ betrachtet.

Der Schaltkreis Seq stellt zusammen mit den beiden Parallelisierer-Stufen (siehe Abschnitt 5.4.2) den Kern der Glasfaser-Übertragung dar. Da diese Schaltkreise nicht auf vorhandener Literatur aufbauen (im Gegensatz zum Beispiel zur Cache/RAM-Logik)[2], wird zusätzlich zur Beschreibung der Designs die Korrektheit der Schaltkreise bewiesen. Auf das Design des Schaltkreises Seq wird in den folgenden Abschnitten eingegangen.

5.3.2 Alternativen zur Sequenzialisierung

Für das Design des eigentlichen Sequenzialisierungs-Schaltkreises Seq bieten sich zwei grundsätzliche Ansätze an: ein parallel beschreibbares Shift-Register S und ein Mux-Baum M (siehe Abbildung 5.3).

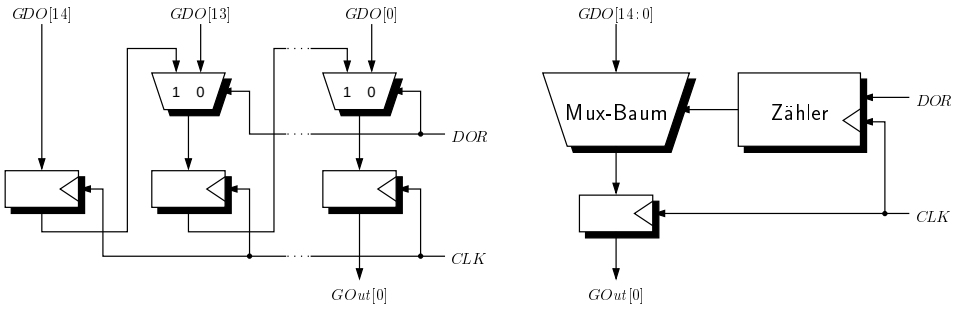

Abbildung 5.3: Sequenzialisierung mit Shiftregister und Mux-Baum

Für das Delay $D(S)$ und die Kosten $C(S)$ des Shiftregisters gilt beim Sequen-

[2] Über die Cache-Schaltkreise wird in [37] auch kein Korrektheitsbeweis geführt. Momentan wird jedoch von Sven Beyer im Rahmen seiner Promotion am Lehrstuhl von Professor Paul die Korrektheit des Schaltkreises mit dem Beweissystem PVS überprüft.

zialisieren von 15 Bit:

$$\begin{aligned} D(S) &= D_{Reg} + D_{Mux}, \\ C(S) &= 15 \cdot C_{Reg} + 14 \cdot C_{Mux}. \end{aligned}$$

Beim Mux-Baum können die Kosten des Zählers vernachlässigt werden, da für die 10 Mux-Bäume nur ein Zähler benötigt wird. Geht man davon aus, dass die Ausgänge des Zählers direkt durch Register gesteuert werden, und läßt man den Zähler in der üblichen Weise hochzählen (0,1,2,. . . ,13,14,0,. . .), ergibt sich für das Delay $D(M)$ und die Kosten $C(M)$ des Mux-Baums:

$$\begin{aligned} D(M) &= \max\{D_{Reg} + 4 \cdot D_{Mux}, D(\mathsf{Zähler})\}, \\ C(M) &= C_{Reg} + 14 \cdot C_{Mux}. \end{aligned}$$

Der Ausgang des Sequenzialisierungs-Schaltkreises wird mit hoher Frequenz betrieben. Es muss also besondere Rücksicht auf das Delay genommen werden. Zusätzlich sind auch die Kosten sehr wichtig, da der Schaltkreis in der teuren ECL-Technologie realisiert werden muss.
Betrachtet man nur das Delay, bietet sich das Shiftregister an. Aus diesem Grund wurde dieser Ansatz zunächst verfolgt. Es hat sich jedoch herausgestellt, dass eine Realisierung auf Grund des hohen Platz- und des Stromverbrauchs nicht möglich ist. Daher wurde ein Mux-Baum realisiert.

Der Mux-Baum hat nur circa die halben Kosten, jedoch ein wesentlich höheres Delay. Die Berechnung des Delays basiert auf der Annahme, das der Zähler in der üblichen Weise hochzählt. Steuert man den Mux-Baum aber auf eine intelligentere Weise, so kann das Delay der oberen beiden Multiplexer-Stufen versteckt werden.

5.3.3 Aufbau des Muxbaums

Für die Sequenzialisierung wird ein Mux-Baum mit 16 Eingängen verwendet. Der zusätzliche Eingang zu den 15 Datenbits $GDO[14:0]$ wird für das Übertragungsprotokoll benötigt (siehe Abschnitt 5.3.4). Die Tiefe des Mux-Baums wird dadurch nicht beeinflußt. In welcher Reihenfolge die zu übertragenden Daten $GDO[14:0]$ auf die Eingänge $M[15:0]$ des Mux-Baums verteilt werden, hängt von den Kontrolle des Sequenzers ab und wird später festgelegt.
Um das Delay der oberen beiden Multiplexer-Stufen verstecken zu können, wird der Mux-Baum in fünf 4:1 Multiplexer aufgeteilt (siehe Abbildung 5.4). Jeder der fünf 4:1 Multiplexer hat ein eigenes Paar Select-Signale. Die oberen Multiplexer werden durch die Signale $HC_i[1:0]$ mit $0 \leq i \leq 3$ gesteuert, der untere durch die Signale $LC[1:0]$. Die Ausgänge der oberen Multiplexer werden entsprechend mit $M[i]$, $0 \leq i \leq 3$, bezeichnet.
Im Folgenden wird mit i immer der Index der oberen vier Multiplexer beziehungsweise der zugehörigen Signale bezeichnet. Die Einschränkung $0 \leq i \leq 3$ wird deshalb meist weggelassen.
Der untere Multiplexer selektiert nacheinander die vier Signale $M[3:0]$. Auf diese Weise wird jedes der Signale $M[i]$ nur jeden vierten Takt ausgewählt. Bevor ein Signal erneut ausgewählt wird, kann im oberen Multiplexer bereits

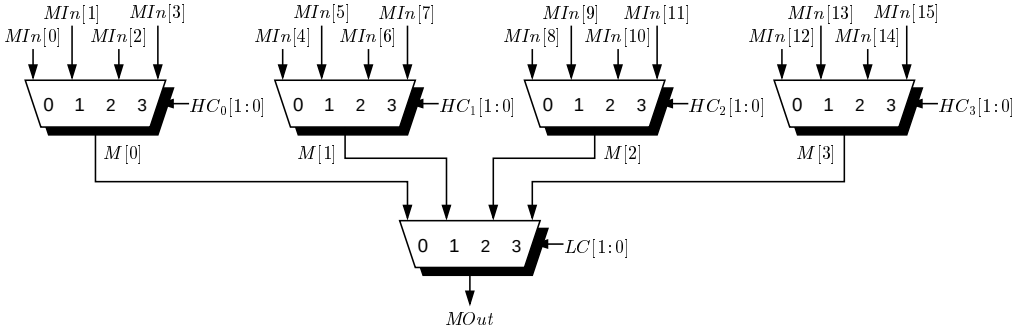

Abbildung 5.4: Aufbau des Mux-Baums

das entsprechende Signal für die nächste Iteration vorberechnet werden. Wird das Signal vier Takte später erneut ausgewählt, liegt der neue Wert schon am Ausgang des oberen Multiplexers an und kann ohne Verzögerung im unteren Multiplexer weiterverarbeitet werden. Auf die Berechnung von LC und HC_i wird in Abschnitt 5.3.5 näher eingegangen.
Welcher Eingang des Mux-Baumes durchgelassen wird, ergibt sich aus einer Kombination von LC und HC_i. Um eine kompaktere Schreibweise zu ermöglichen wird der Vektor $MSel[3:0]$ definiert, der den ausgewählten Eingang zu einem Zeitpunkt $t \in \mathbb{Z}$ angibt.

Definition 5.1 Für $t \in \mathbb{Z}$ wird definiert:

$$MSel[3:0]^t := (LC[1:0]^t, HC_{\langle LC[1:0]\rangle}[1:0]^t).$$

Das folgende Lemma zeigt, dass der zum Zeitpunkt $t \in \mathbb{Z}$ selektierte Eingang des Mux-Baumes auch tatsächlich durch $MSel[3:0]^t$ bestimmt wird.

Lemma 5.2 Sei $t \in \mathbb{Z}$, dann gilt für den Ausgang $MOut$ des Mux-Baumes:

$$MOut^t = MIn[\langle MSel[3:0]^t\rangle].$$

Beweis: Sei $t \in \mathbb{Z}$ und sei $i := \langle LC[1:0]^t\rangle$. Dann gilt:

$$\begin{aligned} MOut^t = M[i]^t &= MIn[4 \cdot i + \langle HC_i[1:0]^t\rangle] \\ &= MIn[\langle LC[1:0]^t, HC_i[1:0]^t\rangle] \\ &= MIn[\langle MSel[3:0]^t\rangle]. \end{aligned}$$

■

5.3.4 Anpassung an das Übertragungsprotokoll

Der Sequenzialisierungs-Schaltkreis muss die Daten $GDO[14:0]$ entsprechend dem in Abschnitt 5.2 eingeführten Protokoll ausgeben. Um die Eins zu Beginn der Übertragung einzufügen, wird der Bus $GDO[14:0]$ vor der Sequenzialisierung zum Bus

$$GDO2[15:0] := (GDO[14:0], 1) \tag{5.1}$$

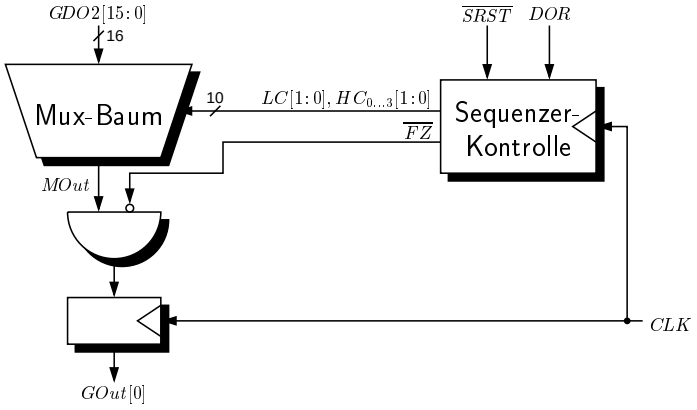

Abbildung 5.5: Aufbau des Sequenzer-Schaltkreises

erweitert. Der Bus $GDO2$ wird an den Eingang des Mux-Baumes angelegt.
Die Nullen zwischen je vier übertragenen Bits ebenfalls vor der Sequenzialisierung einzufügen, würde die Tiefe des Mux-Baumes erhöhen und vier weitere Multiplexer kosten. Daher ist es günstiger, die Nullen nach der Sequenzialisierung durch ein UND-Gatter einzufügen (siehe Abbildung 5.5). Dazu hält die Sequenzer-Kontrolle die Steuersignale für den Mux-Baum jeden fünften Takt an und aktiviert gleichzeitig das Signal $\overline{FZ}$.
Findet keine Übertragung statt ($DOR = 0$), so wird das Signal $\overline{FZ}$ ebenfalls aktiviert. Dadurch sendet der Chip nur Nullen über die Glasfaser-Verbindung.

5.3.5 Die Sequenzerkontrolle

Voraussetzungen

Die Sequenzer-Kontrolle berechnet die Signale $\overline{FZ}$, $LC[1:0]$ und $HC_i[1:0]$. Das Resetsignal $\overline{SRST}$ ist bei der Initialisierung des Chips und nach jeder Übertragung aktiv. Dadurch geht die Kontrolle in den Grundzustand. Im Korrektheitsbeweis wird deshalb ohne Beschränkung der Allgemeinheit davon ausgegangen, dass nur eine Übertragung stattfindet.

Voraussetzung 5.3 Sei $t \in \mathbb{Z}$. Für das Eingangssignal $\overline{SRST}$ der Sequenzer–Kontrolle gilt:[3]

$$\overline{SRST}^t = \begin{cases} 0 & \text{falls } t < -2, \\ 1 & \text{falls } t \geq -2. \end{cases}$$

Es werden so lange Daten über die Glasfaserkabel gesendet, wie das Startsignal DOR aktiv ist. Der Parallelisierer auf der Empfängerseite verarbeitet jedoch auf jedem Glasfaserkabel nur die ersten 20 übertragenen Bits. Alle weiteren übertragenen Daten werden ignoriert. Für das Signal DOR wird deshalb ohne Beschränkung der Allgemeinheit folgendes vorausgesetzt.

[3]Es werden hier negative Zeitpunkte benutzt, um den Beginn der Übertragung am Nullpunkt zu normieren. Die Zählung der Takte entspricht dem Clocksignal CLK (320 MHz).

Voraussetzung 5.4 Sei $t \in \mathbb{Z}$. Für das Eingangssignal DOR der Sequenzer–Kontrolle gilt:

$$DOR^t = \begin{cases} 0 & \text{falls } t < 0, \\ 1 & \text{falls } t \geq 0. \end{cases}$$

Korrektheit

Bevor das Design der Sequenzerkontrolle beschrieben wird, werden in diesem Abschnitt die Korrektheits-Kriterien formalisiert, die das Design erfüllen muss. Die Kriterien werden bei der Beschreibung des Designs bewiesen.
Der Ausgang $\overline{FZ}$ der Sequenzer-Kontrolle muss aktiv sein, wenn keine Übertragung stattfindet (also $DOR = 0$ gilt) und in jedem fünften Takt der Übertragung. Zusammen mit Voraussetzung 5.4 ergibt sich der folgende Satz.

Satz 5.5 Sei $t \in \mathbb{Z}$. Für den Ausgang $\overline{FZ}$ der Sequenzerkontrolle muss gelten:[4]

$$\overline{FZ}^t = \begin{cases} 0 & \text{falls } t < 0 \text{ oder } t \equiv 4 \pmod 5, \\ 1 & \text{sonst.} \end{cases}$$

Damit alle Daten übertragen werden, müssen nacheinander alle Eingänge des Mux-Baums selektiert werden. Die Reihenfolge der Selektion ist zunächst egal, da die Belegung der Eingänge des Mux-Baums entsprechend angepasst wird. Es genügt also, wenn in den ersten 16 Takten, in denen $\overline{FZ}$ nicht aktiv ist, paarweise verschiedene Eingänge selektiert werden. Zusammen mit Lemma 5.2 und Satz 5.5 muss also für die ersten 20 Takte der folgende Satz gelten.

Satz 5.6 Seinen $t_0, t_1 \in \{0, \ldots, 19\}$ mit $\overline{FZ}^{t_0} = 1$, $\overline{FZ}^{t_1} = 1$. Dann gilt für den Vektor $MSel[3:0]$:

$$t_0 = t_1 \iff MSel^{t_0} = MSel^{t_1}.$$

Um das Delay der oberen Hälfte des Mux-Baums verstecken zu können, müssen die Select-Signale des durch LC ausgewählten oberen 4:1-Mux seit mehreren Takten stabil sein. Am günstigsten wäre, wenn die Signale seit vier Takten stabil sind. Diese Voraussetzung muss nur erfüllt sein, wenn der Ausgang des Mux-Baumes auch wirklich durchgelassen wird, also $\overline{FZ}$ inaktiv ist.

Satz 5.7 Sei $t \in \mathbb{N}$ mit $\overline{FZ}^t = 1$ und sei $i := \langle LC^t[1:0] \rangle$. Dann gilt für $HC_i^t[1:0]$:

$$HC_i^t[1:0] = HC_i^{t-j}[1:0] \quad \text{für } 0 \leq j \leq 4.$$

Überblick

Die Sequenzer-Kontrolle unterteilt sich in die Schaltkreise LCon und HCon (siehe Abbildung 5.6). Der Schaltkreis LCon berechnet die Signale $\overline{FZ}$ und $LC[1:0]$. Die Signale $HC_i[1:0]$ werden vom Schaltkreis HCon berechnet.

[4] Im Folgenden wird mit $a \equiv b \pmod c$ die Relation Modulo bezeichnet, während $a = b \bmod c$ für die Operation Modulo steht.

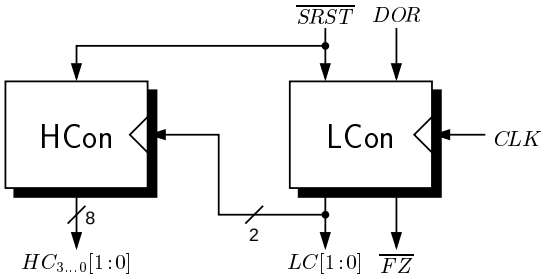

Abbildung 5.6: Die Sequenzer-Kontrolle

Der Schaltkreis HCon besteht aus vier Zählern. Jeder Zähler steuert einen der Busse $HC_i[1:0]$ steuern damit den zugehörigen oberen Mux. Ändert sich LC, so wird genau der Zähler hochgezählt, dessen zugehöriger oberer Mux im letzten Takt durch LC selektiert wurde. Dazu werden die Signale $LC[1:0]$ als Clocksignale von HCon benutzt.

Um die beiden Signale $LC[1:0]$ direkt als Clocksignale von HCon verwenden zu können, muss aus einer Flanke auf einem der beiden Signale auf den letzten Wert von LC zurückgeschlossen werden. Dies ist nicht möglich, wenn LCon in der üblichen Weise $(0, 1, 2, 3, 0, \ldots)$ hochzählt. Aus einer steigenden Flanke von $LC[0]$ läßt sich zum Beispiel nicht folgern, ob LC im letzten Takt den Wert 0 oder 2 hatte.
Aus diesem Grund wird zur Berechnung von LC ein modifizierter Zähler benutzt – im Folgenden GCC (Gray Code Counter) genannt –, der beim Hochzählen 2 und 3 vertauscht $(0, 1, 3, 2, 0, \ldots)$. Wie sich an Abbildung 5.7 nachvollziehen läßt, entspricht jeder steigenden oder fallenden Flanke der Ausgänge $Cnt[1:0]$ des Zählers ein Wert des Zählers im Takt davor. Zum Beispiel hat $Cnt[0]$ genau dann eine steigende Flanke, wenn der letzte Wert 0 war, und eine fallende, wenn der letzte Wert 3 war. Entsprechend kann der Zähler, der HC_0 steuert, mit $LC[0]$ und der Zähler, der HC_3 steuert, mit $\neg LC[0]$ geclockt werden.

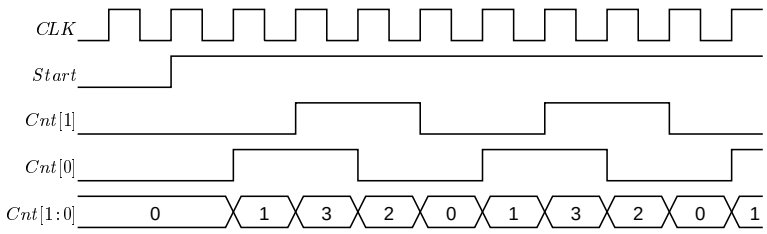

Abbildung 5.7: Timing des modifizierten Zählers GCC

Abbildung 5.8 zeigt den Zähler GCC. Neben dem Clocksignal CLK und dem asynchronen Resetsignal $\overline{RST}$ verwendet der Zähler noch das Eingangssignal $Start$. Erst wenn $Start$ aktiviert wird, fängt GCC an zu zählen.
Um das Verhalten des Schaltkreises GCC in den folgenden Korrektheitsbeweisen kompakt darstellen zu können, wird die von GCC berechnete Inkrementierungs-Funktion γ eingeführt.

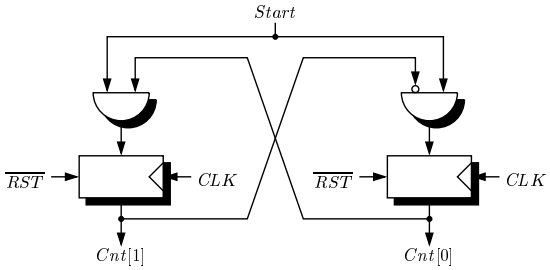

Abbildung 5.8: Der Schaltkreis GCC

Definition 5.8 $\gamma : \mathbb{B}^2 \to \mathbb{B}^2, (x, y) \mapsto \gamma(x, y) := (y, \neg x)$.

j	$\gamma^j(00)$	$\langle\gamma^j(00)\rangle$
0	00	0
1	01	1
2	11	3
3	10	2
4	00	0

Tabelle 5.1: Werte der Funktion γ

Tabelle 5.1 zeigt die Werte der Funktion γ. Dabei bezeichnet γ^j wie üblich das j-fache Hintereinander-Ausführen der Funktion. Über die Funktion γ gilt das folgende Lemma.

Lemma 5.9 Seien $j, k \in \mathbb{N}$. Dann gilt:

$$\gamma^j(00) = \gamma^k(00) \Longleftrightarrow j \equiv k \pmod 4.$$

Beweis: "$\Leftarrow$" Die Aussage folgt induktiv aus $\gamma^4(00) = 00$.

"$\Rightarrow$" Seien $j' := j \bmod 4$ und $k' := k \bmod 4$ mit $j' \neq k'$. Aus der Tabelle 5.1 und der Rück-Richtung ergibt sich:

$$\gamma^j(00) = \gamma^{j'}(00) \neq \gamma^{k'}(00) = \gamma^k(00).$$ ■

Lemma 5.10 Sei $t \in \mathbb{Z}$. Für den Ausgangsbus $Cnt[1{:}0]$ des Schaltkreises GCC gilt:

$$Cnt^t = \begin{cases} 00 & \text{falls } \overline{RST}^{t-1} = 0 \text{ oder } \overline{RST}^t = 0 \text{ oder } Start^{t-1} = 0, \\ \gamma(Cnt^{t-1}) & \text{sonst.} \end{cases}$$

Beweis: Die Aussage folgt direkt aus der Definition von γ. ■

LCon

Zum Einfügen der Nullen muss der Zähler jeden vierten Takt angehalten werden. Dazu wird der Zähler GCC im Schaltkreis LCon (siehe Abbildungen 5.9 und 5.10) um ein Register erweitert, in dem gespeichert wird, ob der Zähler vier Takte gezählt hat. Dies ist genau dann der Fall, wenn der Ausgang $LC[1:0]$ des Zählers im letzten Takt den Wert 2 hatte. Ist dies der Fall, wird das Signal $EQ2$ für einen Takt aktiviert und damit der Zähler über das Startsignal für einen weiteren Takt auf dem Wert 0 gehalten. Das Startsignal des Zählers wird gleichzeitig als Ausgang $\overline{FZ}$ des Schaltkreises LCon verwendet.

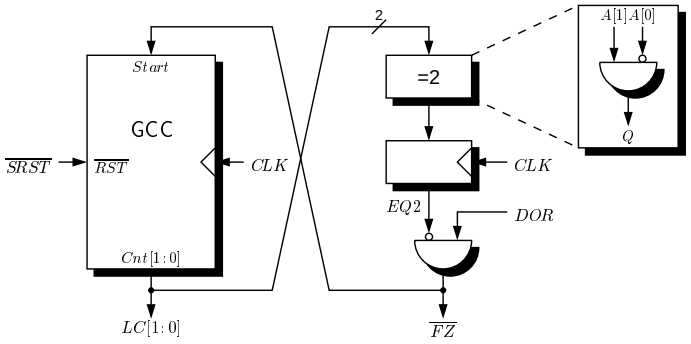

Abbildung 5.9: Der Schaltkreis LCon

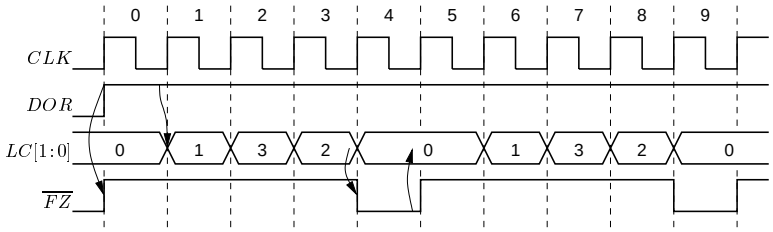

Abbildung 5.10: Timing des Schaltkreises LCon

Beweis (zu Satz 5.5): Sei $t \in \mathbb{Z}$. Zu zeigen ist:

$$\overline{FZ}^t = \begin{cases} 0 & \text{falls } t < 0 \text{ oder } t \equiv 4 \pmod 5, \\ 1 & \text{sonst.} \end{cases}$$

Für alle $t < 0$ gilt mit der Voraussetzung 5.4:

$$\begin{aligned} DOR^t = 0 &\implies \overline{FZ}^t = 0 \\ &\overset{5.10}{\implies} LC[1{:}0]^{t+1} = 00 \\ &\implies EQ2^{t+2} = 0. \end{aligned}$$

Es folgt $LC[1{:}0]^0 = 00$ und $EQ2^0 = 0$. Induktiv ergibt sich mit $\min\{t | \gamma^t(00) = 10\} = 3$ (siehe Tabelle 5.1) für $0 \leq t \leq 3$:

$$\begin{aligned} EQ2^{t-1} = 0 &\implies LC[1{:}0]^t = \gamma(LC[1{:}0]^{t-1}) = \gamma^t(00), \\ \langle LC[1{:}0]^{t-1} \rangle = \langle \gamma^{t-1}(00) \rangle \neq 2 &\implies EQ2^t = 0 \implies \overline{FZ}^t = 1. \end{aligned}$$

Weiter gilt:

$$\begin{aligned}
\langle LC[1:0]^3\rangle = 2 &\implies EQ2^4 = 1 \implies \overline{FZ}^4 = 0,\\
EQ2^3 = 0 &\implies LC[1:0]^4 = \gamma^4(00) = 00,\\
\langle LC[1:0]^4\rangle = 0 &\implies EQ2^5 = 0 \implies \overline{FZ}^5 = 1,\\
EQ2^4 = 1 &\implies LC[1:0]^5 = 00.
\end{aligned}$$

Der Schaltkreis hat also nach fünf Takte den gleichen Zustand. Induktiv ergibt sich somit die Behauptung des Satzes. ■

Aus dem Beweis ergibt sich das folgende Korollar.

Korollar 5.11 Sei $t \in \mathbb{Z}$ und sei $r := t \bmod 5$. Dann gilt für den Ausgang $LC[1:0]$ des Schaltkreises LCon:

$$LC[1:0]^t = \begin{cases} 00 & \text{falls } t < 0,\\ \gamma^r(00) & \text{falls } t \geq 0. \end{cases}$$

HCon

Im Schaltkreis HCon werden andere Clocksignale als CLK verwendet. Damit die Zählung der Takte einheitlich ist, beziehen sich jedoch weiterhin alle Taktangaben auf das Signal CLK.
Der Schaltkreis HCon besteht, wie bereits erwähnt, aus vier parallelen Zählern GCC (siehe Abbildungen 5.11 und 5.12). Das Startsignal der Zähler ist konstant 1. Die UND-Gatter in den Zählern (siehe Abbildung 5.8) können also wegoptimiert werden.[5] Die Zähler für HC_i verwenden zum Clocken die Signale $LC[0]$, $LC[1]$, $\neg LC[1]$ und $\neg LC[0]$ für $i = 0, \ldots, 3$. Um das Verhalten von HCon beschreiben zu können, muss zunächst das Verhalten von LC genauer untersucht werden.

