(19) **Bundesrepublik Deutschland**
Deutsches Patent- und Markenamt

(10) **DE 602 13 601 T2** 2007.08.09

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 402 395 B1**
(21) Deutsches Aktenzeichen: **602 13 601.6**
(86) PCT-Aktenzeichen: **PCT/US02/12234**
(96) Europäisches Aktenzeichen: **02 723 894.8**
(87) PCT-Veröffentlichungs-Nr.: **WO 2002/095598**
(86) PCT-Anmeldetag: **17.04.2002**
(87) Veröffentlichungstag der PCT-Anmeldung: **28.11.2002**
(97) Erstveröffentlichung durch das EPA: **31.03.2004**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **02.08.2006**
(47) Veröffentlichungstag im Patentblatt: **09.08.2007**

(51) Int Cl.[8]: ***G06F 15/78*** (2006.01)

(30) Unionspriorität:
861112 **18.05.2001** **US**

(84) Benannte Vertragsstaaten:
DE, FR, GB, IE

(73) Patentinhaber:
Xilinx, Inc., San Jose, Calif., US

(74) Vertreter:
derzeit kein Vertreter bestellt

(72) Erfinder:
DAO, Kim, Khang, San Jose, CA 95131, US;
BAXTER, A., Glenn, Los Gatos, CA 95033, US

(54) Bezeichnung: **PROGRAMMIERBARE HARDWARELOGIK, WELCHE EIN PROGRAMMIERBARES INTERFACE UND EINE ZENTRALE RECHENEINHEIT BEINHALTET**

Beschreibung

GEBIET DER ERFINDUNG

[0001] Die vorliegende Erfindung betrifft eine programmierbare Logikvorrichtung und insbesondere eine programmierbare Logikvorrichtung mit einem Schnittstellenkern und einer zentralen Verarbeitungseinheit.

BESCHREIBUNG DER VERWANDTEN TECHNIK

[0002] Ein Mikroprozessor ist eine allgemein bekannte integrierte Schaltung, die einen Computer steuert. Die Art und Weise, auf die der Mikroprozessor den Computer steuert, bestimmt die Geschwindigkeit und Leistungsfähigkeit des Computers. Um den Computer steuern zu können, verarbeitet der Mikroprozessor eine Unmenge von Anweisungen, die ihn jeweils dazu auffordern, eine bestimmte Operation durchzuführen. Ein typischer Mikroprozessor enthält eine zentrale Verarbeitungseinheit (CPU), die die gewünschten arithmetischen und logischen Funktionen ausführt, sowie eine Eingabe/Ausgabeeinheit, die es der CPU ermöglicht, Eingangswerte von den restlichen Bestandteilen des Computers zu empfangen, um diese Funktionen ausführen zu können, und/oder ihre Ergebnisse den restlichen Bestandteilen des Computers mitzuteilen.

[0003] **Fig.** 1A stellt einen bekannten Mikroprozessor **100** dar, der eine CPU **101** und zwei Erweiterungsbusse **102** und **103** enthält. Mit Hilfe der Erweiterungsbusse kann die CPU **101** mit anderen Vorrichtungen kommunizieren, wodurch sich der Arbeitsbereich des Mikroprozessors ausweitet. Der Erweiterungsbus **102**, der auch als Prozessor-Local-Bus (PLB) bezeichnet wird, verbindet die CPU **101** mit Hochgeschwindigkeitsvorrichtungen **104**. Zu diesen Hochgeschwindigkeitsvorrichtungen **104** könnten Speicher- und Hochleistungsperipheriegeräte gehören. Eine Vorrichtung, die die Steuerung des PLB **102** übernimmt und für ihren eigenen Transfer sorgt, wird als „Master" bezeichnet, während eine Vorrichtung, die vom Master Befehle zum Senden von Daten empfängt, als „Slave" bezeichnet wird.

[0004] Der Erweiterungsbus **103**, der auch als chipintegrierter peripherer Bus (On-chip Peripheral Bus = OPB) bezeichnet wird, sorgt für den Zugang der CPU **101** zu langsamen Vorrichtungen **105**. Zu diesen langsamen Vorrichtungen **105** könnten UART-Chips und Ethernet-Verbindungen gehören. Es sei angemerkt, daß zu langsamen Vorrichtungen **105** genauso wie zu Hochgeschwindigkeitsvorrichtungen **104** sowohl Master als auch Slaves gehören können. Um jedoch zu verhindern, daß diese langsamen Vorrichtungen **105** die Leistung der CPU **101** beeinflussen, ist der OPB **103** nicht direkt mit der CPU **101** verbunden. Stattdessen wird der OPB **103** über eine OPB-Brücke **106** an den PLB **102** gekoppelt. Es sei angemerkt, daß die meisten Systeme davon ausgehen, daß die Segmentierung von Bussen erfolgt, damit langsame Vorrichtungen keine wertvolle Bandbreite auf dem Hauptbus verbrauchen. Es ist jedoch häufig von Vorteil, die Segmentierung als Bus parallel zum Hauptbus zu behandeln, die den vorteilhaften Betrieb von Peripheriegeräten mit höherer Geschwindigkeit bei geringer Auswirkung auf den Hauptbus ermöglicht. Bei der OPB-Brücke **106** handelt es sich um ein Systembauelement, das für den Transport von Daten von einem Bus zum anderen ausgelegt ist. Die OPB-Brücke **106** kann Datenformate und Protokolle automatisch umwandeln und erleichtert dadurch die Übertragung von Informationen zwischen OPB **103** und PLB **102**.

[0005] Anwendungsspezifische integrierte Schaltungen (ASICs) sind im Fachbereich der integrierten Schaltungen ebenfalls allgemein bekannt. Bei einem ASIC handelt es sich um einen Chip, der für eine vorgegebene Anwendung bestimmt ist. Zum Bau eines ASIC werden mehrere Grundschaltungszellen miteinander verbunden. Diese Schaltungszellen sind in der Regel in einer vom ASIC-Hersteller gelieferten Bibliothek zu finden. Einige Hersteller liefern derzeit Mikroprozessorzellen zum Integrieren in den ASIC. Diese Mikroprozessorarten werden „integrierte" Mikroprozessoren genannt.

[0006] Programmierbare Logikvorrichtungen (Programmable Logic Devices = PLDs) sind im Fachbereich der integrierten Schaltungen ebenfalls allgemein bekannt. Eine PLD kann vom Anwender vor Ort programmiert werden, um logische Strukturen zu implementieren. Eine PLD-Art ist das anwenderprogrammierbare Gate-Array (Field-Programmable Gate Array = FPGA). Bei einer typischen Architektur enthält ein FPGA ein Array aus konfigurierbaren Logikbausteinen (Configurable Logic Blocks = CLBs), das von programmierbaren Eingabe/Ausgabebausteinen (Input/Output Blocks = IOBs) umgeben ist. Die IOBs sorgen für die Schnittstelle zwischen den Anschlußstiften und den CLBs, während die CLBs die Funktionselemente für die Konstruktion von Logik auf dem FPGA bereitstellen. Die CLBs und die IOBs sind durch eine Hierarchie von programmierbaren Leitwegressourcen miteinander verbunden. Diese CLBs, IOBs und programmierbaren Leitwegressourcen werden durch das Laden eines Konfigurationsbitstroms in das FPGA individuell angepaßt. Dieser Konfigurationsbitstrom wird mit Hilfe von Software-Tools erzeugt.

[0007] Fig. 2 stellt ein vereinfachtes Blockdiagramm eines CLB **200** bei dem von Xilinx, Inc. vertriebenen FPGA Virtex™ dar. Der CLB **200** enthält zwei „Scheiben" **201A** und **201B**. Jede Scheibe **201** enthält einen ersten LUT **202** für das Empfangen mehrerer erster Eingangssignale G1–G4 von der programmierbaren Leiterbahn und einen zweiten LUT **203** für das Empfangen mehrerer zweiter Eingangssignale F1–F4, die ebenfalls von der programmierbaren Leiterbahn kommen. Jede Scheibe **201** enthält des weiteren ein Flip-Flop **204** für das Empfangen eines Ausgangssignals Y (an seinem Anschluß D) von dem LUT **202** und für das Ausgeben eines Signals YQ (an seinem Anschluß Q). Auf ähnliche Weise enthält jede Scheibe **201** des weiteren ein Flip-Flop **205** für das Empfangen eines Ausgangssignals X (an seinem Anschluß D) von dem LUT **203** und für das Ausgeben eines Signals XQ (an seinem Anschluß Q). Weitere Einzelheiten zu dem FPGA Virtex™ werden auf den Seiten 3–7 bis 3–17 und den Seiten 3–76 bis 3–87 in dem von Xilinx, Inc. veröffentlichten Dokument "The Programmable Logic Data Book 2000" offengelegt.