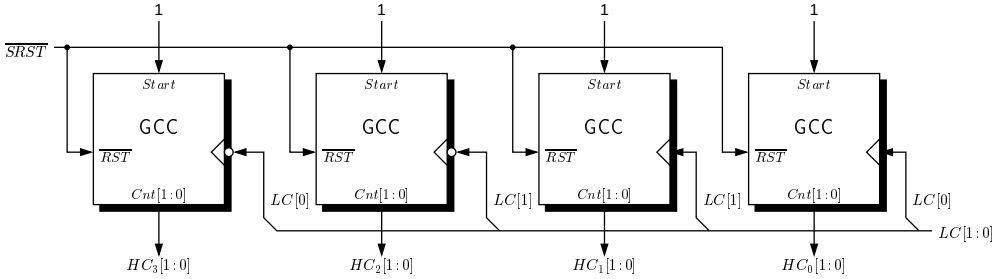

Abbildung 5.11: Der Schaltkreis HCon

[5] Dies klingt zunächst wenig, aber da der Sequenzer die Breite des gesamten Chips bestimmt, werden bei der Herstellung der Chips dadurch etwa € 500 gespart.

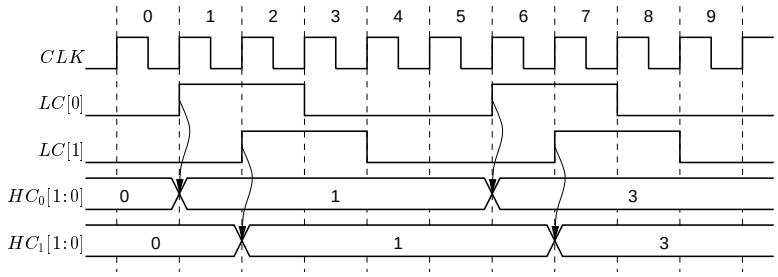

Abbildung 5.12: Timing des Schaltkreises HCon

Lemma 5.12 Sei $t \in \mathbb{Z}$. Für die Ausgänge $\overline{FZ}$ und $LC[1:0]$ von LCon gilt:

$$\begin{aligned}
LC[0]^t = 0 \ \wedge \ LC[0]^{t+1} = 1 &\iff \overline{FZ}^t = 1 \wedge \langle LC^t \rangle = 0,\\
LC[1]^t = 0 \ \wedge \ LC[1]^{t+1} = 1 &\iff \overline{FZ}^t = 1 \wedge \langle LC^t \rangle = 1,\\
\neg LC[1]^t = 0 \ \wedge \ = \neg LC[1]^{t+1} = 1 &\iff \overline{FZ}^t = 1 \wedge \langle LC^t \rangle = 2,\\
\neg LC[0]^t = 0 \ \wedge \ = \neg LC[0]^{t+1} = 1 &\iff \overline{FZ}^t = 1 \wedge \langle LC^t \rangle = 3.
\end{aligned}$$

Beweis: Ist $\overline{FZ}^t = 0$, so gilt mit Satz 5.5 $t < 0$ oder $t \equiv 4 \pmod 5$. Aus Korollar 5.11 folgt $LC[1:0]^t = 00$. Aus $\overline{FZ}^t = 0$ folgt außerdem $LC[1:0]^{t+1} = 00$. Ein steigende oder fallende Flanke auf einem der beiden Signale $LC[1:0]$ kann also nur vorkommen, wenn $\overline{FZ}$ im Takt vorher inaktiv war.

Ist $\overline{FZ}^t = 1$, so folgt aus dem Beweis von Satz 5.5 $LC[1:0]^{t+1} = \gamma(LC[1:0]^t)$. Die Behauptung folgt nun aus der Definition von γ (siehe Tabelle 5.1). ■

Aus $X^t = 0 \ \wedge \ X^{t+1} = 1$ folgt, dass X eine steigende Flanke im Takt $t+1$ hat. Aus dem Aufbau von HCon ergibt sich damit das folgende Korollar.

Korollar 5.13 Sei $t \in \mathbb{Z}$. Für die Ausgangsbusse $HC_i[1:0]$ des Schaltkreises HCon gilt:

$$HC_i^{t+1} = \begin{cases} \gamma(HC_i^t) & \text{falls } \overline{FZ}^t = 1 \text{ und } \langle LC^t \rangle = i,\\ HC_i^t & \text{sonst.} \end{cases}$$

Der nächste Wert der Select-Signale HC_i wird also genau dann berechnet, wenn der Ausgang des entsprechenden Mux im letzten Takt über die Glasfaser gesendet wurde. Damit läßt sich Satz 5.7 beweisen.

Beweis (zu Satz 5.7): Sei $t \in \mathbb{N}$ mit $\overline{FZ}^t = 1$ und sei $i := \langle LC^t[1:0] \rangle$. Zu zeigen ist für $HC_i^t[1:0]$:

$$HC_i^t = HC_i^{t-j} \quad \text{für } 0 \leq j \leq 4.$$

Sei $t' := t - j$ mit $0 < j \leq 4$. Ist $\overline{FZ}^{t'} = 0$, so folgt aus Korollar 5.13:

$$HC_i^{t'} = HC_i^{t'+1}. \tag{5.2}$$

Sei also $\overline{FZ}^{t'} = 1$. Weiterhin seien $r := t \bmod 5$ und $r' := t' \bmod 5$. Damit folgt:

$$\left.\begin{array}{rcl} 0 < t - t' \leq 4 & \Longrightarrow & r \neq r' \\ \overline{FZ}^{t} = 1 & \overset{5.5}{\Longrightarrow} & r' \neq 4 \\ \overline{FZ}^{t'} = 1 & \overset{5.5}{\Longrightarrow} & r \neq 4 \end{array}\right\} \Longrightarrow \quad r \not\equiv r' \pmod 4$$

$$\Longrightarrow \quad LC^{t'} \overset{5.11}{=} \gamma^{r'}(00) \overset{5.9}{\neq} \gamma^{r}(00) = LC^{t}$$

$$\overset{5.13}{\Longrightarrow} \quad HC_i^{t'} = HC_i^{t'+1}. \tag{5.3}$$

Wendet man die Gleichungen (5.2) und (5.3) nacheinander für $j = 1, \ldots, 4$ an, folgt die Behauptung des Satzes. ■

Es bleibt also nur noch Satz 5.6 zu beweisen. Dazu muss zunächst untersucht werden, welchen Wert die Busse HC_i haben. Es gilt das folgende Lemma.

Lemma 5.14 Seien $t \in \mathbb{N}$ mit $\overline{FZ}^{t} = 1$ und $i := \langle LC^t \rangle$. Sei $j := \lfloor \frac{t}{5} \rfloor$, dann gilt für den Ausgang HC_i des Schaltkreises HCon:

$$HC_i^t = \gamma^j(00).$$

Beweis: Vorbemerkung: für $t \leq -2$ gilt nach Voraussetzung 5.3 $\overline{SRST}^{t-1} = 0$, also mit Lemma 5.10 $HC_i^t = 00$. Nach Satz 5.5 gilt $\overline{FZ}^{-1} = 0$ und mit Korollar 5.13 folgt $HC_i^{-1} = HC_i^{-2} = 00$. Zusammengefaßt ergibt sich:

$$HC_i^t = 00 \text{ für } t < 0. \tag{5.4}$$

Die Ausgänge HC_i sind also vor dem Start der Übertragung alle auf 0. Das Lemma kann nun durch Induktion über j bewiesen werden.

Induktionsanfang ($j = 0$): Sei $t \in \mathbb{N}$ mit $\lfloor \frac{t}{5} \rfloor = j = 0$, das heißt $t < 5$. Sei $\overline{FZ}^{t} = 1$ und $i := \langle LC^t \rangle$. Aus Satz 5.5 folgt $t < 4$ und damit:

$$HC_i^t \overset{5.7}{=} HC_i^{t-4} \overset{(5.4)}{=} 00 = \gamma^0(00).$$

Induktionsschritt ($j - 1 \to j$): Es gelte die Aussage des Lemmas für alle $t' \in \mathbb{N}$ mit $\lfloor \frac{t'}{5} \rfloor < j$. Sei nun $t \in \mathbb{N}$ mit $\lfloor \frac{t}{5} \rfloor = j$, $\overline{FZ}^{t} = 1$ und $i := \langle LC^t \rangle$. Aus Satz 5.5 und Korollar 5.11 folgt:

$$\overline{FZ}^{t-5} = 1 \text{ und } \langle LC^{t-5} \rangle = i.$$

Mit Satz 5.7 und Korollar 5.13 folgt:

$$HC_i^t = HC_i^{t-4} = \gamma(HC_i^{t-5}) \overset{\text{IV}}{=} \gamma^{j+1}(00).$$ ■

Damit läßt sich der Wert von $MSel^t[3:0]$ für $t \in \mathbb{Z}$ berechnen.

Lemma 5.15 Seien $t \in \mathbb{Z}$, $r := t \bmod 5$ und $s := \lfloor \frac{t}{5} \rfloor$. Dann gilt:

$$MSel[3:0]^t = (\gamma^r(00), \gamma^s(00)).$$

Beweis: Seien $t \in \mathbb{Z}$, $r := t \bmod 5$ und $s := \lfloor \frac{t}{5} \rfloor$.

$$\begin{aligned} MSel[3:0]^t &= (LC^t, HC^t_{\langle LC^t \rangle}) \\ &\overset{5.14}{=} (LC^t, \gamma^s(00)) \\ &\overset{5.11}{=} (\gamma^r(00), \gamma^s(00)). \end{aligned}$$

■

Beweis (zu Satz 5.6): Seien $t_0, t_1 \in \{0, \ldots, 19\}$ mit $\overline{FZ}^{t_0} = 1$, $\overline{FZ}^{t_1} = 1$. Zu zeigen ist:

$$t_0 = t_1 \Longleftrightarrow MSel[3:0]^{t_0} = MSel[3:0]^{t_1}.$$

"$\Rightarrow$" Klar.

"$\Leftarrow$" Seien $r_0 := t_0 \bmod 5$, $r_1 := t_1 \bmod 5$, $s_0 := \lfloor \frac{t_0}{5} \rfloor$ und $s_1 := \lfloor \frac{t_1}{5} \rfloor$. Aus $MSel[3:0]^{t_0} = MSel[3:0]^{t_1}$ folgt mit Lemma 5.15:

$$(\gamma^{r_0}(00), \gamma^{s_0}(00)) = MSel[3:0]^{t_0} = MSel[3:0]^{t_1} = (\gamma^{r_1}(00), \gamma^{s_1}(00))$$

Mit Lemma 5.9 folgt $r_0 \equiv r_1 \pmod 4$ und $s_0 \equiv s_1 \pmod 4$. Weiterhin gilt:

$$\begin{aligned} t_0, t_1 \in \{0, \ldots, 19\} &\Longrightarrow s_0, s_1 < 4, \\ \overline{FZ}^{t_0} = \overline{FZ}^{t_1} = 0 &\overset{5.5}{\Longrightarrow} r_0, r_1 < 4. \end{aligned}$$

Also gilt $r_0 = r_1$ und $s_0 = s_1$ und somit

$$t_0 = 5 \cdot s_0 + r_0 = 5 \cdot s_1 + r_1 = t_1.$$

■

5.3.6 Zusammenfassung

Mit den Sätzen über die Sequenzerkontrolle kann nun das Verhalten des gesamten Schaltkreises Seq (siehe Abbildung 5.5) beschrieben werden.

Lemma 5.16 Seien $t < 20$, $r := (t-1) \bmod 5$ und $s := \lfloor \frac{t-1}{5} \rfloor$. Dann gilt für den Ausgang $GOut[0]$ des Sequenzers:

$$GOut[0]^t = \begin{cases} 0 & \text{falls } t \leq 0 \text{ oder } t \equiv 0 \pmod 5, \\ MIn[\langle \gamma^r(00), \gamma^s(00) \rangle] & \text{sonst.} \end{cases}$$

Beweis: Sei $t \in \mathbb{Z}$. Für den Ausgang $GOut[0]$ gilt:

$$GOut[0]^t = MOut^{t-1} \wedge \overline{FZ}^{t-1}.$$

Aus Satz 5.5 folgt:

$$GOut[0]^t = \begin{cases} 0 & \text{falls } t-1 < 0 \text{ oder } t-1 \equiv 4 \pmod 5, \\ MOut^{t-1} & \text{sonst.} \end{cases}$$

Die Behauptung folgt damit aus den Lemmata 5.15 und 5.2. ■

MIn	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$GDO2$	10	14	6	2	11	15	7	3	9	13	5	1	8	12	4	0

Tabelle 5.2: Mapping von $GDO2$ auf MIn

Damit die zu übertragenden Daten $GDO2[15:0]$ beginnend mit dem niederwertigsten Bit gesendet werden, muss also gelten:

$$GDO2[i] = MIn[\langle \gamma^r(00), \gamma^s(00) \rangle] \text{ mit } r := i \bmod 4 \text{ und } s := \left\lfloor \tfrac{i}{4} \right\rfloor . \tag{5.5}$$

Da nur in 4 von 5 Takten Daten übertragen werden, wird in dieser Rechnung Modulo 4 gerechnet beziehungsweise durch 4 geteilt. Damit ergibt sich das in Tabelle 5.2 gezeigte Mapping. Man rechnet leicht nach, daß dieses Mapping die Gleichung (5.5) erfüllt. Damit gilt der folgende Satz.

Satz 5.17 Sei $t \in \{1, \ldots, 20\}$ mit $t \equiv 4 \pmod 5$ und sei $S := 4 \cdot \lfloor \frac{t-4}{5} \rfloor$. Dann gilt für den Ausgang $GOut[0]$ des Sequenzers:

$$(GOut[0]^t, \ldots, GOut[0]^{t-3}) = GDO2[S+3:S].$$

Da $GOut[0]^1 = GDO2[0] \overset{(5.1)}{=} 1$ gilt, beginnt also jede Übertragung mit einer Eins.

5.4 Parallelisierer

5.4.1 Erkennen der Glasfaser-Übertragung

Bevor der Parallelisierer die über die Glasfaser-Verbindung empfangenen Daten parallelisieren kann, muss er den Beginn einer Übertragung erkennen. Der Beginn einer Übertragung wird signalisiert, indem auf den Signalen $GIn[9:0]$ eine Eins übertragen wird. Eine Überprüfung aller Signale wäre aber zu aufwendig. Deshalb wird im Design nur getestet, ob die Verundung der Signale $GIn[5]$ und $GIn[4]$ den Wert Eins hat.

Da vor der Übertragung auf allen Signalen Nullen gesendet werden, erkennt der Parallelisierer den Beginn einer Übertragung zu früh, wenn beide Signale gleichzeitig den falschen Wert haben. Der Erwartungswert für das erste Auftreten dieses Fehlers beträgt

$$\frac{10^{15} \cdot 10^{15}}{320 \cdot 10^6 \text{ MHz}} \approx 10^{14} \text{ Jahre.} \tag{5.6}$$

Bei einem tatsächlichen Beginn einer Übertragung kann aber eines der Signale mit einer Wahrscheinlichkeit von 10^{-15} den Wert Null haben. Der Beginn der Übertragung würde in diesem Fall nicht erkannt werden. Mit den Annahmen aus Abschnitt 5.1 ergibt sich für diesen Fehler eine MTBF von

$$\frac{10^{15} \cdot 3}{2 \cdot 10^{10} \text{ MHz}} \approx 4,75 \text{ Jahren.}$$

Dies wurde beim Design des Chips übersehen. Da eine MTBF von 4,75 Jahren für den Test des Prototypen ausreicht, ist dies jedoch nicht problematisch. In einer überarbeiteten Version des Prototypen sollte der Beginn der Übertragung durch einen "Mehrheitsentscheid" von drei Signalen erkannt werden (siehe Abbildung C.18). Der Parallelisierer würde dann mit der Parallelisierung beginnen, sobald mindestens zwei von drei über die Glasfaser empfangenen Signale den Wert Eins haben. Dies würde mit Gleichung (5.6) zu einer MTBF von circa 10^{14} Jahren führen.
In den folgenden Korrektheitsbeweisen wird davon ausgegangen, dass alle Daten richtig übertragen wurden.

5.4.2 Überblick

Der Parallelisierer berechnet aus den über die Glasfaser-Verbindung empfangenen Signale $GIn[9:0]$ wieder die parallelen Daten $GDI[127:0]$, Adressen $GAI[7:0]$ und das Modusbit $ErwIn$. Zusätzlich wird überprüft, ob diese Signale fehlerfrei übertragen wurden. Ist dies nicht der Fall, aktiviert der Parallelisierer das Signal $Error$. Sobald die Parallelisierung abgeschlossen ist und die Signale GDI, GAI und $ErwIn$ stabil sind, wird das Signal $NewData$ aktiviert. Durch das Resetsignal $\overline{PRST}$ wird der Parallelisierer nach einer Übertragung wieder in den Grundzustand gebracht und kann die nächste Übertragung bearbeiten.

Bevor der Parallelisierer die empfangenen Daten an die Cache/RAM-Logik übergeben kann, müssen die Signale mit speziellen Gattern von ECL nach CMOS umgewandelt werden. Diese Gatter verbrauchen in der verwendeten Technologie [3] von allen Gattern am meisten Strom und zusammen mit Registern auch am meisten Platz. Würde man erst nach der gesamten Parallelisierung von ECL nach CMOS umwandeln, so benötigt man für dem Parallelisierer mindestens 145 Register zum Speichern der Daten und 145 Umwandler. Diese würden Kosten und Stromverbrauch des Parallelisierers zu sehr in die Höhe treiben.[6]
Es ist also notwendig, den Parallelisierer in zwei Stufen aufzuteilen und schon nach einer teilweisen Parallelisierung die Signale nach CMOS zu wandeln. Die empfangenen Signale $GIn[9:0]$ werden von der ersten Stufe des Parallelisierers in je vier parallele Signale umgewandelt. Da die in den Datenstrom eingefügten Nullen bereits in der ersten Stufe "herausgefiltert" werden können, werden diese Signale mit einem Fünftel der Frequenz der ursprünglichen Signale betrieben. Bei einer Frequenz von 320 MHz bei der ECL-Clock hat das zugehörige Clock-Signal also eine Frequenz von 64 MHz. Dies genügt, da die Umwandler und die CMOS-Gatter bei Frequenzen bis zu 80 MHz arbeiten können.

Abbildung 5.13 zeigt einen Überblick über den Parallelisierer. Die erste Stufe des Parallelisierers (Par1) wird in ECL realisiert, die zweite Stufe (Par2) in CMOS. Der erste Stufe des Parallelisierers verteilt die über die einzelnen Signale $GIn[9:0]$ übertragenen Daten auf je 4 Signale des Busses $GF[39:0]$. Aus den Signalen $GF[39:0]$ werden in Par2 schließlich die empfangenen Daten $GDI[127:$

[6] Da die Umwandler von CMOS nach ECL wesentlich strom- und platzsparender sind, stellt sich dieses Problem beim Sequenzer nicht.

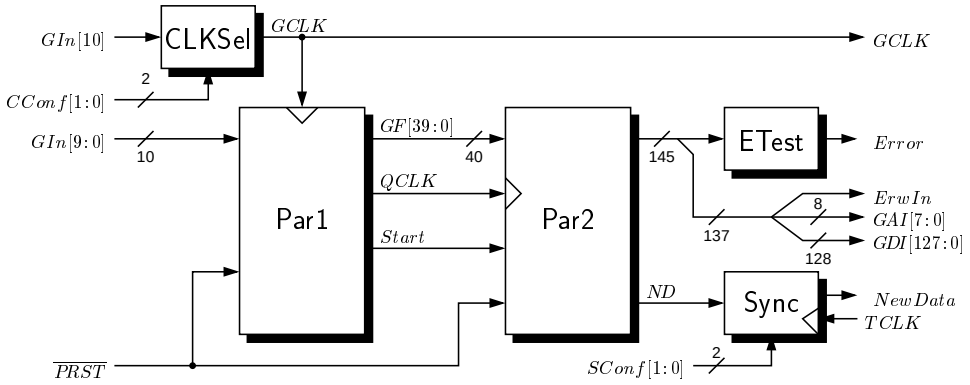

Abbildung 5.13: Überblick über den Parallelisierer

0], Adressen $GAI[7:0]$, das Modusbit $ErwIn$ sowie die Parity-Bits $GPI[7:0]$ berechnet.
Die zweite Stufe des Parallelisierers erhält als Clocksignal von der ersten Stufe das Signal $QCLK$. Dieses Signal hat ein Fünftel der Frequenz des Eingangs-Clocksignals $GIn[10]$ (also circa 64 MHz). Die zweite Stufe beginnt mit der Parallelisierung, wenn die erste Stufe das Signal $Start$ aktiviert. Sobald die Parallelisierung abgeschlossen ist, aktiviert die zweite Stufe das Signal ND.

Der Schaltkreis ETest testet, ob es bei der Übertragung zu Fehlern gekommen ist. Dazu wird aus den empfangenen Signalen $GDI[127:0]$, $GAI[7:0]$ und $ErwIn$ erneut die Parity-Bits berechnet (siehe Abschnitt 5.3.1). Diese Parity-Bits werden dann mit den übertragenen Parity-Bits $GPI[7:0]$ verglichen. Sind die beiden Vektoren gleich, so werden die übertragenen Daten als korrekt angesehen. Andernfalls wird das Signal $Error$ aktiviert.

Im Schaltkreis CLKSel (siehe Abbildung 5.14) kann das vom Parallelisierer verwendete Clocksignal $GCLK$ konfiguriert werden. Dies wurde sicherheitshalber implementiert, da aufgrund von Ungenauigkeiten in den Lasern beziehungsweise Receivern und im Herstellungsprozess der Dies das genaue Timing zwischen dem Clocksignal $GIn[10]$ und den Datensignalen $GIn[9:0]$ nicht berechnet werden kann.

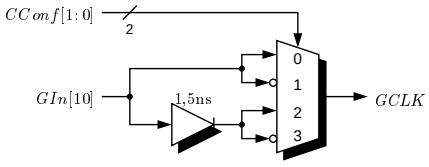

Abbildung 5.14: Der Schaltkreis CLKSel

Je nach Einstellung der Konfigurations-Signale $CConf[1:0]$ wird für $GCLK$ die übertragene Clock $GIn[10]$, ein um circa 1,5ns verzögertes Signal oder die jeweils invertierten Signale verwendet. Bei einer Taktfrequenz von 320 MHz ist es so möglich, die positiven Flanken des Clocksignals $GCLK$ in Schritten von einem Vierteltakt zu verschieben.

Bevor das Signal *ND* an die Kontrolle der Cache/RAM-Logik übergeben wird, muss es mit der langsamen Clock $TCLK$ synchronisiert werden (siehe Abschnitt 3.4). Im Schaltkreis Sync kann zwischen verschiedenen Varianten der Synchronisation gewählt werden. Im Betrieb wird man experimentell feststellen müssen, welche Synchronisations-Variante die Beste ist.

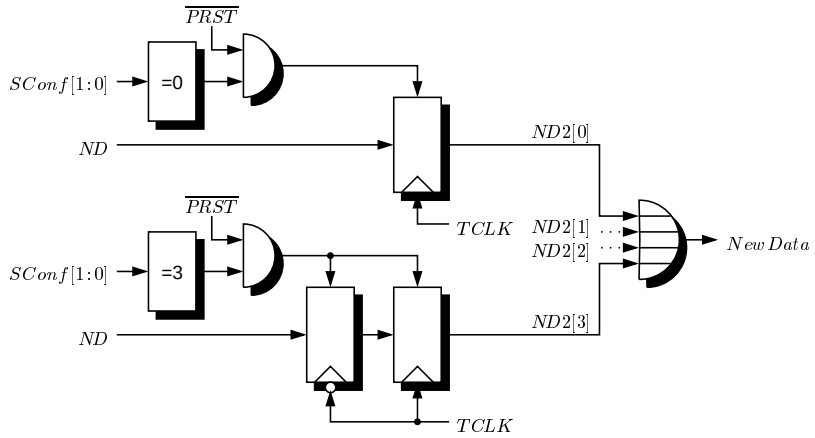

Abbildung 5.15: Der Schaltkreis Sync

Die Auswahl der Synchronisations-Variante geschieht durch die Konfigurations-Signale $SConf[1:0]$. Abbildung 5.15 zeigt die Teile des Schaltkreises Sync für die Werte 0 und 3 von $\langle SConf[1:0]\rangle$. Die Teile für die Werte 1 und 2 sind analog aufgebaut.
Ist $\langle SConf[1:0]\rangle = 3$, so wird eine relativ konservative Synchronisierungs-Variante ausgewählt: zunächst wird das Signal *ND* mit der negativen und dann mit der positiven Flanke des Clocksignals $TCLK$ der Cache/RAM-Logik geclockt. Ist $\langle SConf[1:0]\rangle = 0$, so wird auf den Synchronisierungs-Schaltkreis verzichtet und nur ein einzelnes Register verwendet. Die Synchronisierungs-Varianten für die Werte 1 und 2 werden in Anhang A beschrieben.
Das Resetsignal für die Register ist nur für den Teil des Schaltkreises inaktiv, der durch $SConf[1:0]$ ausgewählt wird. Dadurch ist höchstens eines der Signale $ND2[2:0]$ aktiv. Das synchronisierte Signal $NewData$ kann also durch ein ODER-Gatter berechnet werden. Durch Aktivieren des Resetsignals $\overline{PRST}$ kann $NewData$ auf den Wert 0 gesetzt werden.

Im den folgenden Abschnitten wird näher auf die beiden Parallelisierer-Stufen eingegangen. Der Einfachheit halber wird nur die Parallelisierung des Signals $GIn[0]$ zu den Signalen $GDO[14:0]$ betrachtet.

5.4.3 Voraussetzungen

Auf dem Rückweg von den RAM-Chips zum Cache kann nicht permanent über $GIn[10]$ ein Clocksignal übertragen werden (siehe Abschnitt 3.3). Die Zählung der Takte in den folgenden Voraussetzungen und Sätzen bezieht sich deshalb immer auf das Clocksignal *CLK* des Sequenzers. Für den Schaltkreis CLKSel wird im folgenden von $\langle CConf[1:0]\rangle = 2$ ausgegangen. Wird über $GIn[10]$ ein

Clocksignal übertragen, so gilt dann $GCLK = \neg GIn[10] = \neg(\neg CLK) = CLK$ (siehe Abschnitt 5.3.1). In einem idealisierten Timing hat also $GCLK$ steigende Flanken, wenn auch CLK steigende Flanken hat.