[0008] Einige FPGAs wie das FPGA Virtex können so programmiert werden, daß sie Bausteine mit einer vorgegebenen Funktionalität enthalten, die als „Kern" bezeichnet werden. Bei einer Ausführungsform kann der Kern einen vorgegebenen Satz Konfigurationsbits enthalten, die das FPGA so programmieren, dass es eine oder mehrere Funktionen ausführt. Bei einer weiteren Ausführungsform kann ein Kern Quellcode oder Schemata enthalten, die die Logik und Konnektivität einer Struktur beschreiben. Typische Kerne können unter anderem digitale Signalverarbeitungsfunktionen (Digital Signal Processing = DSP), Speicher, Speicherelemente und mathematische Funktionen bereitstellen. Kerne können für bestimmte FPGAs mit einem optimal geplanten Layout versehen werden. Kerne können auch parametrisierbar sein, d.h. sie können es dem Anwender ermöglichen, Parameter einzugeben, um eine bestimmte Kernfunktionalität zu aktivieren oder zu ändern.

[0009] Fig. 3 stellt einen IP-Prozeßaufbau **300** für die Verwendung eines oder mehrerer Kerne in einem FPGA dar. Es sei angemerkt, daß sich IP hier auf allgemeines geistiges Eigentum (Intellectual Property) wie beispielsweise Logikstrukturen für ein FPGA bezieht. Sowohl vom Anwender bereitgestellte Logikstrukturen als auch von der Inhaberin der vorliegenden Erfindung oder anderen entwickelte Kerne werden als IP angesehen. Bei dem Aufbau **300** kann ein Kerngenerator **304**, der von Tools **301** auf Systemebene aktiviert wird, einen oder mehrere Steckkerne von Drittanbietern aufnehmen. Alternativ dazu kann der Kerngenerator **304** einen Kern verwenden, der innerhalb eines Satzes Bibliotheken **303** bereitgestellt wird, wobei diese Bibliotheken **303** Bestandteile des Kerngenerators **304** sind. Wenn der Kern ausgewählt worden ist, kann er einem FPGA-Software-Tool **306** zugeführt werden. Zu beispielhafter FPGA-Software könnte die Software Alliance™, Foundation™ oder Foundation ISE™ gehören, die alle von der Inhaberin der vorliegenden Erfindung lizenziert werden. Diese Software kann ebenso von einem Anwender eine Struktur **305** auf oberster Ebene erhalten. Diese Struktur auf oberster Ebene bestimmt die Logikstruktur, die zusätzlich zu dem/den Kernen auf dem FPGA implementiert wird. Bei einer Ausführungsform kann das FPGA-Software-Tool **306** Strukturen **305** auf oberster Ebene in VHDL, Verflog oder standardmäßiger schematischer Form erhalten. Das FPGA-Software-Tool **306** erzeugt den Konfigurationsbitstrom, der ein FPGA **307** so programmiert, dass die von dem/den Kernen und der Struktur auf oberster Ebene bestimmten Funktionen bereitgestellt werden.

[0010] Im Zuge technologischer Verbesserungen tauchen FPGAs zunehmend in bisher ASIC-typischen Anwendungen auf. Bei einem FPGA bestehen jedoch größere Einschränkungen hinsichtlich der physischen Ressourcen als bei einem ASIC. Aus diesem Grunde ist das Bereitstellen von Mikroprozessor-Bestandteilen auf einem FPGA keine einfache Aufgabe und erfordert spezielle Überlegungen.

KURZDARSTELLUNG DER ERFINDUNG

[0011] Gemäß der vorliegenden Erfindung enthält eine programmierbare Logikvorrichtung (PLD) eine zentrale Verarbeitungseinheit (CPU) und eine mit der CPU gekoppelte programmierbare Schnittstelle. Die PLD enthält des weiteren mindestens eine Vorrichtung, die entweder über die programmierbare Schnittstelle Informationen an die CPU liefert, über die programmierbare Schnittstelle Informationen von der CPU empfängt oder über die programmierbare Schnittstelle mit einer anderen Vorrichtung kommuniziert. Die programmierbare Schnittstelle enthält einen Koppelpunkt-Schalter zum Koppeln der mehreren Vorrichtungen. Der Koppelpunkt-Schalter enthält sowohl Adreß/Steuerwege als auch Datenwege.