Wie beim Sequenzer wird das Signal $\overline{PRST}$ bei der Initialisierung und nach jeder Übertragung aktiviert. Man kann also davon ausgehen, dass es vor der Übertragung aktiv war. Es wird deshalb die folgende Voraussetzung benutzt.

Voraussetzung 5.18 Sei $t \in \mathbb{Z}$. Für das Resetsignal des Parallelisierers $\overline{PRST}$ gilt:

$$\overline{PRST}^t = \begin{cases} 0 & \text{falls } t < -2, \\ 1 & \text{falls } t \geq -2. \end{cases}$$

Der Eingangsbus $GIn[10:0]$ des Parallelisierers ist über die Glasfaser-Verbindung direkt mit dem Ausgangsbus $GOut[10:0]$ des Sequenzers verbunden. Für den Korrektheitsbeweis wird deshalb $GIn[10:0] = GOut[10:0]$ angenommen. Für das Signal $GIn[0]$ gilt nach Lemma 5.16 und Satz 5.17:

Lemma 5.19 Seien $t \in \mathbb{Z}$ mit $t < 20$ und $S := 4 \cdot \lfloor \frac{t-4}{5} \rfloor$. Dann gilt für den Eingang $GIn[0]$ des Parallelisierers:

$$GIn[0]^t = 0 \quad \text{falls} \quad t \leq 0 \text{ oder } t \equiv 0 \pmod 5,$$
$$(GIn[0]^t, \ldots, GIn[0]^{t-3}) = GDO2[S+3:S] \quad \text{falls} \quad t > 0 \text{ und } t \equiv 4 \pmod 5.$$

5.4.4 Erste Parallelisierer-Stufe

Datenpfade

Das empfangene Signal $GIn[0]$ wird in der ersten Parallelisierer-Stufe zunächst durch ein 4 Bit breites Shiftregister geschoben (siehe Abbildung 5.16). Durch Aktivieren des Signals *GFce* wird der momentane Inhalt des Shiftregisters in ein weiteres vier Bit breites Register gesichert. Durch das Sichern werden die Ausgänge $GF[3:0]$ der ersten Parallelisierer-Stufe für mehrere Takte stabil gehalten und damit die Frequenz der Signale genügend für die Umwandlung nach CMOS gesenkt.

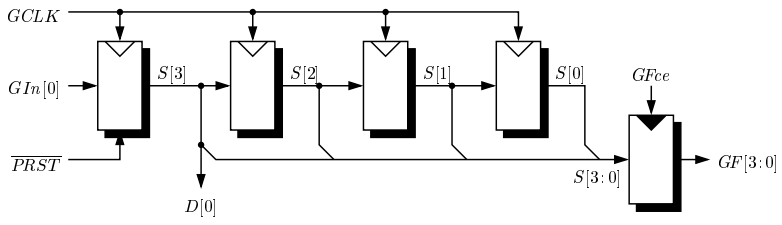

Abbildung 5.16: Datenpfade der ersten Parallelisierer-Stufe

Der Ausgang $D[0]$ des ersten Stufe des Shiftregisters, wird an die Kontrolle übergeben. Dieses Signal wird benutzt, um zu erkennen, ob eine neue Übertragung beginnt. Das Signal kann durch Aktivieren des Resetsignals $\overline{PRST}$ auf den Wert 0 gesetzt werden. Für das Signal $D[0]$ gilt das folgende Lemma.

Lemma 5.20 Sei $t \in \mathbb{Z}$. Für den Ausgang $D[0]$ gilt:

$$D[0]^t = \begin{cases} 0 & \text{falls } t \leq 0, \\ GIn[0]^{t-1} & \text{falls } t > 0. \end{cases}$$

Beweis: Nach Voraussetzung 5.18 gilt für $t \leq -2$:

$$\overline{PRST}^{t-1} = 0 \Longrightarrow D[0]^t = 0.$$

Auf dem Weg vom Cache zu den RAM-Chips wird auf dem Signal $GIn[10]$ immer ein Clocksignal übertragen (siehe Abbildung 5.2). Es gilt also:

$$\begin{aligned} D[0]^{-1} &= GIn[0]^{-2} \overset{5.19}{=} 0, \\ D[0]^0 &= GIn[0]^{-1} \overset{5.19}{=} 0. \end{aligned}$$

Auf dem Rückweg von den RAM-Chips zum Cache wird erst ab dem Takt 1 ein Clock-Signal übertragen. Damit folgt:

$$D[0]^0 = D[0]^{-1} = D[0]^{-2} = 0.$$

Für alle Takte t mit $t > 0$ gilt $GCLK = CLK$ und somit:

$$D[0]^t = GIn[0]^{t-1}. \qquad \blacksquare$$

Damit läßt sich das folgende Lemma über die Ausgänge $GF[3:0]$ der ersten Parallelisierer-Stufe zeigen.

Lemma 5.21 Sei $t \in \mathbb{N}$ mit $t \geq 5$. Für den Ausgang $GF[3:0]$ gilt:

$$GF[3:0]^t = \begin{cases} (GIn[0]^{t-2}, \ldots, GIn[0]^{t-5}) & \text{falls } GFce^{t-1} = 1, \\ GF[3:0]^{t-1} & \text{sonst.} \end{cases}$$

Beweis: Aus $t - 4 > 0$ folgt für alle $j \in \{0, \ldots, 3\}$:

$$S[3]^{t-4+j} \overset{5.20}{=} GIn[0]^{t-5+j}.$$

Es gilt $S[0]^{t-1} = S[1]^{t-2} = S[2]^{t-3} = S[3]^{t-4}$ und somit folgt:

$$S[3:0]^{t-1} = (GIn[0]^{t-2}, \ldots, GIn[0]^{t-5}).$$

Die Behauptung folgt aus der Definition von Registern mit Clock-Enable. $\blacksquare$

Kontrolle

Die Kontrolle der ersten Parallelisierer-Stufe (siehe Abbildungen 5.17 und 5.18) muss zunächst erkennen, ob eine neue Übertragung startet. Dazu wird das Signal *Start* aktiviert, sobald die beiden Signale $D[5]$ und $D[4]$ gleichzeitig den Wert 1 haben. Das Signal *Start* bleibt so lange aktiv, bis der Parallelisierer durch Aktivierung von $\overline{PRST}$ in den Grundzustand gebracht wird. Für das Signal *Start* gilt das folgende Lemma.

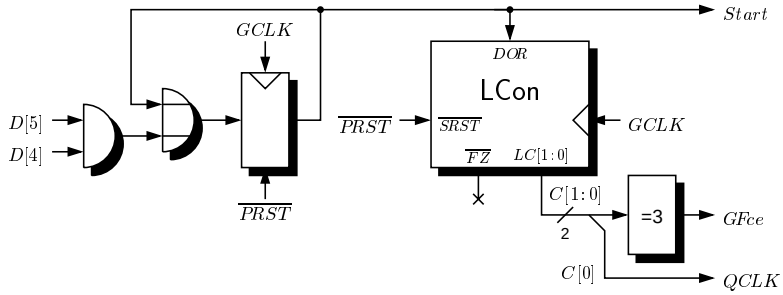

Abbildung 5.17: Kontrolle der ersten Parallelisierer-Stufe

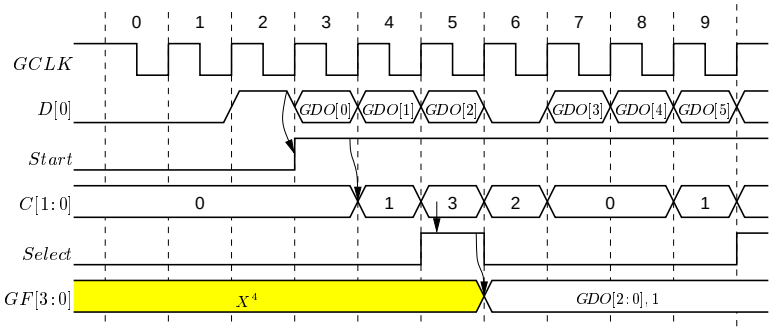

Abbildung 5.18: Timing der ersten Parallelisierer-Stufe

Lemma 5.22 Sei $t \in \mathbb{Z}$. Für das Signal *Start* gilt:

$$Start^t = \begin{cases} 0 & \text{falls } t < 3, \\ 1 & \text{falls } t \geq 3. \end{cases}$$

Beweis: Nach Voraussetzung 5.18 gilt für $t \leq -2$:

$$\overline{PRST}^{t-1} = 0 \text{ und daher } Start^t = 0$$

Für $t \leq 0$ gilt nach Lemma 5.20:

$$(D[4] \wedge D[5])^t = 0.$$

Weiterhin gilt:

$$\begin{aligned} (D[4] \wedge D[5])^1 &= (GIn[4] \wedge GIn[5])^0 \overset{5.19}{=} 0, \\ (D[4] \wedge D[5])^2 &= (GIn[4] \wedge GIn[5])^1 \overset{5.19}{=} 1. \end{aligned}$$

Daraus folgt $Start^t = 0$ für alle $t < 3$ und $Start^3 = 1$. Aus $Start^3 = 1$ folgt $Start^t = 1$ für alle $t \geq 3$ und damit die Behauptung. ■

Mit dem Signal *Start* wird der Schaltkreis LCon (siehe Seite 48) gesteuert. Das Signal entspricht dem Signal *DOR* in der Beschreibung dieses Schaltkreises. Der Ausgang $LC[1:0]$ von LCon wird im folgenden mit $C[2:0]$ bezeichnet. Aus $C[2:0]$ wird das Clockenable-Signal *GFce* für die Datenpfade der ersten Parallelisierer-Stufe und das Clock-Signal *QCLK* für die zweite Parallelisierer-Stufe berechnet. Analog zum Korollar 5.11 ergibt sich das folgende Lemma.

Lemma 5.23 Sei $t \in \mathbb{Z}$ und $r := t - 3 \bmod 5$. Für die Signale $C[1:0]$ der Kontrolle der ersten Parallelisierer-Stufe gilt:

$$C[1:0]^t = \begin{cases} 0 & \text{falls } t < 3, \\ \gamma^r(00) & \text{falls } t \geq 3. \end{cases}$$

Da *GFce* genau dann aktiv ist, wenn $\langle C[1:0] \rangle = 3$ ist, und $QCLK = C[0]$ ist, gelten die beiden folgenden Korollare.

Korollar 5.24 Sei $t \in \mathbb{Z}$. Für den Ausgang *GFce* der Kontrolle der ersten Parallelisierer-Stufe gilt:

$$GFce^t = \begin{cases} 1 & \text{falls } t \equiv 0 \pmod 5 \text{ und } t \geq 3, \\ 0 & \text{sonst.} \end{cases}$$

Korollar 5.25 Sei $t \in \mathbb{Z}$. Dann gilt für den Ausgang $QCLK$ der ersten Parallelisierer-Stufe:

$$QCLK^{t-1} = 0 \wedge QCLK^t = 1 \Longleftrightarrow t \equiv 4 \pmod 5 \text{ und } t \geq 3.$$

Ein Takt des Clocksignal $QCLK$ entspricht also fünf Takten des Clocksignals CLK. Bei einer Frequenz von 320 MHz des Signals CLK ergibt sich für $QCLK$ eine Frequenz von 64 MHz. Es kann also problemlos in CMOS umgewandelt werden.

Zusammenfassung

Fasst man die Lemmas für die Datenpfade und die Kontrolle der ersten Parallelisierer-Stufe zusammen, ergibt sich der folgende Satz für die Ausgänge $GF[3:0]$.

Satz 5.26 Sei $t \in \{6, \ldots, 25\}$ und sei $S := 4 \cdot \lfloor \frac{t-6}{5} \rfloor$. Für den Ausgang $GF[3:0]$ der ersten Parallelisierer-Stufe gilt:

$$GF[3:0]^t = \begin{cases} GDO2[S+3:S] & \text{falls } t \equiv 1 \pmod 5, \\ GF[3:0]^{t-1} & \text{sonst.} \end{cases}$$

Beweis: Aus Korollar 5.24 folgt für $t \in \{6, \ldots, 25\}$:

$$GFce^{t-1} = 1 \Longleftrightarrow t - 1 \equiv 0 \pmod 5.$$

Zusammen mit Lemma 5.21 folgt:

$$GF[3:0]^t = \begin{cases} (GIn[0]^{t-2}, \ldots, GIn[0]^{t-5}) & \text{falls } t \equiv 1 \pmod 5, \\ GF[3:0]^{t-1} & \text{sonst.} \end{cases}$$

Für $t \equiv 1 \pmod 5$ und $t \leq 25$ gilt $t - 2 < 20$ und $t - 2 \equiv 4 \pmod 5$. Es folgt mit Lemma 5.19:

$$GF[3:0]^t = GDO2[S+3:S]. \qquad \blacksquare$$

Die Ausgänge $GF[3:0]$ der ersten Parallelisierer-Stufe sind also jeweils für fünf Takte des Clocksignals CLK stabil. Sie können also auch in CMOS umgewandelt werden.

5.4.5 Zweite Parallelisierer-Stufe

Datenpfade

Die Datenpfade der zweiten Parallelisierer-Stufe bestehen lediglich aus 15 Registern (siehe Abbildung 5.19). In diese Register wird nacheinander der Wert des Busses $GF[3:0]$ geclockt. Das Register für die konstante Eins ($GDO2[0]$) wird weggelassen.

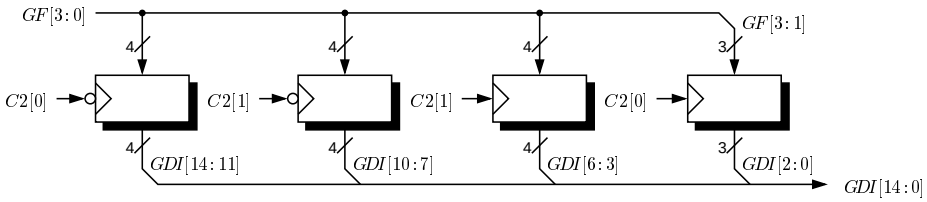

Abbildung 5.19: Datenpfade der zweiten Parallelisierer-Stufe

Kontrolle

In der Kontrolle der zweiten Parallelisierer-Stufe wird das Startsignal *Start* noch einmal um einen Takt des Clocksignals *QCLK* verzögert (siehe Abbildungen 5.20 und 5.21). Dieses Signal dient als Startsignal für den Zähler GCC. Die Ausgangssignale $C2[1:0]$ des Zählers werden zum Clocken der Register in den Datenpfaden benutzt.

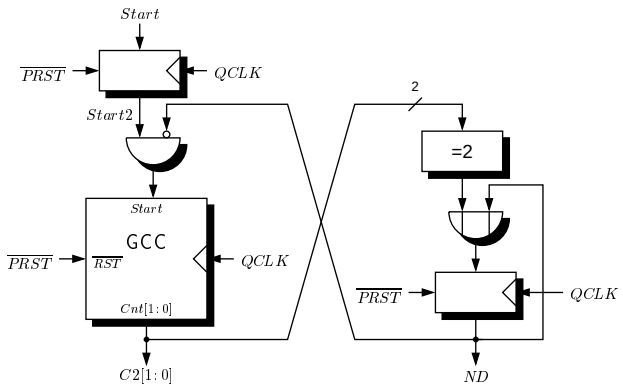

Abbildung 5.20: Kontrolle der zweiten Parallelisierer-Stufe

Damit die Register in den Datenpfaden nicht wieder überschrieben werden, muss der Zähler nach einem Durchlauf angehalten werden. Wie beim Schaltkreis LCon des Sequenzers wird das Signal *ND* auf 1 gesetzt, sobald der Ausgang des Zählers $C2[1:0]$ den Wert 2 hat. Ist *ND* aktiv, so wird der Zähler angehalten. Im Gegensatz zum Schaltkreis LCon wird *ND* aber auf dem Wert 1 gehalten, bis des Resetsignal $\overline{PRST}$ aktiviert wird. Zusätzlich wird das Signal *ND* benutzt, um anzuzeigen, dass die Übertragung beendet ist.

Wie sich aus Korollar 5.25 ergibt, hat das Signal *QCLK* nur zu Beginn von Takten $t \in \mathbb{N}$ mit $t \equiv 4 \pmod 5$ eine positive Flanke. Es genügt also, in allen

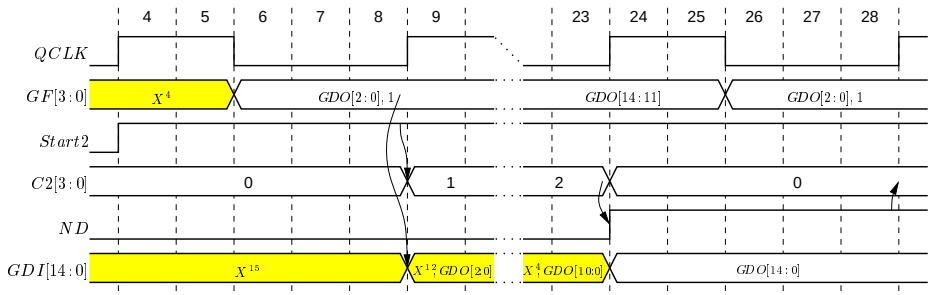

Abbildung 5.21: Timing der zweiten Parallelisierer-Stufe

folgenden Lemmas nur Takte $t \in \mathbb{Z}$ mit $t \equiv 4 \pmod 5$ zu betrachten.

Lemma 5.27 Sei $t \in \mathbb{Z}$ mit $t \equiv 4 \pmod 5$. Dann gilt für das Signal $Start2$:

$$Start2^t = \begin{cases} 0 & \text{falls } t < 4, \\ 1 & \text{falls } t \geq 4. \end{cases}$$

Beweis: Für $t \leq -2$ gilt mit Voraussetzung 5.18:

$$\overline{PRST}^{t-1} = 0 \text{ und damit } Start2^t = 0.$$

Für $-2 < t < 4$ folgt aus Korollar 5.25, dass $QCLK$ keine steigende Flanke hat und damit:

$$Start2^t \quad = \quad Start2^{t-5} = 0.$$

Für $t \geq 4$ folgt aus Lemma 5.22:

$$Start2^t \quad = \quad Start^{t-1} = 1. \qquad \blacksquare$$

Lemma 5.28 Sei $t \in \mathbb{Z}$ mit $t \equiv 4 \pmod 5$. und sei $s := \lfloor \frac{t-4}{5} \rfloor$. Dann gilt für $C2[1:0]$:

$$\begin{aligned} C2[1:0]^t &= \begin{cases} \gamma^s(00) & \text{falls } 4 \leq t < 29, \\ 00 & \text{sonst}, \end{cases} \\ ND^t &= \begin{cases} 0 & \text{falls } t < 24, \\ 1 & \text{falls } t \geq 24. \end{cases} \end{aligned}$$

Beweis: Analog zum Beweis von Lemma 5.27 gilt für $t < 4$:

$$\begin{aligned} C2[1:0]^t &= 00, \\ ND^t &= 0. \end{aligned}$$

Sei nun $t \geq 4$. Induktiv folgt mit $\min\{k \in \mathbb{N} | \langle \gamma^k(00) \rangle = 2\} = 3$ für $t < 24$:

$$\begin{aligned} ND^{t-5} = 0 &\implies C2[1:0]^t = \gamma^s(C2[1:0]^{t-5}) = \gamma^s(00), \\ \langle C2[1:0]^{t-5} \rangle \neq 2 &\implies ND^t = 0. \end{aligned}$$

Für $t = 24$ folgt daraus:

$$\begin{aligned} ND^{19} = 0 &\implies C2[1:0]^{24} = \gamma^4(00) = \gamma^s(00), \\ \langle C2[1:0]^{19} \rangle = 2 &\implies ND^{24} = 1. \end{aligned}$$

Aus $ND^{24} = 1$ folgt für alle $t \geq 24$ $ND^t = 1$ und damit $C2[1:0]^{t+5} = 00$. ■

Korollar 5.29 Sei $t \in \mathbb{Z}$. Dann gilt für den Ausgang $C2[1:0]$ der Kontrolle:

$$\begin{aligned} C2[0]^{t-1} = 0 \wedge C2[0]^t = 1 &\iff t = 9, \\ C2[1]^{t-1} = 0 \wedge C2[1]^t = 1 &\iff t = 14, \\ \neg C2[0]^{t-1} = 0 \wedge \neg C2[0]^t = 1 &\iff t = 19, \\ \neg C2[1]^{t-1} = 0 \wedge \neg C2[1]^t = 1 &\iff t = 24. \end{aligned}$$

Beweis: Die Behauptung folgt aus der Definition von γ. ■

Zusammenfassung

Wendet man das Korollar 5.29 auf die Datenpfade der zweiten Parallelisierer-Stufe an, ergibt sich der abschließende Satz über die Glasfaser-Übertragung.

Satz 5.30 Sei $t \in \mathbb{N}$ mit $t \geq 24$. Dann gilt für den Ausgang $GDI[14:0]$ des Parallelisierers:

$$GDI^t[14:0] = GDO[14:0].$$

Beweis: Aus Satz 5.26 und Korollar 5.29 folgt:

$$\begin{aligned} GDI[2:0]^t \overset{5.29}{=} GF[3:1]^8 &\overset{5.26}{=} GDO2[3:1] \overset{(5.1)}{=} GDO[2:0] \ \forall t \geq 9, \\ GDI[6:3]^t \overset{5.29}{=} GF[3:0]^{13} &\overset{5.26}{=} GDO2[7:4] \overset{(5.1)}{=} GDO[6:3] \ \forall t \geq 14, \\ GDI[10:7]^t \overset{5.29}{=} GF[3:0]^{18} &\overset{5.26}{=} GDO2[11:8] \overset{(5.1)}{=} GDO[10:7] \ \forall t \geq 18, \\ GDI[14:11]^t \overset{5.29}{=} GF[3:0]^{23} &\overset{5.26}{=} GDO2[15:12] \overset{(5.1)}{=} GDO[14:11] \ \forall t \geq 24. \end{aligned}$$

Für $t \geq 24$ gilt also die Behauptung. ■

Vernachlässigt man die Verzögerung der Glasfaser-Verbindung, liegen die übertragenen Daten also 24 Takte nach Aktivieren von *DOR* am Bus $GDI[14:0]$ an. Gleichzeitig wird das Signal *ND* aktiviert. Addiert man die maximale Dauer von zwei Takten für die Synchronisation von *DOReady* zu *DOR* ergibt sich, dass die Glasfaser-Übertragung 26 Takte von *CLK* dauert.
Damit genügend steigende Flanken auf dem Signal $GOut[10]/GIn[10]$ übertragen werden, muss *DOReady* also für mindestens 26 Takte der schnellen Clock aktiv sein. Dies ist zu beachten, wenn andere Takt-Verhältnisse als 32:1 zwischen CMOS-Clock und ECL-Clock verwendet werden sollen. Liegt das Verhältnis zwischen 13:1 und 25:1 kann das Konfigurations-Signal *SDD* aktiviert werden, um die Übertragung auf zwei Takte der langsamen Clock zu verlängern (siehe Anhang A).

Kapitel 6

Realisierung

In diesem Kapitel werden die einzelnen Schritte beschrieben, die zur Realisierung des Dies notwendig waren. In Abschnitt 6.1 wird das verwendete Design-System vorgestellt. Die Abschnitte 6.2 und 6.3 behandeln die Erstellung des logischen und des physikalischen Designs. Auf die Herstellung des Dies und die abschließenden Fertigungstests wird in Abschnitt 6.4 eingegangen.

6.1 Design-Umgebung

Für die Erstellung des Designs wurde das Design-System *Cadence* [8] verwendet. Dieses Design-System enthält für jeden Schritt zur Erstellung des Designs ein spezialisiertes Programm. Da zum Zeitpunkt dieser Diplomarbeit am Lehrstuhl von Professor Paul an der Universität des Saarlandes niemand Erfahrung im Umgang mit *Cadence* hatte, bestand ein nicht unwesentlicher Teil der Arbeit darin, die Programme einzurichten und den Umgang mit ihnen zu erlernen.
Zusätzlich waren die Programme größtenteils auf reine CMOS-Designs ausgerichtet, so daß die den ECL-Teil betreffenden Arbeitsschritte sehr aufwendig waren und größtenteils von Hand durchgeführt werden mussten. Dies hat dazu geführt, dass nachträgliche Änderungen im logischen oder physikalischen Design sehr aufwendig waren. Kurz vor Fertigstellung des Designs wurde ein grundlegender logischer Fehler gefunden, der die Fertigstellung des Chips noch einmal um ein halbes Jahr herausgezögert hat.

6.2 Erstellung des logischen Designs

Zur Erstellung des logischen Designs bietet Cadence zwei Ansätze an: die direkte Eingabe der Schaltbilder auf Gatterebene (*Schematic Entry*) mit dem Programm *Composer* [9] oder die Verwendung der Hardware-Beschreibungs-Sprachen *Verilog HDL* [14] oder *VHDL* [2]. Wird die Schaltung in einer dieser Sprachen eingegeben, so muss sie anschließend mit dem Programm *Synergy* [11] auf Gatterebene übersetzt werden.
Der größte Teil der Schaltungen wurde direkt auf Gatterebene entworfen. Es bot sich also an, diese Teile auch direkt als Schematics im *Composer* einzugeben. Da der Übersetzungsschritt entfällt, hat man so eine bessere Kontrolle über das

endgültige Layout. Die wichtigsten Schematics des Designs sind in Anhang C abgebildet.
Die Kontrolle der Cache/RAM-Logik wurde hingegen nicht auf Gatterebene entworfen (siehe Abschnitt 4.7). Eine manuelle Übersetzung auf Gatterebene wäre sehr mühevoll und müsste bei jedem Fehler größtenteils wiederholt werden. Die Hardware-Beschreibungs-Sprachen bieten in diesem Fall die Möglichkeit, das Design wesentlich einfacher und übersichtlicher einzugeben. Auf diese Weise können Fehler bequem behoben werden. Es wurde die Sprache *Verilog HDL* verwendet. Der Quellcode der Kontrolle findet sich in Anhang D.

Zum Überprüfen der Korrektheit wurden die Schaltkreise zunächst simuliert. Als eine der Hauptschwierigkeiten stellte sich dabei die Verwendung von ECL heraus. Mit dem schnellen Digital-Simulator *Verilog XL* [12] ist es nicht möglich, ECL-Gatter zu simulieren. Der Analog-Simulator *Spectre* [7] hingegen, mit dem ECL-Gatter simuliert werden können, ist zur Simulation größerer Schaltkreise zu langsam. Die Simulation von 800ns (also 8 Takten der langsamen Clock) mit diesem Simulator dauert ungefähr 24 Stunden.
Um während der Designphase schnell simulieren zu können, wurde zunächst der gesamte Chip in CMOS eingegeben und mit *Verilog XL* simuliert. Nachdem die Simulationen für das CMOS-Design fehlerfrei beendet wurden, wurden die in ECL zu realisierenden Schaltkreise (hauptsächlich die Schaltkreise Seq und Par1) von Hand nach ECL übersetzt. Abschließend wurde das gesamte Design mit dem Analog-Simulator überprüft, um eventuelle Fehler beim Übersetzen des Designs zu finden.

Wie bereits erwähnt wurde, ist zu einem relativ späten Zeitpunkt der Designphase ein grundlegender Fehler im Design von Sequenzer und Parallelisierer aufgetaucht, der mit den Simulationen nicht gefunden wurde. Um beim erneuten Design nicht wieder einem solchen Fehler zu erliegen, wurde beim zweiten Anlauf für die Glasfaser-Übertragung zusätzlich ein Korrektheitsbeweis geführt (siehe Abschnitte 5.3 und 5.4).

6.3 Erstellung des physikalischen Designs

Die Erstellung des physikalischen Designs aus dem logischen Design, also das Platzieren der Gatter auf dem Die und das Legen der Leitungen zwischen den Gattern (*Place and Route*), hat einen Großteil der Zeit in Anspruch genommen. Zunächst wurde ein grundlegendes Design für das Placing des Dies erstellt. Da es nicht sinnvoll ist, die in CMOS und ECL realisierten Teile zu mischen, wurde der Die in drei Teile aufgeteilt (siehe Abbildung 6.1): an zwei Seiten des Dies werden die beiden ECL-Teile mit den Schaltkreisen Seq und Par1 und den dazugehörigen ECL-Pads platziert, der CMOS-Teil liegt dazwischen. Der CMOS-Teil besteht aus den CMOS-Pads, den RAM-Bausteinen und dem CMOS-Core, der alle übrigen CMOS-Gatter enthält (auch diejenigen von Parallelisierer und Sequenzer).

Vom CMOS-Core ist nur der Schaltkreis Par2 (siehe Abschnitt 5.4.5), der mit einer höheren Frequenz getaktet wird, zeitkritisch. Aus diesem Grund wurden die

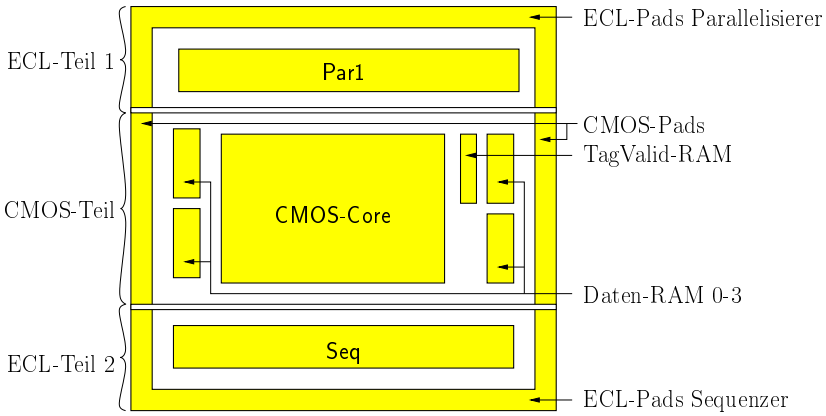

Abbildung 6.1: Überblick über das Placing

Gatter dieses Schaltkreises zunächst von Hand vorplatziert. Anschließend wurde der gesamte CMOS-Core unter Berücksichtigung der vorplatzierten Gatter mit dem Programm *Cell Ensemble* [10] automatisch platziert und geroutet.

Ein automatisches Platzieren und Routen der ECL-Teile kam nicht in Frage, da das Programm nicht für ECL-Gatter ausgelegt ist und die Ergebnisse für diese zeitkritischen Schaltkreise zu schlecht waren. Die ECL-Teile mussten also mit dem *Virtuoso Layout Editor* [13] komplett von Hand platziert und geroutet werden.
Nachdem die beiden ECL-Teile und der CMOS-Core platziert und geroutet waren, mussten die Teile zusammengefügt und mit den RAM-Bausteinen und den CMOS-Pads verbunden werden. Da es auch hier Probleme mit dem automatischen Routen gab, musste das abschließende Routen ebenfalls von Hand geschehen.

Nachdem der Die platziert und geroutet wurde, musste er auf Korrektheit geprüft werden. Dazu wurde zunächst ein *design rule check* (DRC) des Designs durchgeführt. Der DRC prüft die vom Hersteller vorgegebenen Mindestabstände zwischen den Leitungen. Die vom DRC gefundenen Fehler konnten meist schnell behoben werden.
Anschließend wurde mit Hilfe des *logic verification system* (LVS) das platzierte Design mit dem logischen Design verglichen. Dies war wichtig, da große Teile des Dies von Hand geroutet wurden. Da ein Fehler oft viele Fehlermeldungen an anderen (korrekten) Stellen erzeugt, sind diese Fehler oft sehr schwer zu lokalisieren. Dadurch konnten die Fehler meist auch nur einer nach dem anderen behoben werden. Hinzu kommt, dass ein Test des gesamten Designs mit dem LVS mehrere Stunden dauert, so dass dieser Schritt insgesamt viel Zeit in Anspruch genommen hat.

6.4 Fertigung und Fabrikationstest

Das fertige Design wurde am 29. September 2000 bei der Firma AMS in Produktion gegeben. Bei der Herstellung eines Design dieser Größe kann davon

ausgegangen werden, das nur ca. 30% der produzierten Dies fehlerfrei funktionieren. Da das Anbringen der Glasfaser-Anbindung sehr aufwendig ist, werden die Dies vor der Weiterverarbeitung von der Firma Delta [16] getestet.
Zum Testen der Dies wurden Testvektoren erstellt [19]. Jeder Testvektor besteht aus einem Stimulusvektor und einem Strobevektor. Die Einträge des Stimulusvektors entsprechen den Eingangspads des Dies, die Einträge des Strobevektors entsprechen den Ausgangspads. Der Stimulusvektor wird über feine Nadeln an den Eingangspads angelegt. Nach einer kurzen Pause wird der Strobevektor ebenfalls über feine Nadeln ausgelesen und mit dem erwarteten Wert verglichen. Anschließend wird der nächste Testvektor bearbeitet. Der gesamte Fabrikationstest umfaßt ungefähr 70.000 Vektoren. Die Vektoren können unter [34] heruntergeladen werden.
Da die maximale Testfrequenz bei 50 MHz liegt, können mit diesen Tests zwar keine Timing-Probleme gefunden werden, aber zumindest wird das logische Verhalten der Dies getestet. Das Ergebnis dieser Tests steht noch aus.

Kapitel 7

Zusammenfassung und Ausblick

Die Bandbreite bei Datenübertragungen zwischen Chips ist auf elektrischem Wege nahezu vollständig ausgereizt. Eine Erhöhung der Bandbreite um ganze Größenordnungen ist allerdings mit Hilfe optischer Übertragung möglich. Um die Vorteile von Glasfasern in der Kommunikation zwischen Chips zu zeigen, wird am Lehrstuhl für Rechnerarchitektur der Universität des Saarlandes in Zusammenarbeit mit dem Lehrstuhl für Technische Physik von der DLR in Stuttgart ein optisch gekoppelter Cache entwickelt [31].
In dieser Arbeit wurde das Design der inneren Logik des optisch gekoppelten Caches erstellt. Um die Daten mit der gewünschten Frequenz über die Glasfaser-Verbindung übertragen zu können, musste ein kombinierter CMOS/ECL-Die entwickelt werden. Da die ECL-Technologie sehr teuer ist, wurden die in ECL realisierten Schaltkreise in Hinblick auf die Kosten stark optimiert. Von diesen Schaltkreisen wurde eine Korrektheitsbeweis geführt. Die Dies wurden von der Firma AMS hergestellt und befinden sich momentan (Februar 2001) beim Fertigungstest.

Ausblick

Sobald die Fertigungstests abgeschlossen sind, werden die funktionierenden Dies zusammen mit der Glasfaser-Anbindung vom Instut für Technische Physik in Multichip-Modulen integriert. Aus insgesamt fünf der Multichip-Module wird der Prototyp des optisch gekoppelten Caches gebaut. Dieser Prototyp wird auf eine Platine montiert, die von Michael Klein [30] entwickelt wird, und kann dort getestet werden.

Um den Prototypen zu einer industrietauglichen Speicheranbindung weiterzuentwickeln, sind die folgenden Punkte zu beachten.

- Die Bandbreite ist bei der verwendeten Anzahl von Glasfaserkabeln noch nicht wesentlich höher als bei einer elektrischen Übertragung. Damit sich der höhere Aufwand lohnt, muss neben der Frequenz vor allem die Anzahl der Glasfaserkabel um eine Größenordnung erhöht werden.

- Eine größere Anzahl von Glasfaserkabeln ist in einem Multichip-Modul nur mit sehr großem Aufwand möglich. Ein kommerzieller optischer Cache

sollte also auf jeden Fall mit Flip-Chip Bonding an die optische Anbindung angeschlossen werden. Durch die geringere Kapazität der Verbindung zwischen Logik und optischer Anbindung könnte auch auf die Verwendung von ECL verzichtet werden.

- Die Lücke zwischen Prozessor und Cache muss auf jeden Fall geschlossen werden. Der optisch gekoppelter Cache sollte also zusammen mit dem Prozessor auf einem Die implementiert werden.
- Der Hauptspeicher des Prototypen ist viel zu klein. Um eine konkurrenzfähige Größe zu erreichen, müßte die Glasfaseranbindung in kommerzielle DRAM-Chips integriert werden.

Eine wirklich konkurrenzfähige optische Anbindung des Caches kann also nur mit sehr großem Aufwand entwickelt werden. Dies ist ohne Industrie-Beteiligung im Rahmen eines Universitätsprojektes nicht möglich.

Anhang A

Konfigurations-Signale

Tabelle A.1 fasst die zur Konfiguration des Chips verwendeten Signale zusammen.

Name	Bedeutung
CMod	Mit dem Signal *CMod* kann zwischen der Cache- und der RAM-Funktion der Chips umgeschaltet werden. Ist *CMod* aktiv, arbeitet der Chip als Cache, andernfalls als RAM.
Test	Ist das Signal *Test* aktiv, so wird der Chip in den Test-Modus für die Glasfaser-Verbindung versetzt (siehe Abschnitt 4.8)
$Range[1:0]$	Um die vier RAM-Chips unterscheiden zu können, werden sie durchnummeriert. Der Wert $\langle Range[1:0]\rangle$ gibt die Nummer des RAM-Chips an.
ETest	Durch Deaktivieren von *ETest* kann die Überprüfung des Hammingcodes abgeschaltet werden.
opt	Der Parallelisierer garantiert nicht, dass das Signal *Error* berechnet ist, wenn *NewData* aktiviert wird. Ist *opt* aktiv, so wird *Error* dennoch ohne einen Takt zu warten verwendet.
SDD	Ist *SDD* aktiv wird das Startsignal für die Glasfaser-Übertragung *DOReady* für zwei statt nur einen Takt aktiviert.
$CConf[1:0]$	Mit diesen Signalen wird im Parallelisierer das Delay des empfangenen Clocksignals $GIn[10]$ ausgewählt (siehe Abschnitt 5.4.2).
$SConf[1:0]$	Die Signale $SConf[1:0]$ dient zur Auswahl der Synchronisierungs-Variante im Parallelisierer (siehe Abschnitt 5.4.2).

Tabelle A.1: Bedeutung der Konfigurationsignale

Die Bedeutung der Werte von $CConf[1:0]$ und $SConf[1:0]$ wird in Tabelle A.2 aufgeschlüsselt.

Nur die Signale *CMod* und *Test* werden direkt über Pins eingestellt. Die Einstellung der restlichen Signale erfolgt über ein Schieberegister. Dieses wird über die Pins mit Hilfe des Eingangs CIn und des Clock-Signals $CCLK$ beschrieben

$CConf[1:0]$	00	Verwendung von $GIn[10]$
	01	Verwendung von $\neg GIn[10]$
	10	Verwendung von $GIn[10]$ (verzögert)
	11	Verwendung von $\neg GIn[10]$ (verzögert)

$SConf[1:0]$	00	Synchronisierung mit CLK
	01	Synchronisierung mit $\neg CLK$
	10	Synchronisierung mit CLK und $\neg CLK$
	11	Synchronisierung mit $\neg CLK$ und CLK

Tabelle A.2: Bedeutung der Werte von $CConf[1:0]$ und $SConf[1:0]$

(vergleichbar mit JTAG [25]). Zur Vereinfachung des Test-Modus können diese Signale durch Anlegen des Signals FC über die Pins deaktiviert werden.

Anhang B

Belegungen

B.1 Padbelegung

Die Nummerierung der Pads beginnt in der linken oberen Ecke des Designs (Orientierung entsprechend Abbildung 6.1). Es wird entgegen dem Uhrzeigersinn gezählt.

Nr.	Padname	Padtyp	Level
1	5V_P1	PPVCC2	5V
2	0V_P1	PPVEE1	0V
3	5V_P1	PPVCC1	5V
4	0V_P1	PPVEE1	0V
5	5V_P1	PP00P	5V
6	0V_P1	PP00P	0V
7	0V_P2	PP05P	0V
8	5V_P2	PP06P	5V
9	D(31)	IO43B	TTL
10	D(30)	IO43B	TTL
11	D(29)	IO43B	TTL
12	D(28)	IO43B	TTL
13	D(27)	IO43B	TTL
14	D(26)	IO43B	TTL
15	D(25)	IO43B	TTL
16	D(24)	IO43B	TTL
17	D(23)	IO43B	TTL
18	D(22)	IO43B	TTL
19	D(21)	IO43B	TTL
20	D(20)	IO43B	TTL
21	D(19)	IO43B	TTL

Nr.	Padname	Padtyp	Level
22	D(18)	IO43B	TTL
23	D(17)	IO43B	TTL
24	D(16)	IO43B	TTL
25	5V_P2	PP02P	5V
26	0V_P2	PP01P	0V
27	D(15)	IO43B	TTL
28	D(14)	IO43B	TTL
29	D(13)	IO43B	TTL
30	D(12)	IO43B	TTL
31	D(11)	IO43B	TTL
32	D(10)	IO43B	TTL
33	D(9)	IO43B	TTL
34	D(8)	IO43B	TTL
35	D(7)	IO43B	TTL
36	D(6)	IO43B	TTL
37	D(5)	IO43B	TTL
38	D(4)	IO43B	TTL
39	D(3)	IO43B	TTL
40	D(2)	IO43B	TTL
41	5V_P2	PP06P	5V
42	0V_P2	PP05P	0V

Tabelle B.1: Padbelegung (Pads 1 -42)

Nr.	Padname	Padtyp	Level
43	0V_P3	PP00P	0V
44	5V_P3	PP00P	5V
45	0V_P3	PPVEE1	0V
46	5V_P3	PPVCC1	5V
47	0V_P3	PPVEE1	0V
48	5V_P3	PPVCC2	5V
49	5V_P3	PPVCC1	5V
50	0V_P3	PPVEE1	0V
51	0V_P3	PPVSUB	0V
52	5V_P3	PPVCC3	5V
53	O_P(0)	EOHD8M	ECL
54	O_N(0)	EOHD8M	ECL
55	5V_P3	PPVCC1	5V
56	O_P(1)	EOHD8M	ECL
57	O_N(1)	EOHD8M	ECL
58	0V_P3	PPVEE2	0V
59	O_P(2)	EOHD8M	ECL
60	O_N(2)	EOHD8M	ECL
61	5V_P3	PPVCC3	5V
62	O_P(3)	EOHD8M	ECL
63	O_N(3)	EOHD8M	ECL
64	0V_P3	PPVEE1	0V
65	5V_P3	PPVCC1	5V
66	O_P(4)	EOHD8M	ECL
67	O_N(4)	EOHD8M	ECL
68	5V_P3	PPVCC3	5V
69	0V_P3	PPVSUB	0V
70	O_P(10)	EOHD8M	ECL
71	O_N(10)	EOHD8M	ECL
72	5V_P3	PPVCC3	5V
73	O_P(5)	EOHD8M	ECL
74	O_N(5)	EOHD8M	ECL
75	0V_P3	PPVEE1	0V
76	0V_P3	PPVEE2	0V
77	O_P(6)	EOHD8M	ECL
78	O_N(6)	EOHD8M	ECL
79	5V_P3	PPVCC3	5V
80	O_P(7)	EOHD8M	ECL
81	O_N(7)	EOHD8M	ECL
82	5V_P3	PPVCC1	5V
83	O_P(8)	EOHD8M	ECL

Nr.	Padname	Padtyp	Level
84	O_N(8)	EOHD8M	ECL
85	0V_P3	PPVEE2	0V
86	O_P(9)	EOHD8M	ECL
87	O_N(9)	EOHD8M	ECL
88	5V_P3	PPVCC3	5V
89	0V_P3	PPVSUB	0V
90	5V_P3	PPVCC1	5V
91	0V_P3	PPVEE1	0V
92	C_P	EIHD8M	ECL
93	C_N	EIHD8M	ECL
94	0V_P3	PPVEE2	0V
95	5V_P3	PPVCC2	5V
96	5V_P3	PP00P	5V
97	0V_P3	PP00P	0V
98	0V_P2	PP03P	0V
99	5V_P2	PP04P	5V
100	D(0)	IO43B	TTL
101	D(1)	IO43B	TTL
102	0V_P2	PP05P	0V
103	5V_P2	PP06P	5V
104	AACK	IO43B	TTL
105	DACK	IO43B	TTL
106	0V_P2	PP03P	0V
107	5V_P2	PP04P	5V
108	CCLK	IB95B	TTL
109	CIN	IB95B	TTL
110	CMOD	IB95B	TTL
111	FC	IB95B	TTL
112	TST	IB95B	TTL
113	RW	IB95B	TTL
114	DV	IB95B	TTL
115	AV	IB95B	TTL
116	RST	IB95B	TTL
117	A(0)	IB95B	TTL
118	A(1)	IB95B	TTL
119	A(2)	IB95B	TTL
120	A(3)	IB95B	TTL
121	A(4)	IB95B	TTL
122	A(5)	IB95B	TTL
123	A(6)	IB95B	TTL
124	A(7)	IB95B	TTL

Tabelle B.2: Padbelegung (Pads 43 - 124)

Nr.	Padname	Padtyp	Level
125	A(8)	IB95B	TTL
126	A(9)	IB95B	TTL
127	A(10)	IB95B	TTL
128	A(11)	IB95B	TTL
129	0V_P2	PP03P	0V
130	5V_P2	PP04P	5V
131	CLK	IB95B	TTL
132	5V_P2	PP04P	5V
133	0V_P2	PP03P	0V
134	0V_P1	PP00P	0V
135	5V_P1	PP00P	5V
136	0V_P1	PPVEE1	0V
137	5V_P1	PPVCC1	5V
138	0V_P1	PPVEE1	0V
139	5V_P1	PPVCC2	5V
140	0V_P1	PPVSUB	0V
141	I_P(9)	EIHD8M	ECL
142	I_N(9)	EIHD8M	ECL
143	0V_P1	PPVEE1	0V
144	I_P(8)	EIHD8M	ECL
145	I_N(8)	EIHD8M	ECL
146	0V_P1	PPVEE2	0V
147	I_P(7)	EIHD8M	ECL
148	I_N(7)	EIHD8M	ECL
149	5V_P1	PPVCC2	5V
150	I_P(6)	EIHD8M	ECL

Nr.	Padname	Padtyp	Level
151	I_N(6)	EIHD8M	ECL
152	0V_P1	PPVEE1	0V
153	5V_P1	PPVCC1	5V
154	I_P(5)	EIHD8M	ECL
155	I_N(5)	EIHD8M	ECL
156	0V_P1	PPVEE1	0V
157	I_P(10)	EIHD8M	ECL
158	I_N(10)	EIHD8M	ECL
159	0V_P1	PPVEE1	0V
160	0V_P1	PPVSUB	0V
161	I_P(4)	EIHD8M	ECL
162	I_N(4)	EIHD8M	ECL
163	0V_P1	PPVEE1	0V
164	5V_P1	PPVCC2	5V
165	I_P(3)	EIHD8M	ECL
166	I_N(3)	EIHD8M	ECL
167	5V_P1	PPVCC1	5V
168	I_P(2)	EIHD8M	ECL
169	I_N(2)	EIHD8M	ECL
170	0V_P1	PPVEE2	0V
171	I_P(1)	EIHD8M	ECL
172	I_N(1)	EIHD8M	ECL
173	0V_P1	PPVEE1	0V
174	I_P(0)	EIHD8M	ECL
175	I_N(0)	EIHD8M	ECL
176	0V_P1	PPVSUB	0V

Tabelle B.3: Padbelegung (Pads 125 - 176)

B.2 Belegung der Glasfasern

Die zu übertragenden Daten werden mit dem in Abschnitt 5.2 beschriebenen Protokoll über die Glasfasern gesendet (siehe auch Abbildung 5.1). In der Tabelle B.4 werden die zu übertragenden Daten den Werten $D[0]$ bis $D[14]$ aus dem Protokoll zugeordnet.

	$GFOut[0]$	$GFOut[1]$	$GFOut[2]$	$GFOut[3]$	$GFOut[4]$
$D[0]$	$GDO[0]$	$GDO[15]$	$GDO[30]$	$GDO[45]$	$GDO[59]$
$D[1]$	$GDO[1]$	$GDO[16]$	$GDO[31]$	$GDO[46]$	$GDO[60]$
$D[2]$	$GDO[2]$	$GDO[17]$	$GDO[32]$	$GDO[47]$	$GDO[61]$
$D[3]$	$GDO[3]$	$GDO[18]$	$GDO[33]$	$GDO[48]$	$GDO[62]$
$D[4]$	$GDO[4]$	$GDO[19]$	$GDO[34]$	$GDO[49]$	$GDO[63]$
$D[5]$	$GDO[5]$	$GDO[20]$	$GDO[35]$	$GDO[50]$	$GDO[64]$
$D[6]$	$GDO[6]$	$GDO[21]$	$GDO[36]$	$GDO[51]$	$GDO[65]$
$D[7]$	$GDO[7]$	$GDO[22]$	$GDO[37]$	$GDO[52]$	$GDO[66]$
$D[8]$	$GDO[8]$	$GDO[23]$	$GDO[38]$	$GDO[53]$	$GDO[67]$
$D[9]$	$GDO[9]$	$GDO[24]$	$GDO[39]$	$GDO[54]$	$GDO[68]$
$D[10]$	$GDO[10]$	$GDO[25]$	$GDO[40]$	$GDO[55]$	$GDO[69]$
$D[11]$	$GDO[11]$	$GDO[26]$	$GDO[41]$	$GDO[56]$	$GDO[70]$
$D[12]$	$GDO[12]$	$GDO[27]$	$GDO[42]$	$GDO[57]$	$GDO[71]$
$D[13]$	$GDO[13]$	$GDO[28]$	$GDO[43]$	$GDO[58]$	$GDO[72]$
$D[14]$	$GDO[14]$	$GDO[29]$	$GDO[44]$	$GDO[58]$	$GDO[72]$

	$GFOut[5]$	$GFOut[6]$	$GFOut[7]$	$GFOut[8]$	$GFOut[9]$
$D[0]$	$GDO[73]$	$GDO[87]$	$GDO[101]$	$GDO[115]$	$ErwOut$
$D[1]$	$GDO[74]$	$GDO[88]$	$GDO[102]$	$GDO[116]$	$GAO[2]$
$D[2]$	$GDO[75]$	$GDO[89]$	$GDO[103]$	$GDO[117]$	$GAO[3]$
$D[3]$	$GDO[76]$	$GDO[90]$	$GDO[104]$	$GDO[118]$	$GAO[4]$
$D[4]$	$GDO[77]$	$GDO[91]$	$GDO[105]$	$GDO[119]$	$GAO[5]$
$D[5]$	$GDO[78]$	$GDO[92]$	$GDO[106]$	$GDO[120]$	$GAO[6]$
$D[6]$	$GDO[79]$	$GDO[93]$	$GDO[107]$	$GDO[121]$	$GAO[7]$
$D[7]$	$GDO[80]$	$GDO[94]$	$GDO[108]$	$GDO[122]$	$GPO[0]$
$D[8]$	$GDO[81]$	$GDO[95]$	$GDO[109]$	$GDO[123]$	$GPO[1]$
$D[9]$	$GDO[82]$	$GDO[96]$	$GDO[110]$	$GDO[124]$	$GPO[2]$
$D[10]$	$GDO[83]$	$GDO[97]$	$GDO[111]$	$GDO[125]$	$GPO[3]$
$D[11]$	$GDO[84]$	$GDO[98]$	$GDO[112]$	$GDO[126]$	$GPO[4]$
$D[12]$	$GDO[85]$	$GDO[99]$	$GDO[113]$	$GDO[127]$	$GPO[5]$
$D[13]$	$GDO[86]$	$GDO[100]$	$GDO[114]$	$GAO[0]$	$GPO[6]$
$D[14]$	$GDO[86]$	$GDO[100]$	$GDO[114]$	$GAO[1]$	$GPO[7]$

Tabelle B.4: Aufteilung der zu übertragenden Signale

Anhang C

Schematics

In diesem Anhang sind die wichtigsten Schematics des Designs abgebildet. Abbildung C.1 zeigt die Hierarchie der Schematics. Stellen, an denen Schematics fehlen, werden in der Hierarchie durch eine gepunktete Linie angedeutet. Diese Schematics sind entweder trivial oder entsprechen in wesentlichen Teilen einem abgebildeten Schematic. Sie können unter [34] heruntergeladen werden.