[0012] Bei einer Ausführungsform können die Adreß/Steuerwege von einem ersten Satz programmierbarer Ressourcen auf der PLD und die Datenwege von einem zweiten Satz programmierbarer Ressourcen auf der PLD gebildet werden. Der erste und der zweite Satz programmierbarer Ressourcen können sich unterscheiden oder sich auf der PLD überlappen.

[0013] Bei einer Ausführungsform enthält die PLD des weiteren einen chipinternen peripheren Bus (On-chip

Peripheral Bus = OPB), der mit der programmierbaren Schnittstelle gekoppelt ist. Ein Brückenmodul koppelt den OPB mit der programmierbaren Schnittstelle. Eine erste Gruppe von Vorrichtungen auf der PLD (einschließlich Masters und Slaves) kann mit dem OPB gekoppelt sein. Somit kann diese erste Gruppe von Masters und Slaves mit Hilfe des Brückenmoduls mit der programmierbaren Schnittstelle kommunizieren. Zu der ersten Gruppe von Masters und Slaves können Peripheriegeräte gehören, die sich in der Regel als langsame Vorrichtungen bezeichnen lassen. Bei einer Ausfuhrungsform wird mindestens einer dieser Masters und Slaves mit Hilfe eines Kerns auf der PLD erzeugt. Bei einer anderen Ausführungsform befindet sich mindestens einer dieser Masters und Slaves nicht auf dem Chip. Bei noch einer weiteren Ausführungsform kann ein Kern dafür benutzt werden, eine Vorrichtung (der ersten Gruppe) zu erzeugen, die sowohl Master- als auch Slave-Funktionalität aufweist. Die erste Gruppe von Vorrichtungen kann sich auf der PLD oder außerhalb des Chips befinden.

[0014] Eine zweite Gruppe von Vorrichtungen auf der PLD, zu der auch Masters und Slaves gehören, kann mit der programmierbaren Schnittstelle gekoppelt werden. Zu der zweiten Gruppe von Vorrichtungen können Hochgeschwindigkeitsvorrichtungen wie Speichervorrichtungen oder Vorrichtungen mit doppelter Datenrate gehören. Die zweite Gruppe von Vorrichtungen kann sich auf der PLD oder nicht auf dem Chip befinden. Bei einer Ausführungsform enthält einer der Masters in der zweiten Gruppe von Vorrichtungen eine Hochgeschwindigkeits-Busschnittstelle, die mit der programmierbaren Schnittstelle und dem OPB gekoppelt ist, wobei die Hochgeschwindigkeits-Busschnittstelle für die Kommunikation mit einer chipexternen Vorrichtung sorgt. Die PLD der vorliegenden Erfindung lässt sich problemlos in ein System integrieren, das andere Bestandteile enthält, wobei die programmierbare Schnittstelle und die Hochgeschwindigkeits-Busschnittstelle vorteilhafterweise für den effizienten Zugang dieser Bestandteile zu den die CPU betreffenden Funktionen sorgen.

[0015] Bei einer Ausführungsform der vorliegenden Erfindung enthält die programmierbare Schnittstelle einen Koppelpunkt-Schalter. Bei der Adreß/Steuerlogik kann der Koppelpunkt-Schalter mehrere Adreßdecodierer für das Empfangen von Adressen aus den Masters und mehrere Zugriffsarbiter für das Empfangen decodierter Adressen aus den Adreßdecodierern und das Arbitrieren von Transaktionen zwischen den Masters und den Slaves enthalten. Der Koppelpunkt-Schalter kann für eine vollständige oder unvollständige Konnektivität zwischen den Adreßdecodierern und den Zugriffsarbitern sorgen.