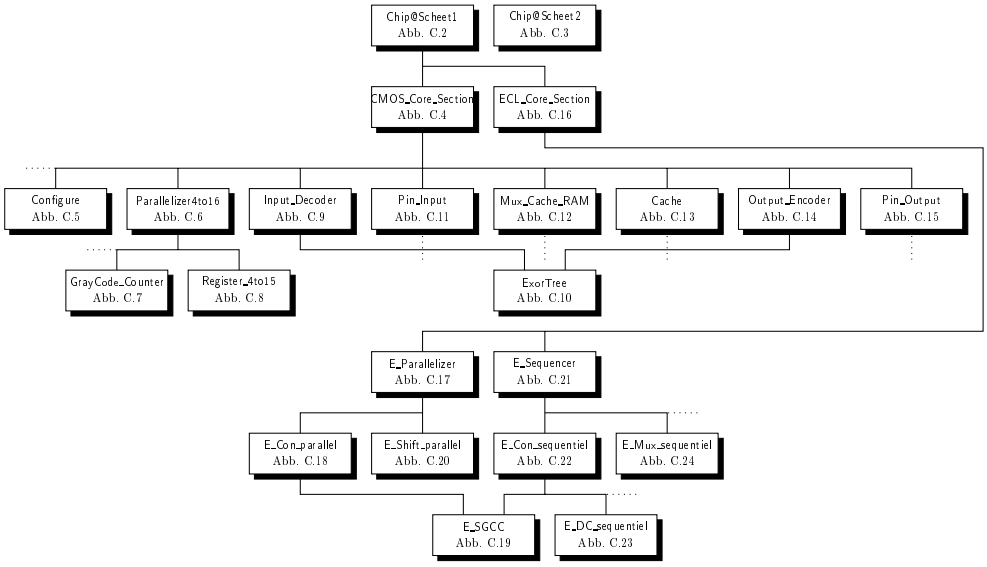

Abbildung C.1: Hierarchie der Schematics

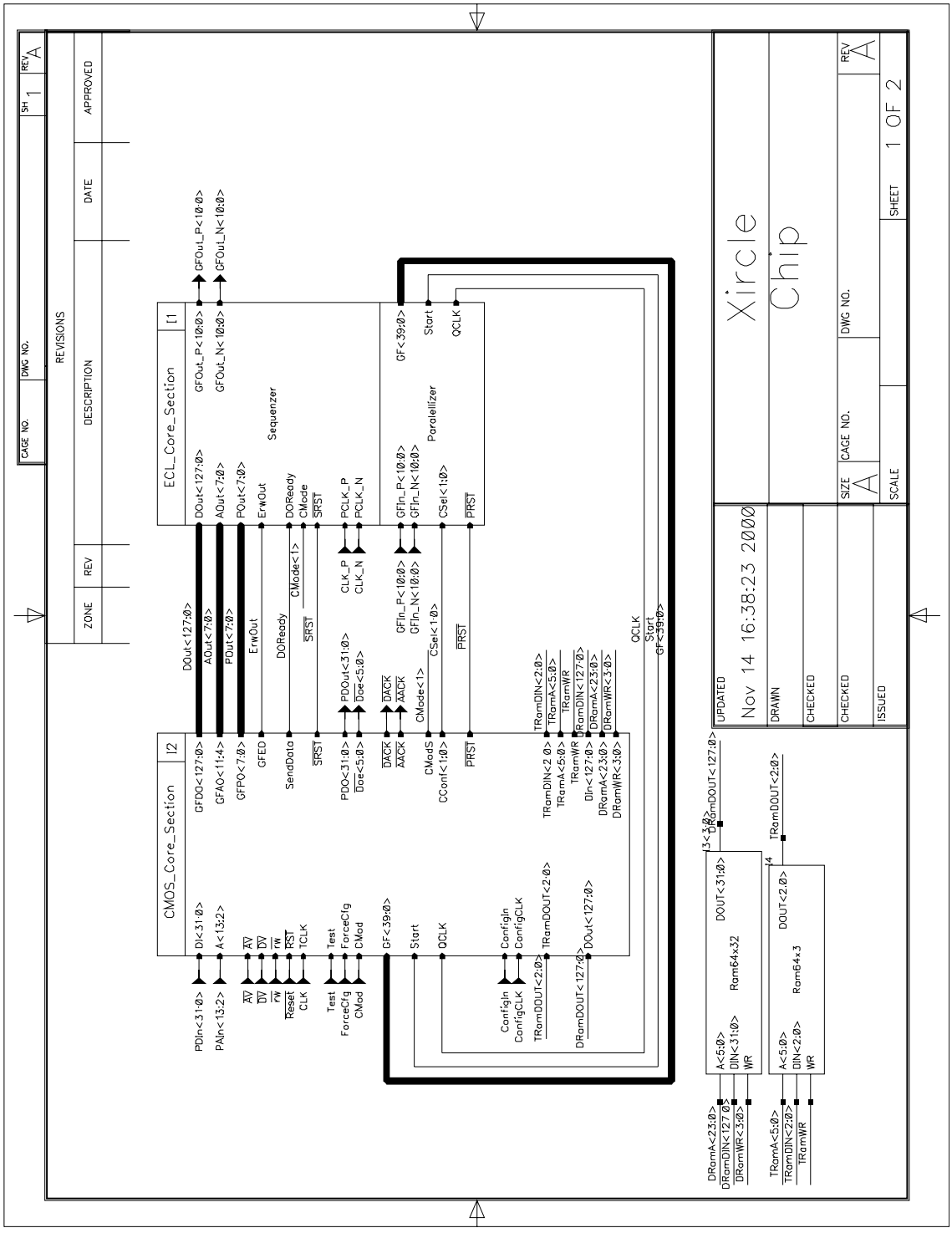

Abbildung C.2: Chip@Scheet1

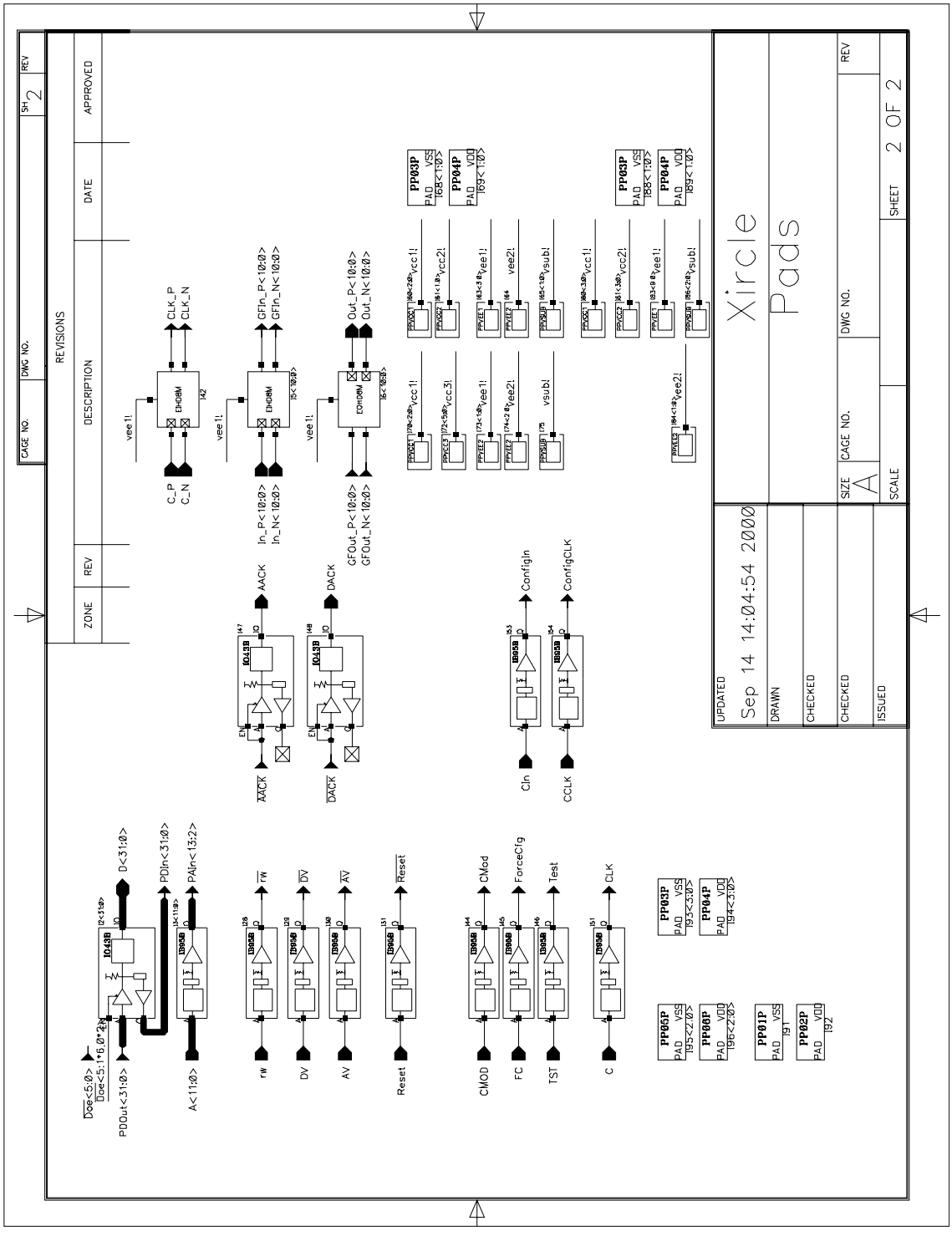

Abbildung C.3: Chip@Scheet2

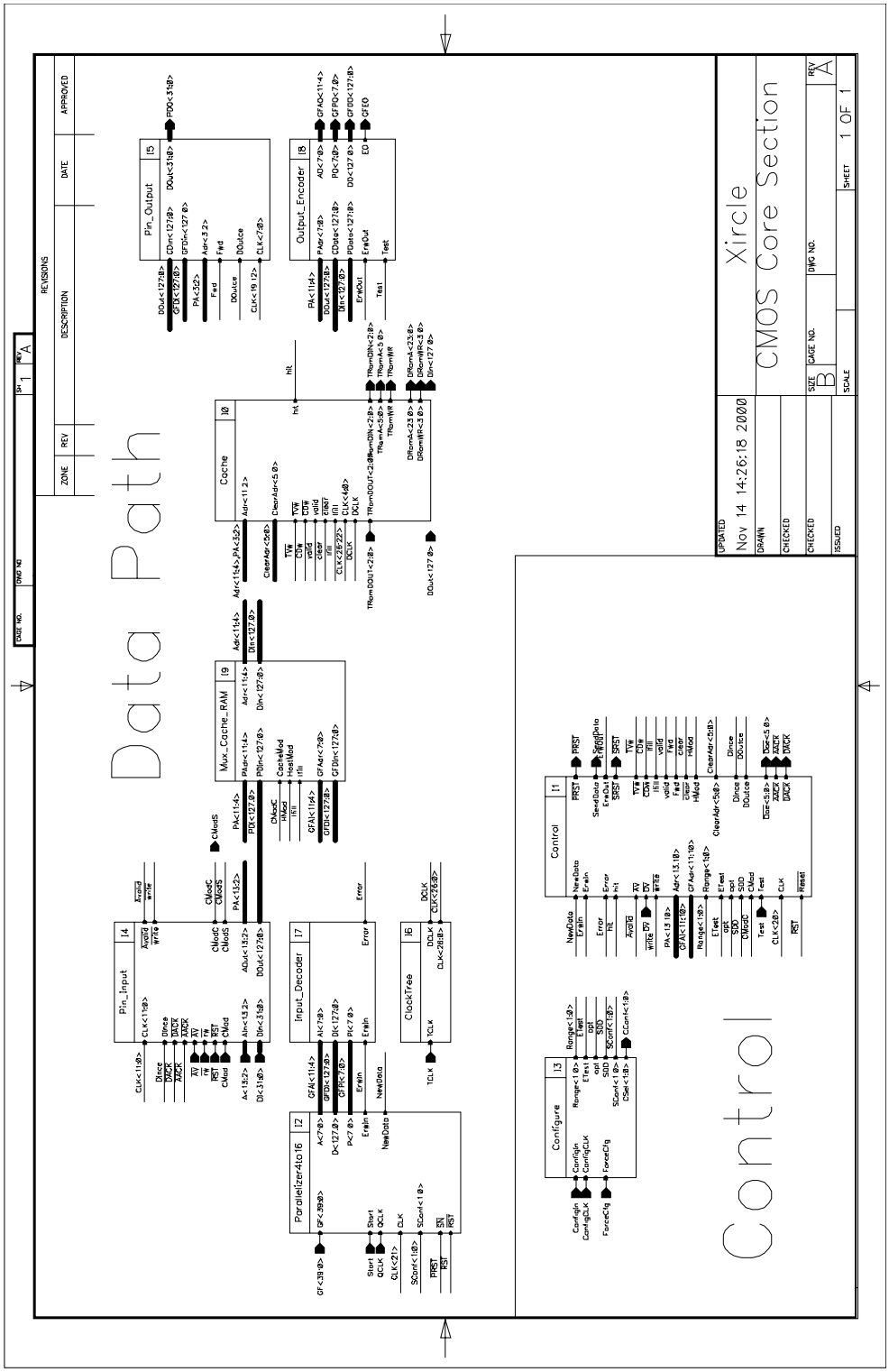

Abbildung C.4: CMOS_Core_Section

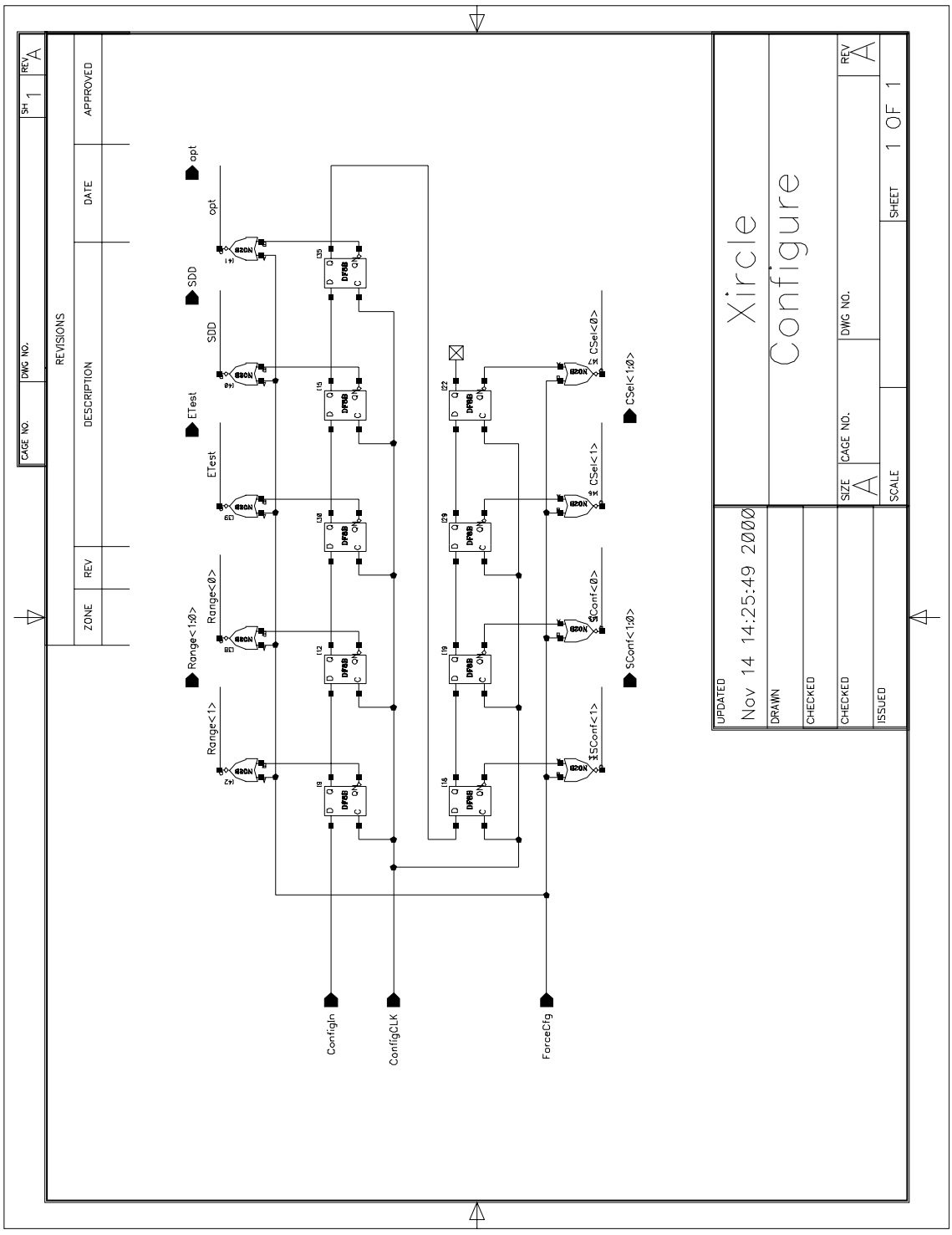

Abbildung C.5: Configure

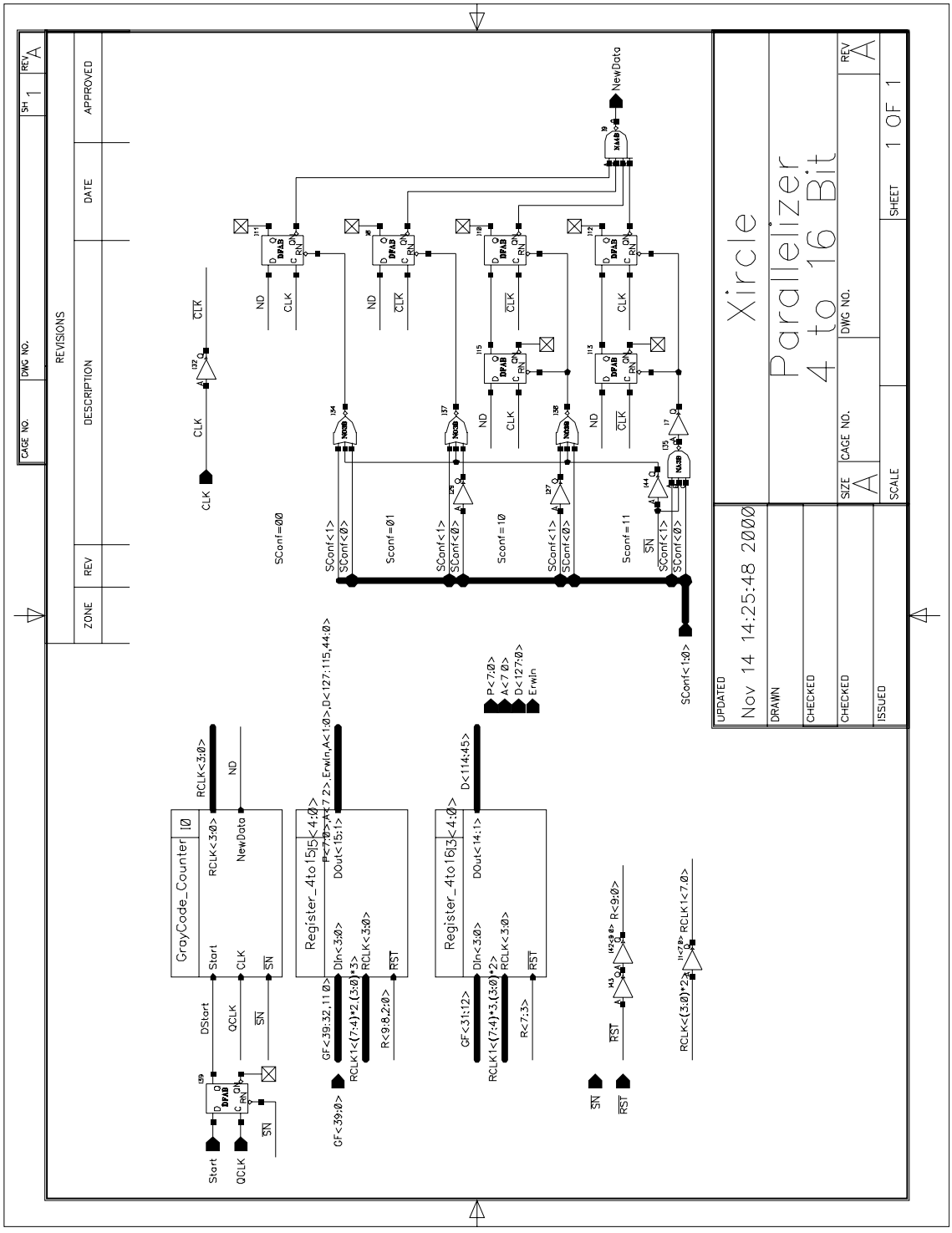

Abbildung C.6: Parallelizer4to16

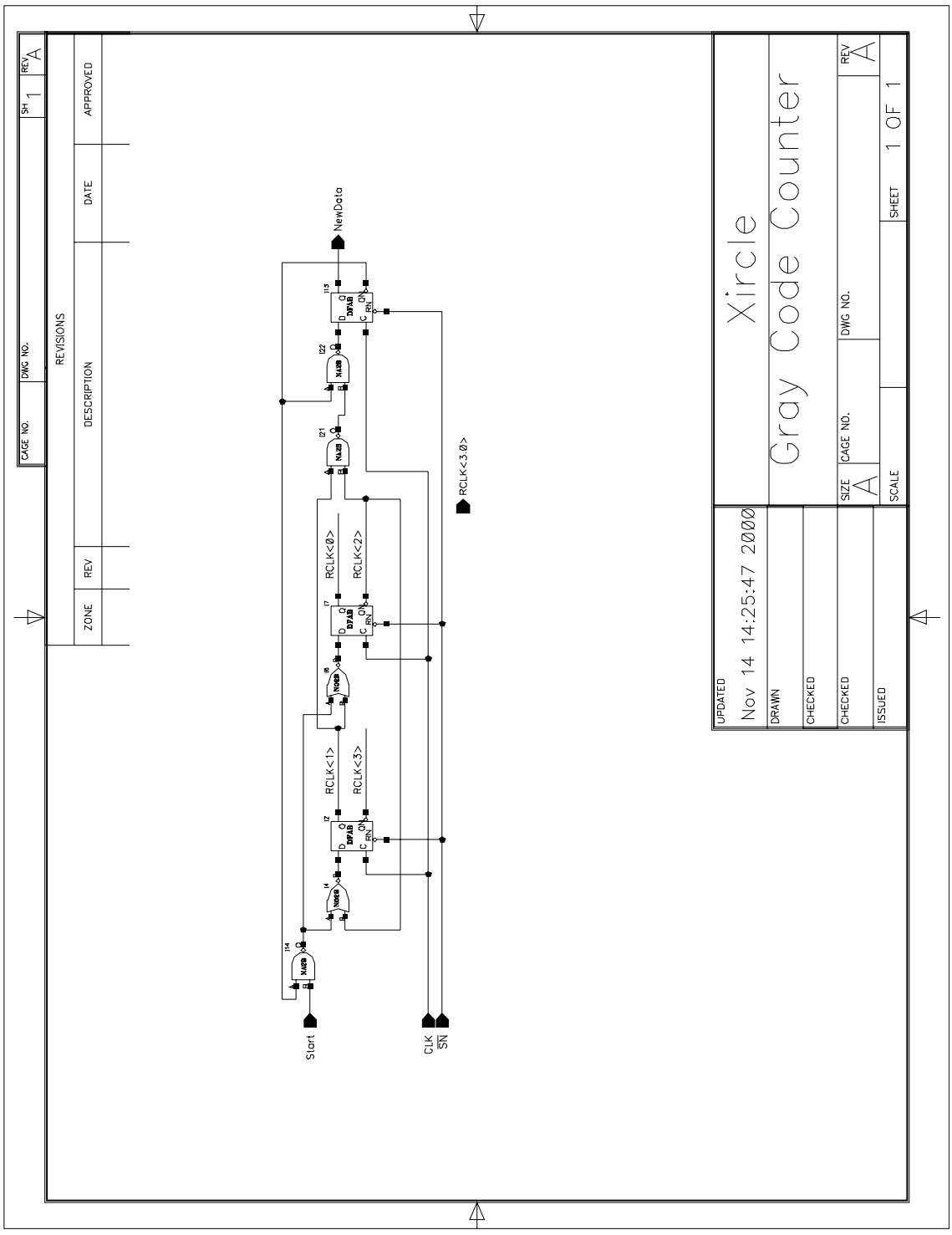

Abbildung C.7: GrayCode_Counter

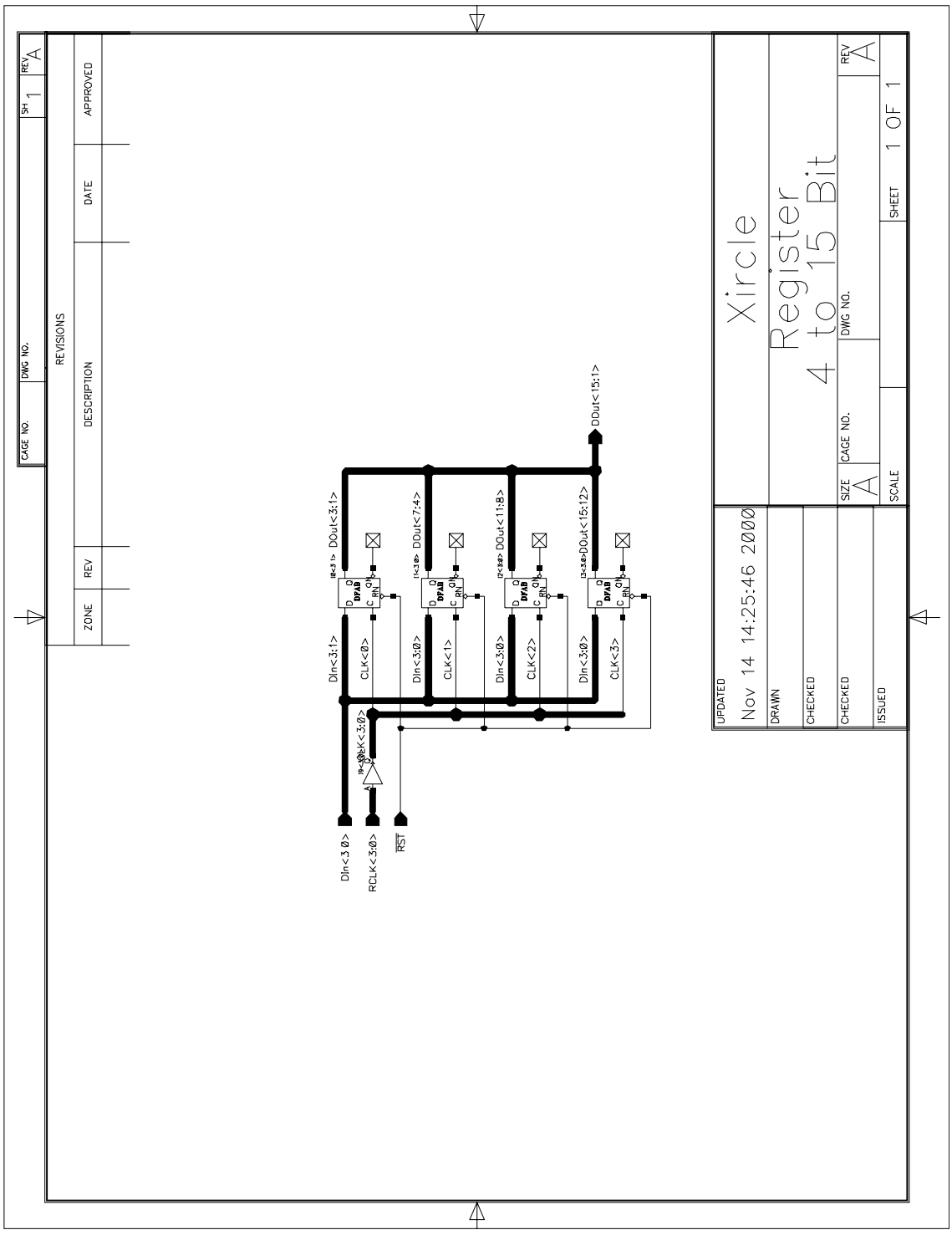

Abbildung C.8: Register_4to15

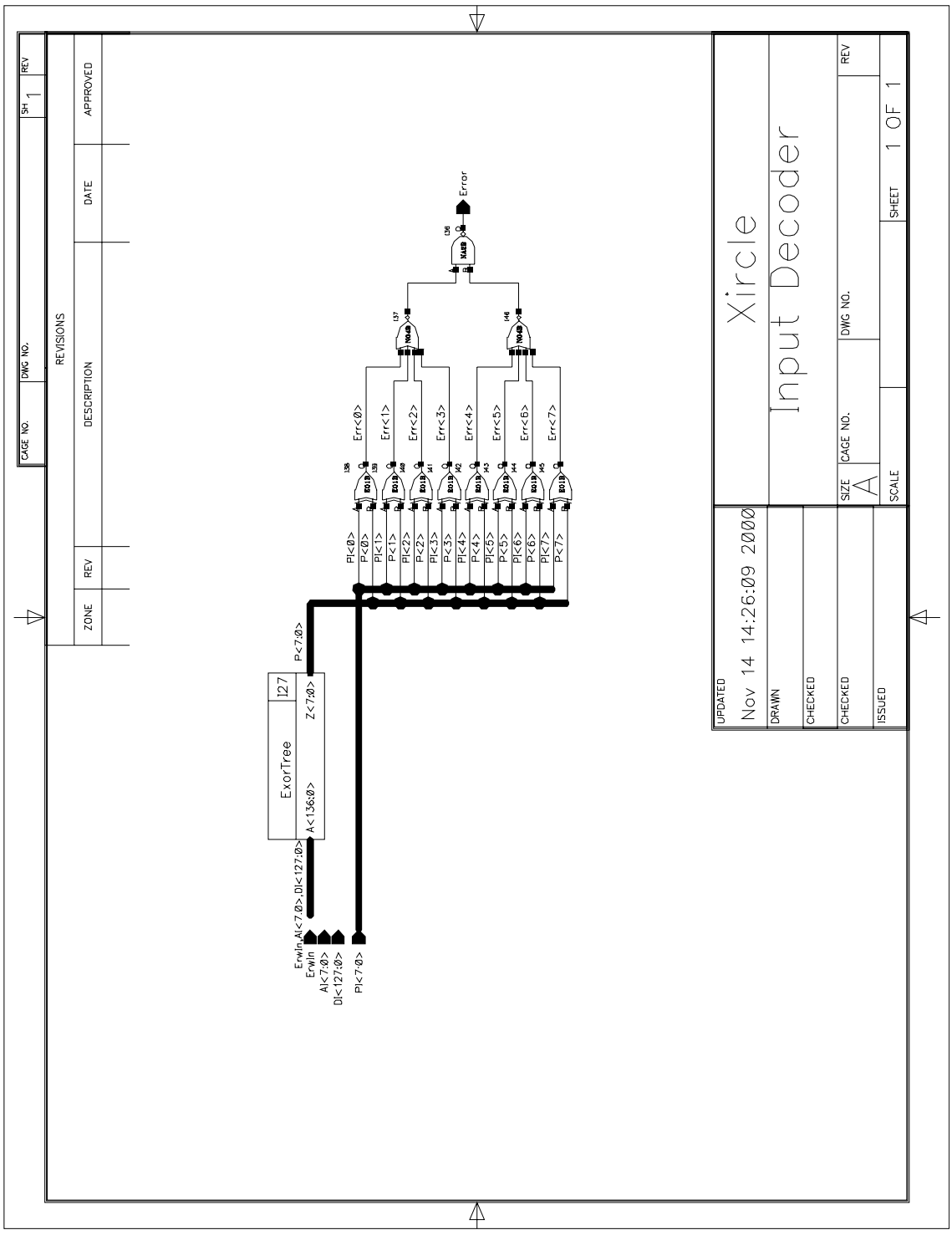

Abbildung C.9: Input_Decoder

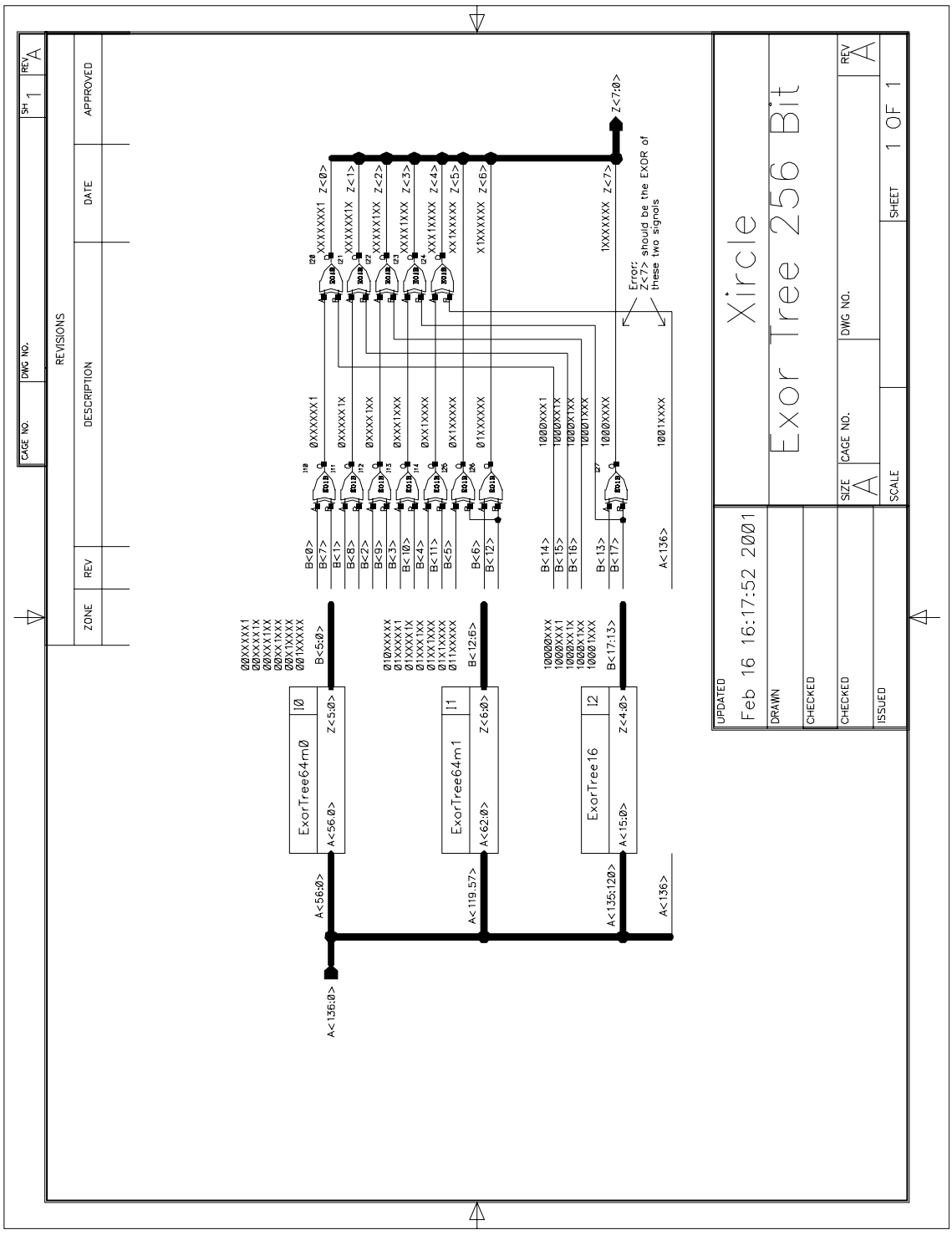

Abbildung C.10: ExorTree

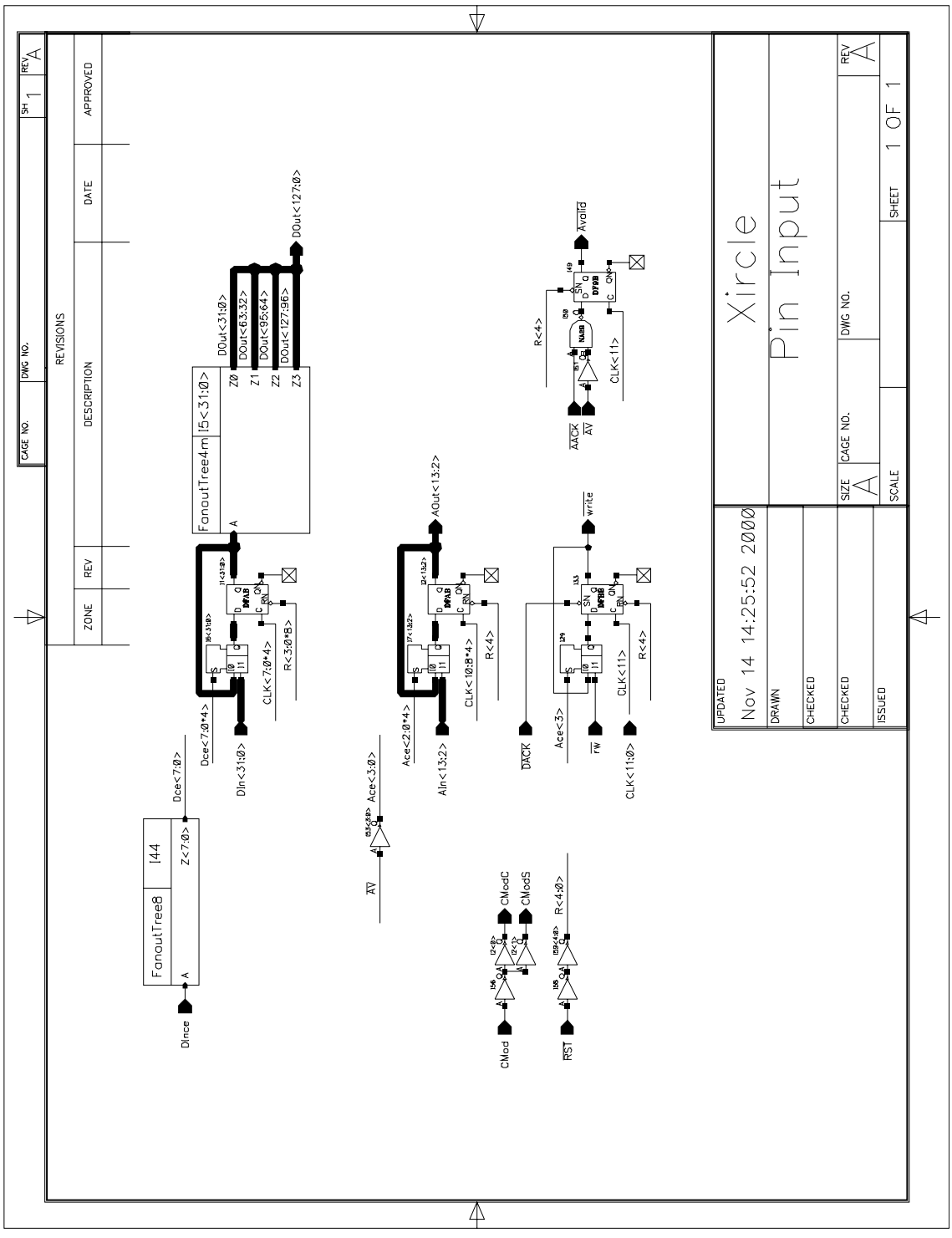

Abbildung C.11: Pin_Input

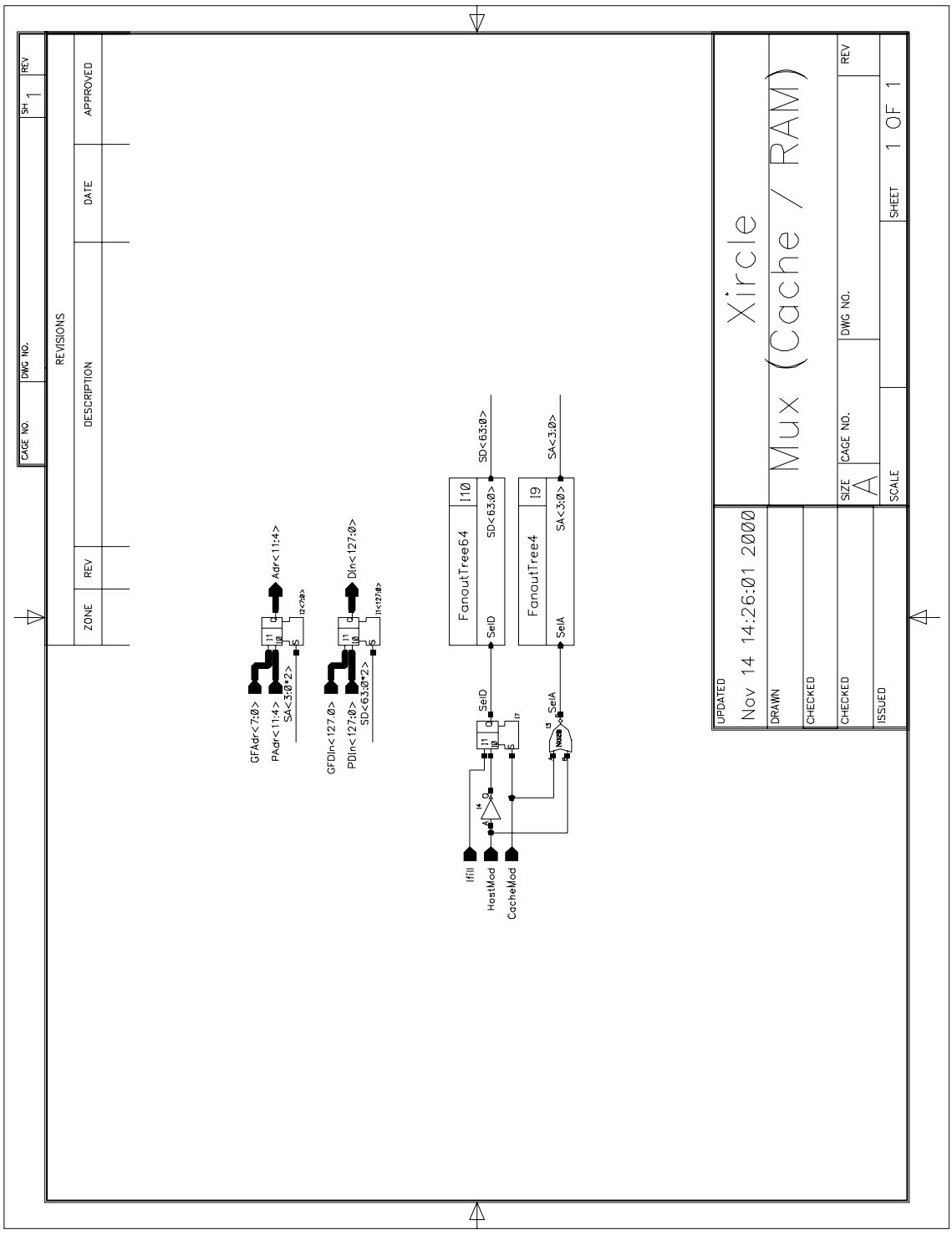

Abbildung C.12: Mux_Cache_RAM

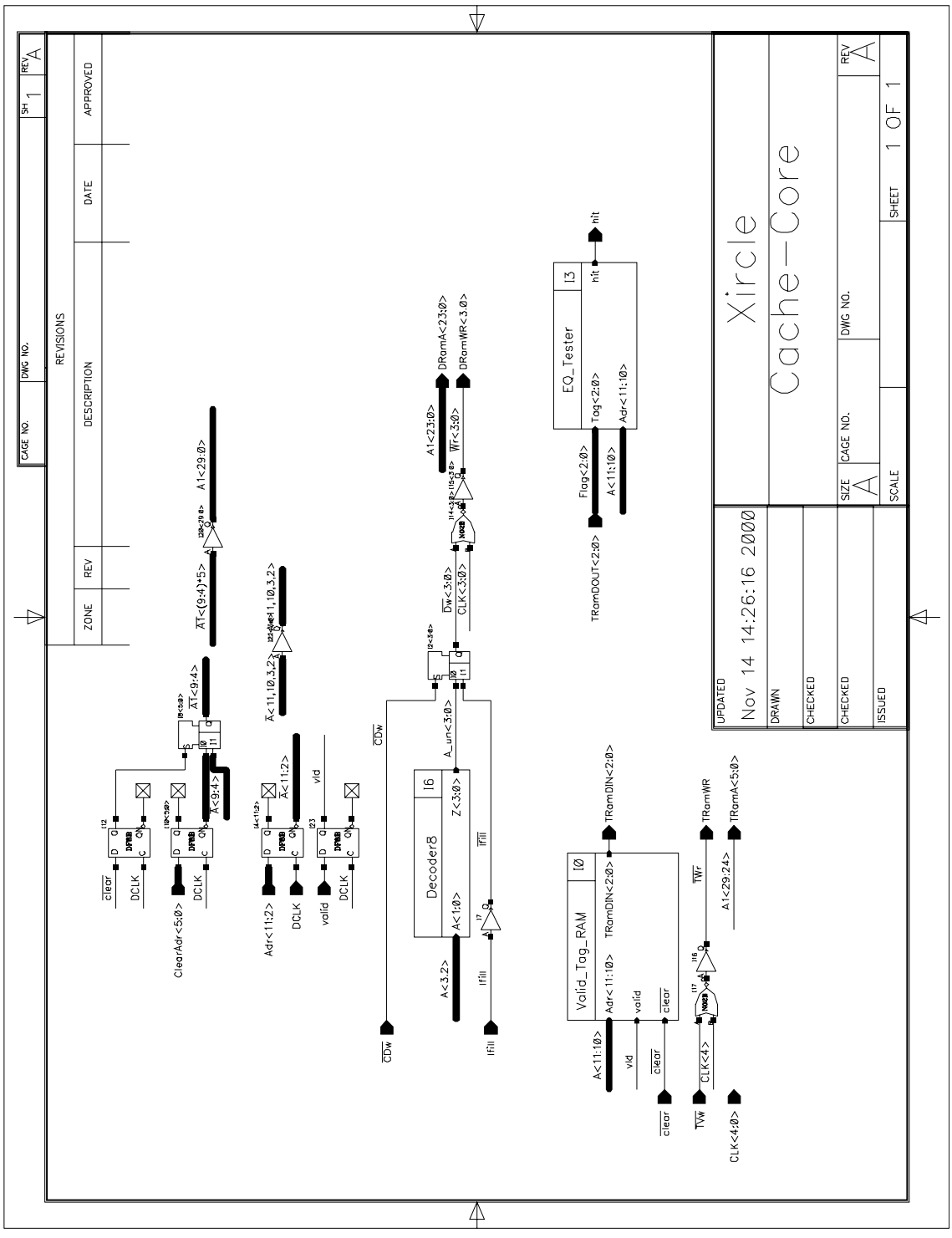

Abbildung C.13: Cache

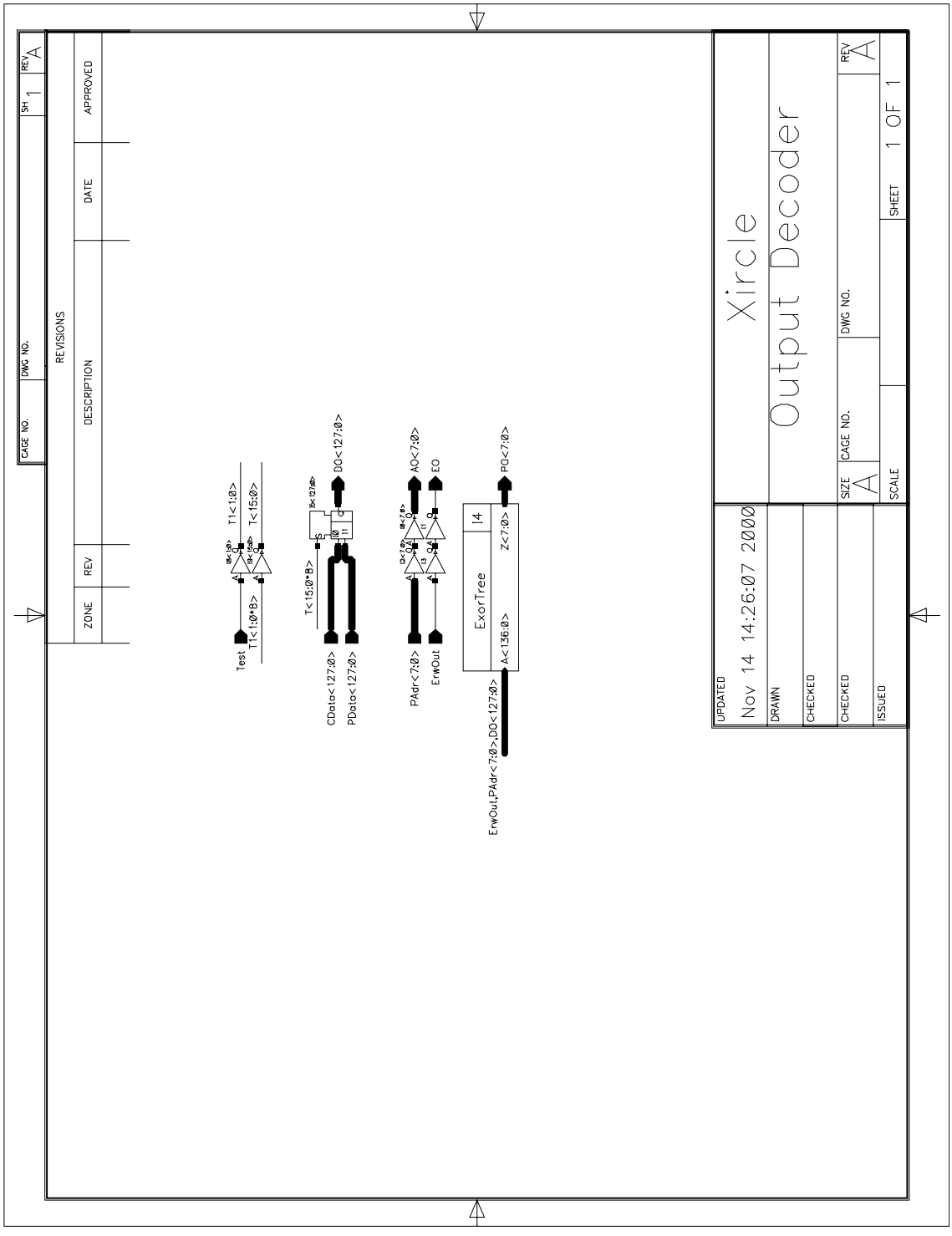

Abbildung C.14: Output_Encoder

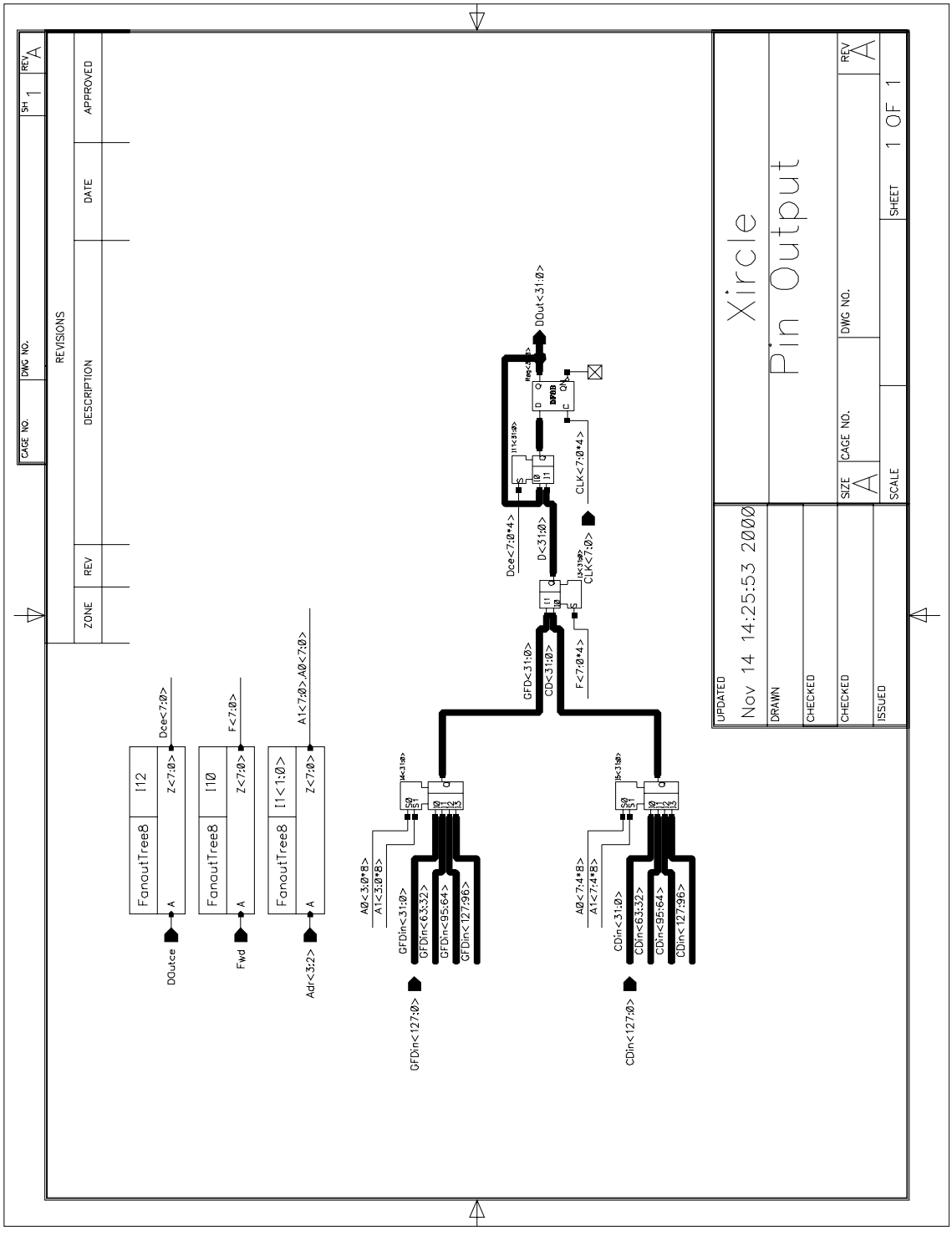

Abbildung C.15: Pin_Output

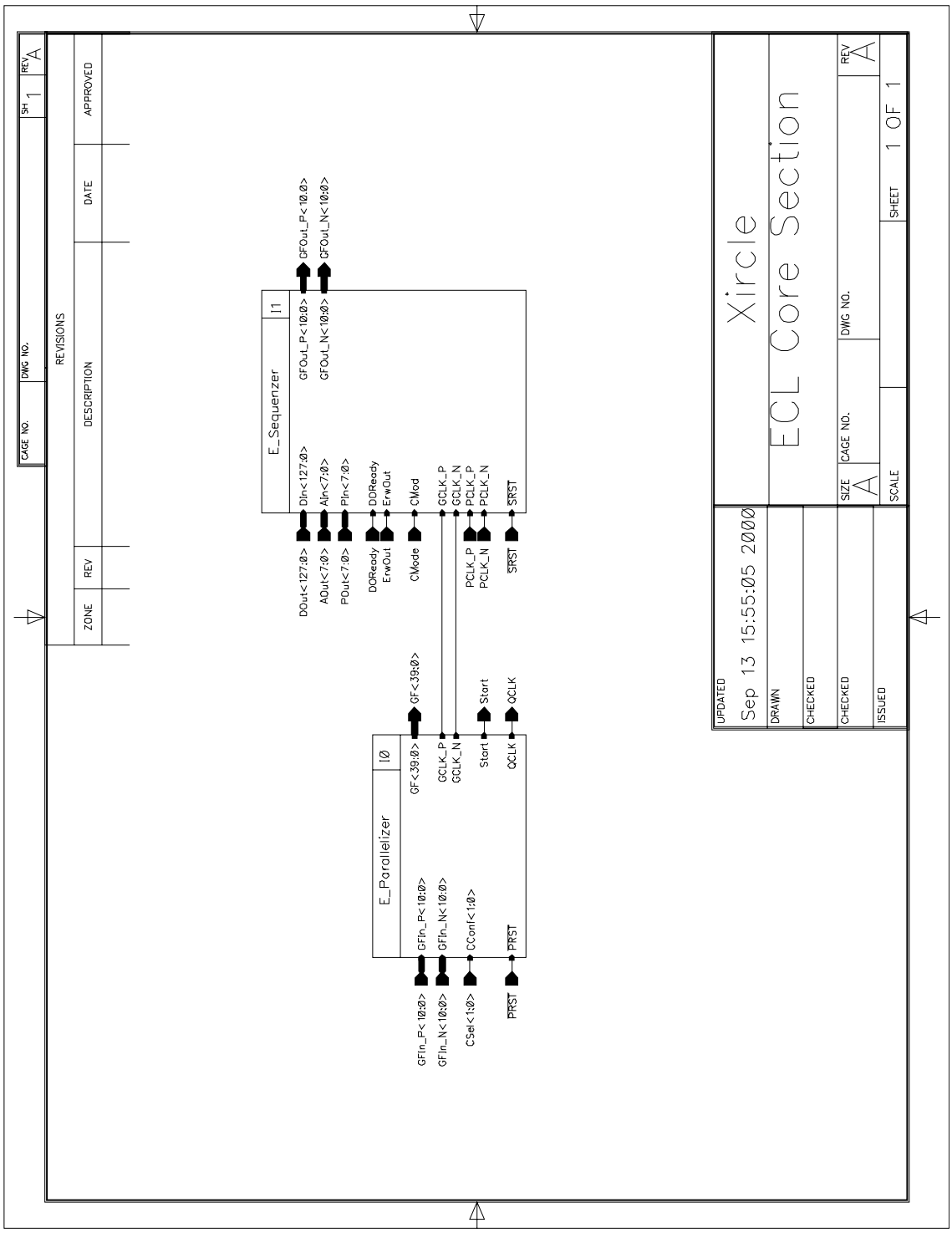

Abbildung C.16: ECL_Core_Section

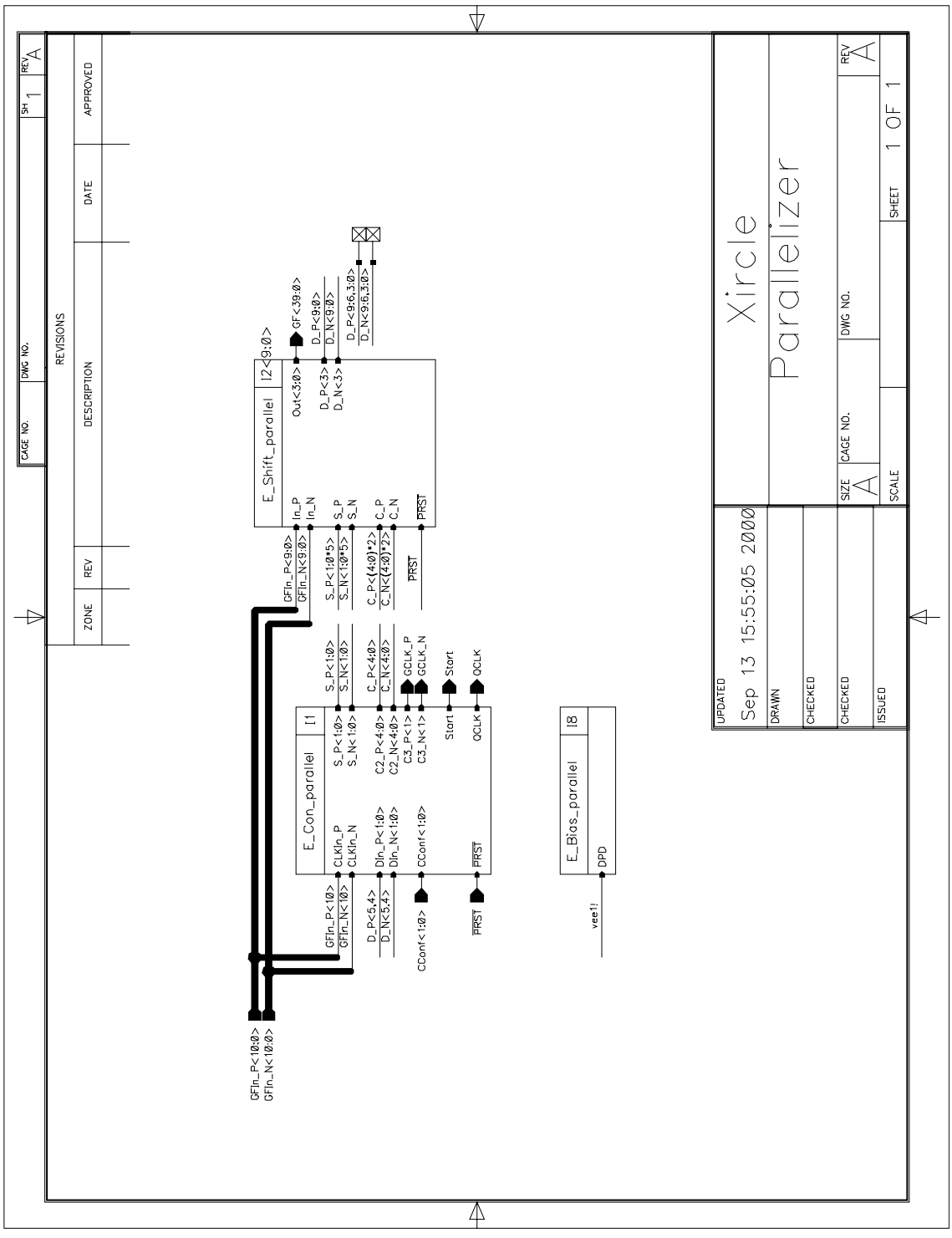

Abbildung C.17: E_Parallelizer

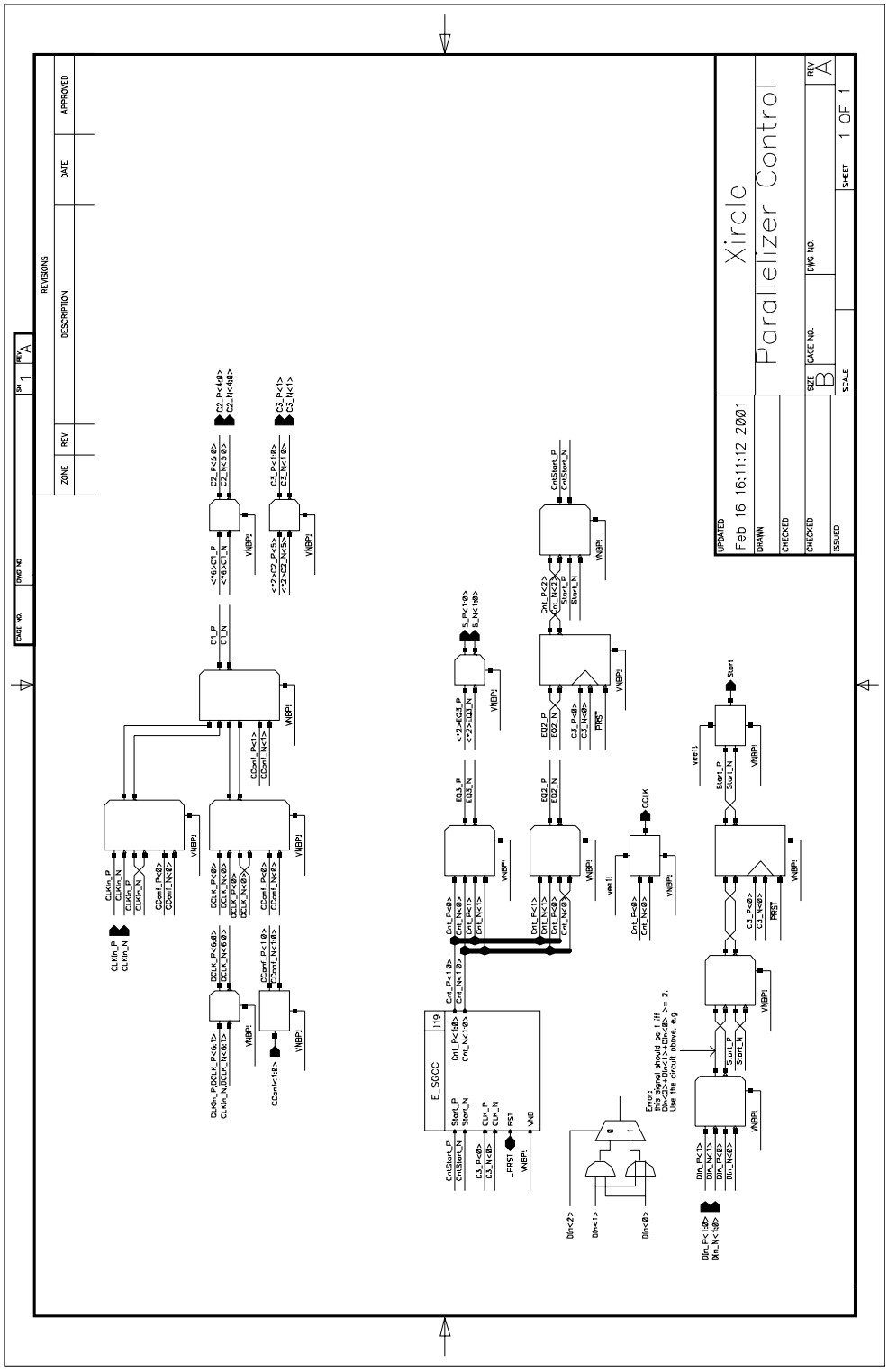

Abbildung C.18: E_Con_parallel

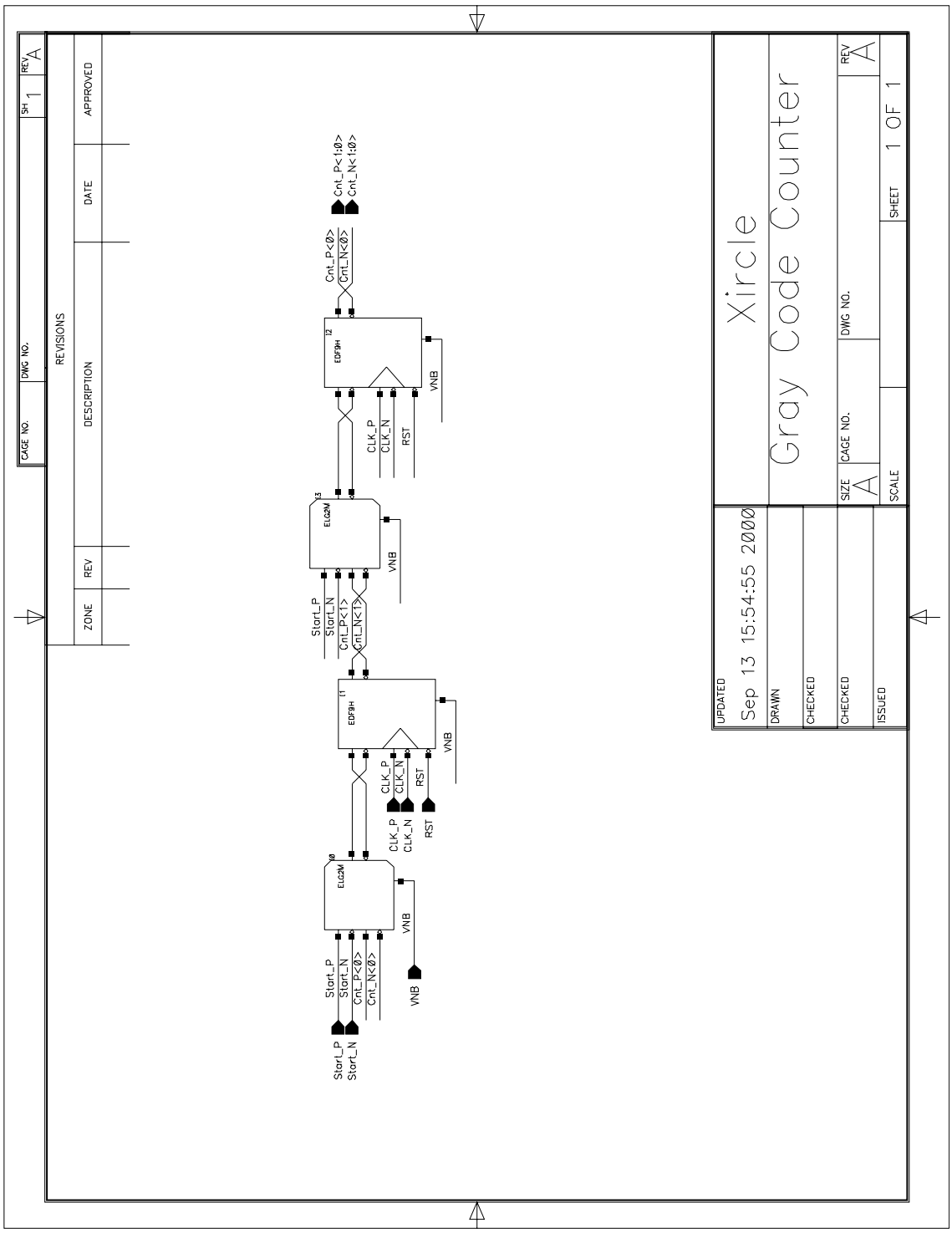

Abbildung C.19: E_SGCC

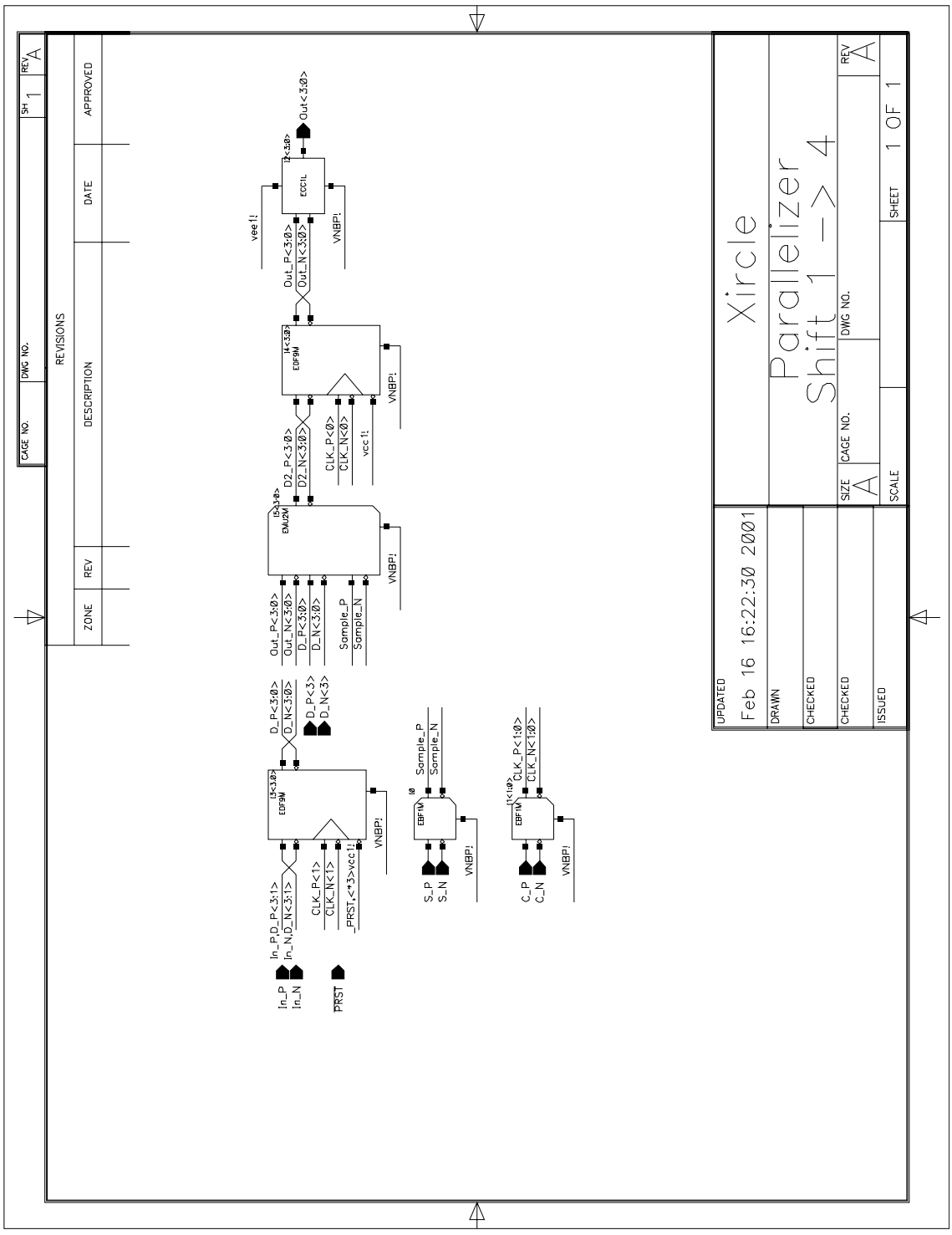

Abbildung C.20: E_Shift_parallel

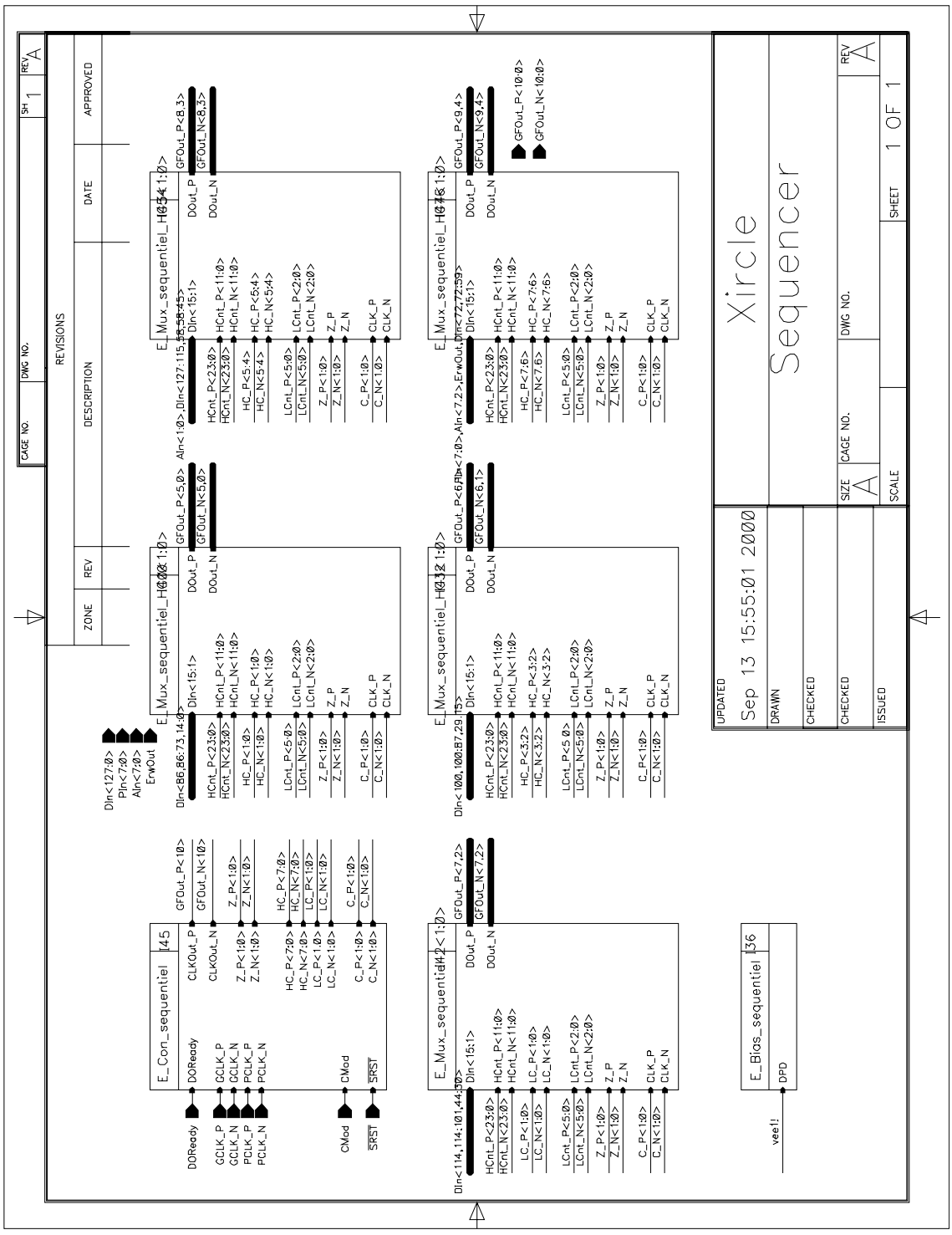

Abbildung C.21: E_Sequencer

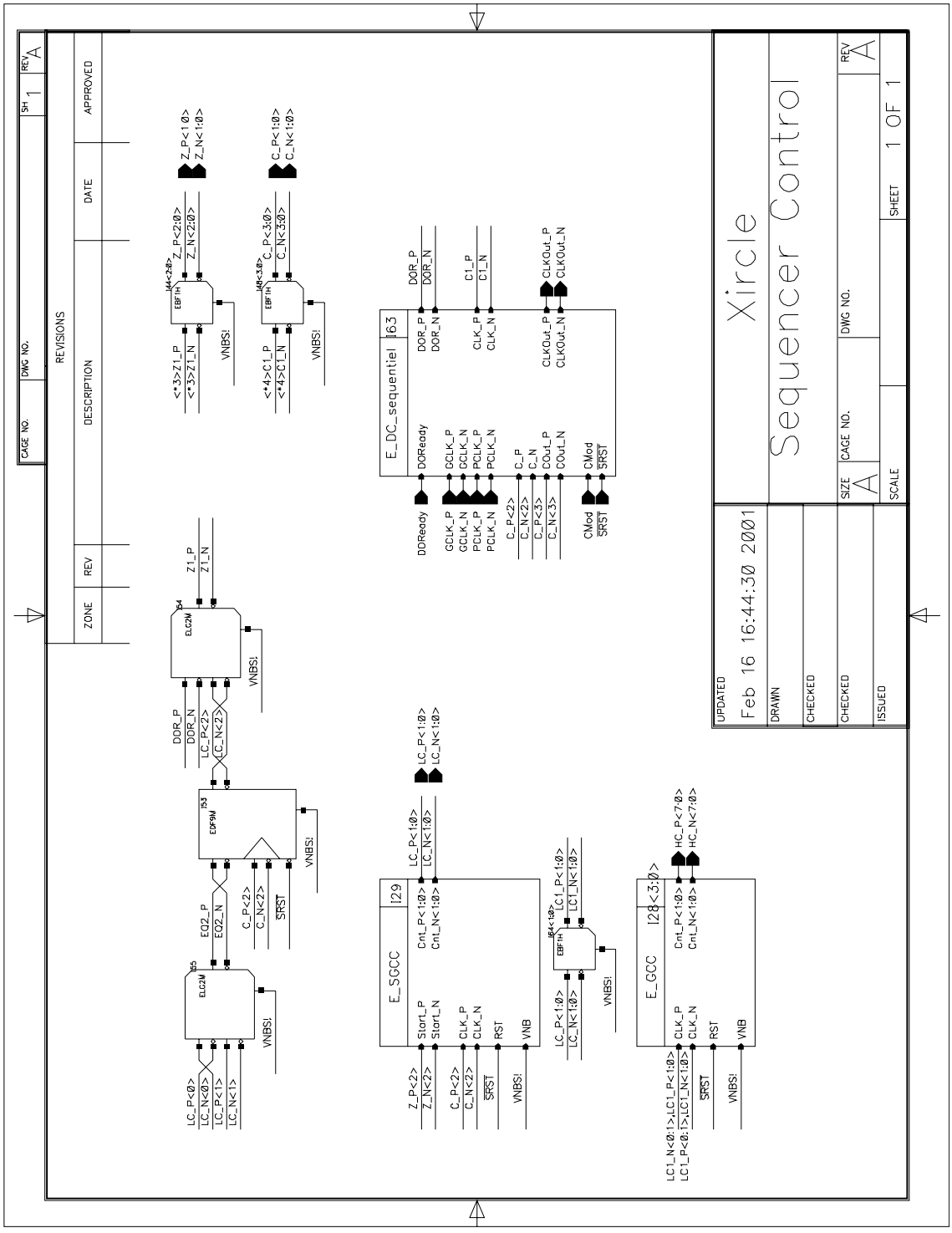

Abbildung C.22: E_Con_sequentiel

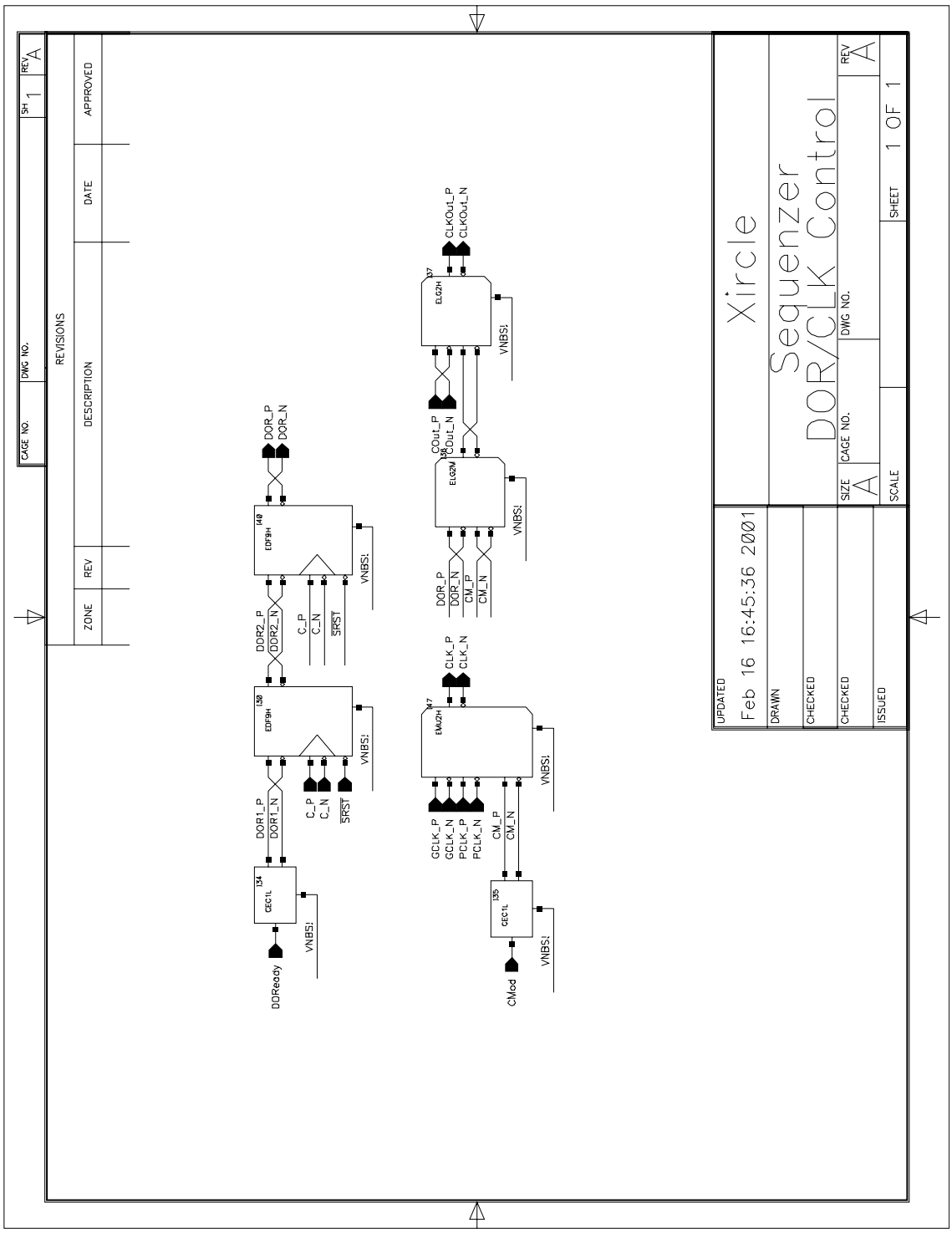

Abbildung C.23: E_DC_sequentiel

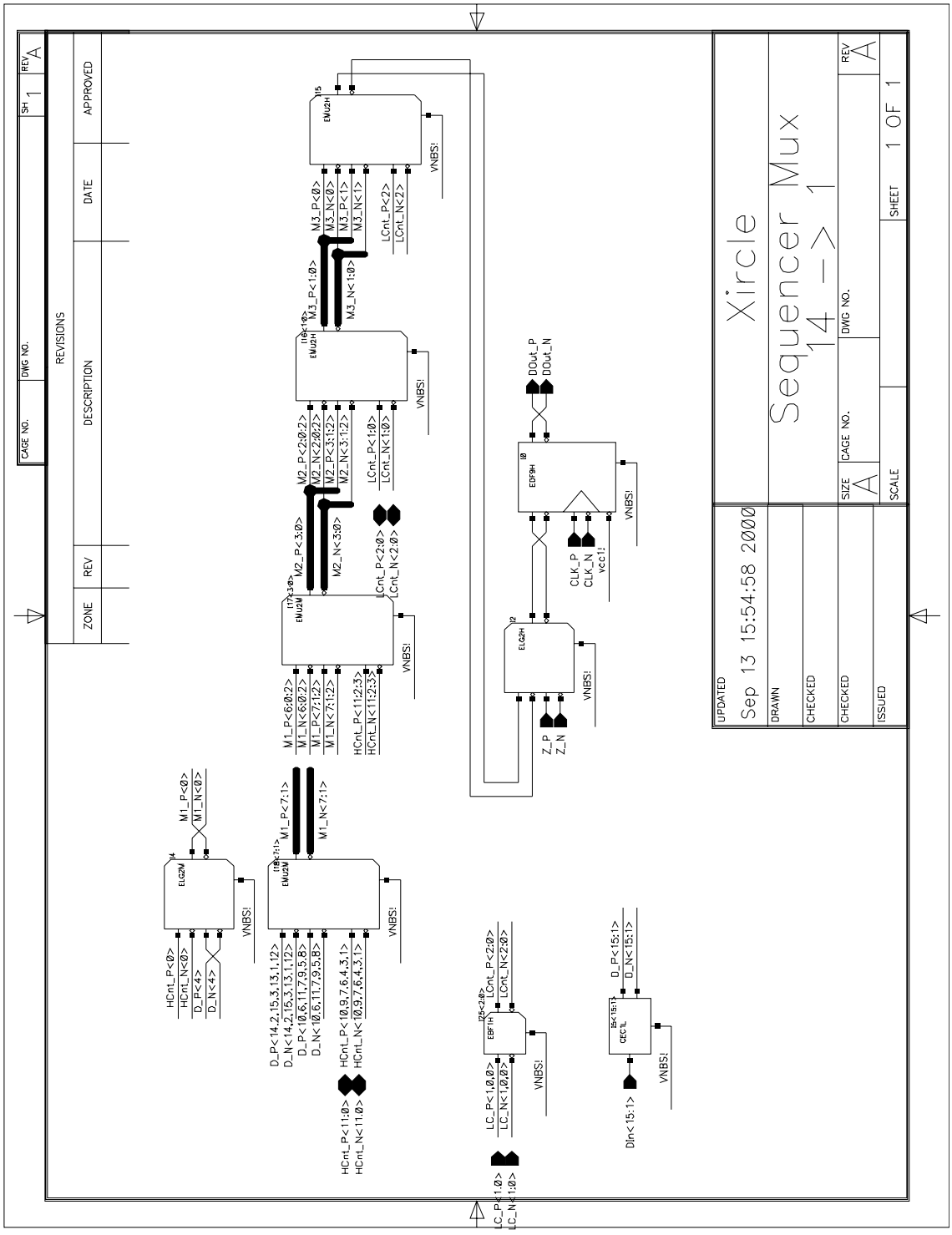

Abbildung C.24: E_Mux_sequentiel

Anhang D

Kontrolle der Cache/RAM-Logik

```
// States
`define C_Start           4'h0
`define C_Wait4DV         4'h1
`define C_FillReq         4'h2
`define C_Wait4Fill       4'h3
`define C_FillTest        4'h4
`define C_LineFillWrite   4'h5
`define C_CacheUpdate     4'h6
`define C_WriteReq        4'h7
`define C_Wait4Write      4'h8
`define Clear             4'h9
`define R_Start           4'ha
`define R_Wait4DV         4'hb
`define R_CacheAccess     4'hc
`define R_SendError       4'hd
`define R_CacheRead       4'he
`define R_CacheWrite      4'hf

// Control Signals
`define SHMod           13'h1000
`define SPRST           13'h0800
`define SSendData       13'h0400
`define SErwOut         13'h0200
`define SSRST           13'h0100
`define STVw            13'h0080
`define SCDw            13'h0040
`define Slfill          13'h0020
`define Svalid          13'h0010
`define SFwd            13'h0008
`define Sclear          13'h0004
`define SDoe            13'h0002
`define SDACK           13'h0001

// Active Control Signals
`define SC_Start          0
`define SC_Wait4DV        0
`define SC_FinishRead     `SDoe | `SDACK
`define SC_FillReq        `STVw | `SSendData
`define SC_Wait4Fill      `SFwd
`define SC_FillTest       `SFwd
`define SC_LineFillRead   `STVw | `Svalid | `Slfill | `SDoe | `SDACK
```

```
‘define SC_LineFillWrite  ‘STVw | ‘Svalid | ‘Slfill
‘define SC_CacheUpdate    ‘SCDw | ‘SErwOut
‘define SC_WriteReq       ‘SErwOut | ‘SSendData
‘define SC_Wait4Write     0
‘define SC_FinishWrite    ‘SDACK
‘define SClear            ‘Sclear | ‘SPRST | ‘SSRST | ‘STVw | ‘Slfill
‘define SR_Start          ‘SHMod
‘define SR_Wait4DV        ‘SHMod
‘define SR_HostRead       ‘SDoe | ‘SDACK
‘define SR_HostWrite      ‘SCDw | ‘SDACK
‘define SR_CacheAccess    ‘SPRST
‘define SR_SendError      ‘SErwOut | ‘SSendData
‘define SR_CacheRead      ‘SSendData
‘define SR_CacheWrite     ‘SSendData | ‘Slfill

// ---------------------------------------------------------------------------

module Control (_PRST, SendData, ErwOut, _SRST, _TVw, _CDw, lfill, valid, Fwd,
                _clear, DInce, DOutce, _Doe_, _AACK, _DACK, HMod, CLK, CMod,
                Test, GFAdr, Adr, ClearAdr, Range, ETest, opt, SDD, hit, Error,
                ErwIn, NewData, _write, _AV, _DV, _Reset);

output DInce;
output DOutce;
output [5:0] ClearAdr;
output HMod;
output _PRST;           // active low
output SendData;
output ErwOut;
output _SRST;           // active low
output _CDw;            // active low
output _TVw;            // active low
output lfill;
output valid;
output Fwd;
output _clear;          // active low
output [5:0] _Doe_;     // active low
output _AACK;           // active low
output _DACK;           // active low

input CLK;
input CMod;
input Test;
input ETest;
input opt;
input SDD;
input [11:10] GFAdr;
input [13:10] Adr;
input [1:0] Range;
input hit;
input Error;
input ErwIn;
input NewData;
input _write;           // active low
input _AV;              // active low
input _DV;              // active low
input _Reset;           // avtive low
```

```
reg [3:0] State;
reg [12:0] Signals;
reg [5:0] Adr_Counter;
reg Wait;

wire [5:0] ClearAdr = Adr_Counter[5:0];
wire HAcc           = Adr[13] & (Adr[12] == CMod) & ((Adr[11:10] == Range[1:0])
                      | CMod);
wire CAcc           = GFAdr[11:10] == Range[1:0];
wire CacheDce       = Test ? ((State == `C_Start) & ~_AV * HAcc) : (((State ==
                      `C_Start) & ~_AV * HAcc * hit) | (State == `C_Wait4DV) |
                      ((opt ? ((State == `C_Wait4Fill) & NewData & ~ErwIn) :
                      (State == `C_FillTest)) & ~(Error & ETest))) & ~_DV;
wire RAMDce         = ((State == `R_Start) & ~(NewData & CAcc) & (~_AV * HAcc)
                      * ~_DV) | ((State == `R_Wait4DV) & ~_DV);
wire DInce          = (CMod ? CacheDce : RAMDce) & (~_write | (CMod & Test));
wire DOutce         = (CMod ? CacheDce : RAMDce) & (_write | (~CMod & Test));
wire _AACK          = ~((CMod ? (State[3:0] == `C_Start) : (State == `R_Start))
                      & HAcc & ~_AV);
wire ECheck         = Signals[13];
wire HMod           = Signals[12];
wire _PRST          = ~Signals[11];
wire SendData       = Signals[10];
wire ErwOut         = Signals[9];
wire _SRST          = ~Signals[8];
wire _TVw           = ~Signals[7];
wire _CDw           = ~Signals[6];
wire lfill          = Signals[5];
wire valid          = Signals[4];
wire Fwd            = Signals[3];
wire _clear         = ~Signals[2];
wire [5:0] _Doe_    = {6{~Signals[1]}};
wire _DACK          = ~Signals[0];

always @ (posedge CLK)
begin

if (~_Reset)
  begin
    State <= `Clear;
    Signals <= `SClear;
    Adr_Counter[5:0] <= 6'b000000;
    Wait <= 0;
  end
else
  if (Test)
    begin
      case (State)
        `Clear:
          begin
            if (CMod)
              begin
                State <= `C_Start;
                Signals <= `SC_Start | `SPRST;
              end
            else
```

```
        begin
          State <= `R_Start;
          Signals <= `SR_Start | `SSRST;
        end
    end
  `C_Start:
    begin
      if (!_AV & HAcc)
        begin
          if (_DV)
            begin
              State <= `C_Wait4DV;
              Signals <= `SC_Wait4DV | `SPRST;
            end
          else
            begin
              State <= `C_WriteReq;
              Signals <= `SC_WriteReq | `SPRST;
              Wait <= 1;
            end
        end
      else
        begin
          State <= `C_Start;
          Signals <= `SC_Start | `SPRST;
        end
    end
  `C_Wait4DV:
    begin
      if (~_DV)
        begin
          State <= `C_WriteReq;
          Signals <= `SC_WriteReq | `SPRST;
          Wait <= 1;
        end
      else
        begin
          State <= `C_Wait4DV;
          Signals <= `SC_Wait4DV | `SPRST;
        end
    end
  `C_WriteReq :
    begin
      if (Wait & SDD)
        begin
          State <= `C_WriteReq;
          Signals <= `SC_WriteReq | `SPRST;
          Wait <= 0;
        end
      else
        begin
          State <= `C_Wait4Write;
          Signals <= `SC_Wait4Write | `SSRST | `SPRST;
        end
    end
  `C_Wait4Write:
    begin
```

```
            State <= `C_Start;
            Signals <= `SC_Start | `SPRST | `SC_FinishWrite;
          end
        `R_Start:
          begin
            if (~_AV & HAcc)
              begin
                if (~_DV)
                  begin
                    State <= `R_Start;
                    Signals <= `SR_Start | `SR_HostRead | `SSRST | `SFwd;
                  end
                else
                  begin
                    State <= `R_Wait4DV;
                    Signals <= `SR_Wait4DV | `SSRST | `SFwd;
                  end
              end
            else
              begin
                State <= `R_Start;
                Signals <= `SR_Start | `SSRST | `SFwd;
              end
            if (NewData)
              begin
                Signals <= Signals | `SPRST;
              end
          end
`R_Wait4DV:
          begin
            if (~_DV)
              begin
                State <= `R_Start;
                Signals <= `SR_Start | `SR_HostRead | `SSRST | `SFwd;
              end
            else
              begin
                State <= `R_Wait4DV;
                Signals <= `SR_Wait4DV | `SSRST | `SFwd;
              end
            if (NewData)
              begin
                Signals <= Signals | `SPRST;
              end
          end
        default:
          begin
            State <= `Clear;
            Signals <= `SClear;
          end
      endcase
    end
  else
    begin
      case (State)
        `Clear:
          begin
```

```
      if (CMod)
        begin
        if (Adr_Counter[5:0] == 6'b111111)
          begin
            State <= `C_Start;
            Signals <= `SC_Start;
          end
        else
          begin
            State <= `Clear;
            Signals <= `SClear;
            Adr_Counter <= Adr_Counter + 1;
          end
        end
      else
        begin
          State <= `R_Start;
          Signals <= `SR_Start;
        end
    end
  `C_Start:
     begin
       if (!_AV * HAcc)
         begin
           if (~hit)
             begin
               State <= `C_FillReq;
               Signals <= `SC_FillReq;
               Wait <= 1;
             end
           else if (_DV)
             begin
               State <= `C_Wait4DV;
               Signals <= `SC_Wait4DV;
             end
           else if (_write)
             begin
               State <= `C_Start;
               Signals <= `SC_Start | `SC_FinishRead;
             end
           else
             begin
               State <= `C_CacheUpdate;
               Signals <= `SC_CacheUpdate;
             end
         end
       else
         begin
           State <= `C_Start;
           Signals <= `SC_Start;
         end
       end
  `C_Wait4DV:
    begin
      if (~_DV)
        begin
          if (_write)
```

```
            begin
              State <= `C_Start;
              Signals <= `SC_Start | `SC_FinishRead;
            end
          else
            begin
              State <= `C_CacheUpdate;
              Signals <= `SC_CacheUpdate;
            end
        end
      else
        begin
          State <= `C_Wait4DV;
          Signals <= `SC_Wait4DV;
        end
    end
  `C_FillReq:
    begin
      if (Wait & SDD)
        begin
          State <= `C_FillReq;
          Signals <= `SC_FillReq;
          Wait <= 0;
        end
      else
        begin
          State <= `C_Wait4Fill;
          Signals <= `SC_Wait4Fill | `SSRST;
        end
    end
  `C_Wait4Fill:
    begin
      if (NewData)
        begin
          if (ErwIn | (opt & Error & ETest))
            begin
              State <= `C_FillReq;
              Signals <= `SC_FillReq | `SPRST;
              Wait <= 1;
            end
          else if (~opt)
            begin
              State <= `C_FillTest;
              Signals <= `SC_FillTest | `SPRST;
            end
          else if (_DV)
            begin
              State <= `C_Wait4Fill;
              Signals <= `SC_Wait4Fill;
            end
          else if (_write)
            begin
              State <= `C_Start;
              Signals <= `SC_Start | `SPRST | `SC_LineFillRead;
            end
          else
            begin
```

```
            State <= `C_LineFillWrite;
            Signals <= `SC_LineFillWrite | `SPRST;
          end
      end
    else
      begin
        State <= `C_Wait4Fill;
        Signals <= `SC_Wait4Fill;
      end
  end
`C_FillTest:
  begin
    if (Error & ETest)
      begin
        State <= `C_FillReq;
        Signals <= `SC_FillReq;
        Wait <= 1;
      end
    else if (_DV)
      begin
        State <= `C_FillTest;
        Signals <= `SC_FillTest;
      end
    else if (_write)
      begin
        State <= `C_Start;
        Signals <= `SC_Start | `SC_LineFillRead;
      end
    else
      begin
        State <= `C_LineFillWrite;
        Signals <= `SC_LineFillWrite;
      end
  end
`C_LineFillWrite:
  begin
    State <= `C_CacheUpdate;
    Signals <= `SC_CacheUpdate;
  end
`C_CacheUpdate:
  begin
    State <= `C_WriteReq;
    Signals <= `SC_WriteReq;
    Wait <= 1;
  end
`C_WriteReq :
  begin
    if (Wait & SDD)
      begin
        State <= `C_WriteReq;
        Signals <= `SC_WriteReq;
        Wait <= 0;
      end
    else
      begin
        State <= `C_Wait4Write;
        Signals <= `SC_Wait4Write | `SSRST;
```

```
      end
  end
`C_Wait4Write:
  begin
    if (NewData)
      begin
     if (ErwIn)
       begin
         State <= `C_WriteReq;
         Signals <= `SC_WriteReq  | `SPRST;
         Wait <= 1;
       end
     else
       begin
         State <= `C_Start;
         Signals <= `SC_Start | `SPRST | `SC_FinishWrite;
       end
      end
    else
      begin
     State <= `C_Wait4Write;
     Signals <= `SC_Wait4Write;
      end
  end
`R_Start:
  begin
    if (NewData)
      begin
        if (CAcc)
          begin
            if (~opt)
              begin
           State <= `R_CacheAccess;
           Signals <= `SR_CacheAccess;
              end
            else
              begin
           if (Error & ETest)
             begin
               State <= `R_SendError;
               Signals <= `SR_SendError | `SPRST;
               Wait <= 1;
             end
           else if (ErwIn)
             begin
               State <= `R_CacheWrite;
               Signals <= `SR_CacheWrite | `SPRST;
               Wait <= 1;
             end
           else
             begin
               State <= `R_CacheRead;
               Signals <= `SR_CacheRead | `SPRST;
               Wait <= 1;
             end
              end
          end
```

```
          else
            begin
              State <= `R_Start;
              Signals <= `SR_Start | `SPRST;
            end
        end
      else if (~_AV & HAcc)
        begin
          if (_DV)
            begin
              State <= `R_Wait4DV;
              Signals <= `SR_Wait4DV;
            end
          else if (_write)
            begin
              State <= `R_Start;
              Signals <= `SR_Start | `SR_HostRead;
            end
          else
            begin
              State <= `R_Start;
              Signals <= `SR_Start | `SR_HostWrite;
            end
        end
      else
        begin
          State <= `R_Start;
          Signals <= `SR_Start;
        end
    end
  `R_Wait4DV:
    begin
      if (_DV)
        begin
          State <= `R_Wait4DV;
          Signals <= `SR_Wait4DV;
        end
      else if (_write)
        begin
          State <= `R_Start;
          Signals <= `SR_Start | `SR_HostRead;
        end
      else
        begin
          State <= `R_Start;
          Signals <= `SR_Start | `SR_HostWrite;
        end
    end
  `R_CacheAccess:
    begin
      if (Error & ETest)
        begin
          State <= `R_SendError;
          Signals <= `SR_SendError;
          Wait <= 1;
        end
      else if (ErwIn)
```

```
      begin
        State <= `R_CacheWrite;
        Signals <= `SR_CacheWrite;
        Wait <= 1;
      end
    else
      begin
        State <= `R_CacheRead;
        Signals <= `SR_CacheRead;
        Wait <= 1;
      end
  end
`R_SendError:
  begin
    if (Wait & SDD)
      begin
        State <= `R_SendError;
        Signals <= `SR_SendError;
        Wait <= 0;
      end
    else
      begin
        State <= `R_Start;
        Signals <= `SR_Start | `SSRST;
      end
  end
`R_CacheRead:
  begin
    if (Wait & SDD)
      begin
        State <= `R_CacheRead;
        Signals <= `SR_CacheRead;
        Wait <= 0;
      end
    else
      begin
        State <= `R_Start;
        Signals <= `SR_Start | `SSRST;
      end
  end
`R_CacheWrite:
  begin
    if (Wait & SDD)
      begin
        State <= `R_CacheWrite;
        Signals <= `SR_CacheWrite;
        Wait <= 0;
      end
    else
      begin
        State <= `R_Start;
        Signals <= `SR_Start | `SSRST;
      end
  end
default:
  begin
    State <= `Clear;
```

```
            Signals <= `SClear;
          end
      endcase
    end
end
endmodule
```

Literaturverzeichnis

[1] AMD Inc. *AMD Athlon Processor: Technical Brief*, 1999.

[2] Peter Ashenden. *The Designer's Guide to VHDL.* Morgan Kaufmann, 1996.

[3] Austria Mikro Systeme International. 0.8 μm ECL Standard Cells. `http://www.amsint.com/products/technology/index_b08.html`.

[4] Austria Mikro Systeme International. *CMOS-Databook.*

[5] Austria Mikro Systeme International. *0.8-Micron and 1.2-Micron BiCMOS Standard Cell Databook*, 1995.

[6] Austria Mikro Systeme International. *BYB / BYE 0.8μm BiCMOS Process*, 1997.

[7] Cadence Design Systems Inc. *Analog Artist: Simulation Help*, 1997.

[8] Cadence Design Systems Inc. *Cadence Application Infrastructure User Guide*, 1997.

[9] Cadence Design Systems Inc. *Composer: Design Entry Help*, 1997.

[10] Cadence Design Systems Inc. *Preview Cell Ensemble Reference*, 1997.

[11] Cadence Design Systems Inc. *Synergy HDL Command Reference*, 1997.

[12] Cadence Design Systems Inc. *Verilog-XL Reference*, 1997.

[13] Cadence Design Systems Inc. *Vituoso Layout Editor Help*, 1997.

[14] Michael D. Ciletti. *Modeling, Synthesis and Rapid Prototyping with the Verilog HDL.* Prentice Hall, 1999.

[15] Thomas H. Corman, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Alsgorithms.* MIT press, 1990.

[16] Danish Electronics, Light & Acoustics. DELTA Sevices - Test Services. `http://www.delta.dk/services/test-services`.

[17] P.J. Denning. The working set model for program behavior. *Communications of the ACM*, 11(5):323–333, 1968.

[18] Europractice. Homepage. `http://www.europractice.com`.

[19] Frauenhofer Institut IIs-A. Digital Virtual Tester (DVT). `http://www.iis.fhg.de/asic/dvt/index.html`.

[20] R. Geels, S. Corzine, and L. Coldren. InGaAs vertical-cavity surface emitting lasers. *IEEE Quantum Electron.*, 27:1359–1367, 1991.

[21] K. Goossen, J. Walker, L. D'Asaro, et al. GaAs MQW Modulators Integrated with Silicon CMOS. *IEEE Phot. Tech. Lett.*, 7:360–362, 1995.

[22] R.W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[23] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, Inc., 2nd edition, 1996.

[24] IBM. *PowerPC 603e Embedded Hardware Specification: Advance Information*, 1998.

[25] Institute of Electrical and Electronics Engineers. *IEEE Standard 1149.1-1990, Standard Test Access Port and Boundary Scan Architecture*, 1993.

[26] Intel Corp. *Intel Pentium 4 Processor in the 423-pin Package at 1.30 GHz, 1.40 GHz and 1.50 GHz Datasheet*, 2000.

[27] Howard W. Johnson and Martin Graham. *High Speed Digital Design.* Prentice-Hall, 1993.

[28] Randy H. Katz. *Contemporary Logic Design.* The Benjamin/Cummings Publishing Company, Inc., 1994.

[29] Jörg Keller and Wolfgang J. Paul. *Hardware Design*, volume 15 of *Teubner-Texte zur Informatik.* Teubner, 1995.

[30] Michael Klein. Entwurf und realisierung der fiberlink-testplatine. Diplomarbeit, Universität des Saarlandes, FR. 6.2 - Informatik, 2001.

[31] Michael Klein, Wolfgang Paul, Jochen Preiß, G. Renz, and Markus Scholl. Optical interconnect between cache and main memory. *LaserOpto*, 2001. to be published.

[32] Daniel Kröning. Cache-Simulationen für einen 32-bit RISC-Prozessor bei einer Standard-SPEC-Workload. FoPra-Dokumentation, Universität des Saarlandes, FB. Informatik, 1997.

[33] Glen G Langdon Jr. *Computer Design.* Computeach Press Inc., 1982.

[34] Lehrstuhl für Rechnerarchitektur der Universität des Saarlandes. Fiberlink – Mainpage. `http://www-wjp.cs.uni-sb.de/projects/fiberlink`.

[35] Lucent Technologies. Process for Fabricating OE/VLSI Chips. `http://www.lucent.com/oevlsi`.

[36] Silvia M. Müller and Wolfgang J. Paul. *The Complexity of Simple Computer Architectures.* Springer, 1995.

[37] Silvia M. Müller and Wolfgang J. Paul. *Computer Architecture: Complexity and Correctness.* Springer, 2000.

[38] Steven A. Przybylski. *Cache and Memory Hierarchy Design.* Morgan Kaufmann Publishers, Inc., 1990.

[39] Günther Renz and Markus Scholl. Persönliche Komunikation. DLR, Stuttgart, 2001.

[40] Sefan Scharl. Persönliche Komunikation. DLR, Stuttgart, 1999.

[41] Sun Microsystems Computer Corporation. *The SuperSPARC Microprocessor: Technical White Paper*, 1992.

[42] Xilinx Inc. *Virtex-E 1.8V Field Programmable Gate Arrays*, 2000.