[0016] Bei der Datenlogik kann der Koppelpunkt-Schalter mehrere Schreibmultiplexer für das Empfangen von Schreibdaten aus den Masters enthalten, wobei jeder Schreibmultiplexer die Schreibdaten gezielt einem Slave zur Verfügung stellt, und mehrere Lesemultiplexer für das Empfangen von Lesedaten aus den Slaves, wobei jeder Lesemultiplexer die Lesedaten gezielt einem Master zur Verfügung stellt. Die Schreib- und die Lesemultiplexer können von den mehreren Zugriffsarbitern gesteuert werden. Der Koppelpunkt-Schalter kann für eine vollständige oder unvollständige Konnektivität zwischen jedem Schreibmultiplexer und den mehreren Master-Vorrichtungen sorgen. Der Koppelpunkt-Schalter kann ebenso für eine vollständige oder unvollständige Konnektivität zwischen jedem Lesemultiplexer und den mehreren Slave-Vorrichtungen sorgen. Bei einer Ausführungsform enthält die Adreß/Steuer- und/oder die Datenlogik des weiteren Register für das Pipelining von Transaktionen zwischen den Masters und den Slaves.

[0017] Gemäß einer anderen Ausführungsform der vorliegenden Erfindung enthält ein anwenderprogrammierbares Gate-Array (FPGA) mit einer integrierten zentralen Verarbeitungseinheit (CPU) einen von einem Kern erzeugten Hybridschalter, wobei der Hybridschalter für die Kommunikation zwischen der CPU und der auf dem FPGA implementierten Anwenderlogik sorgt. Bei dieser Ausführungsform enthält die CPU eine Master-Vorrichtung, und die Logik weist mehrere Slave-Vorrichtungen und mehrere Master-Vorrichtungen auf. Dieser Hybridschalter enthält sowohl Koppelpunkt-Konfigurationen als auch Konfigurationen mit gemeinsam genutztem Bus, die Wege zwischen den Master-Vorrichtungen und den Slave-Vorrichtungen bereitstellen.

[0018] Insbesondere enthält der Hybridschalter mehrere Adreßdecodierer für das Empfangen von Adressen aus den Masters und mehrere Zugriffsarbiter für das Empfangen decodierter Andressen aus den Adreßdecodierern und das Arbitrieren von Transaktionen zwischen den Masters und den Slaves. Mindestens ein Zugriffsarbiter sorgt für die Arbitrierung von mehr als einem Slave, und mindestens ein Zugriffsarbiter sorgt für die Arbitrierung von nur einem Slave. Der Koppelpunkt-Schalter kann ebenso mehrere Schreibmultiplexer für das Empfangen von Schreibdaten aus den Masters und für das Zuführen von Schreibdaten zu dem Slave enthalten. Bei einer Ausführungsform führt mindestens ein Schreibmultiplexer seine Schreibdaten mehr als einem Slave zu, und mindestens ein Schreibmultiplexer führt seine Schreibdaten nur einem Slave zu. Der Koppelpunkt-Schalter kann ebenso mehrere Lesemultiplexer für das Empfangen von Lesedaten aus den Slaves und für das gezielte Zuführen der Lesedaten zu den Masters enthalten. Bei einer Ausführungsform führt mindestens ein Lesemultiplexer seine Lesedaten mehr als einem Master zu, und mindestens ein Lesemultiplexer führt

seine Lesedaten nur einem Master zu. Die Schreib- und die Lesemultiplexer können von den mehreren Zugriffsarbitern gesteuert werden. Der Hybridschalter kann wie der Koppelpunkt-Schalter der vorliegenden Erfindung Pipelining für Transaktionen zwischen den Masters und den Slaves enthalten.

[0019] Gemäß einer Ausführungsform der vorliegenden Erfindung enthält ein anwenderprogrammierbares Gate-Array (FPGA) mehrere Master-Vorrichtungen, wobei ein integrierter Mikroprozessor mindestens aus einer solchen Master-Vorrichtung, mehreren Slave-Vorrichtungen und einem Mittel für das gezielte Bereitstellen von Konnektivität zwischen den mehreren Master-Vorrichtungen und den mehreren Slave-Vorrichtungen besteht. Das Mittel für das gezielte Bereitstellen von Konnektivität kann durch einen Kern erzeugt werden.

[0020] Bei der Adreß/Steuerlogik kann das Mittel für das gezielte Bereitstellen von vollständiger Konnektivität ein Mittel für das Decodieren von Adressen aus den Master-Vorrichtungen und ein Mittel für die Arbitrierung von Transaktionen zwischen den mehreren Master-Vorrichtungen und den mehreren Slave-Vorrichtungen auf der Grundlage der decodierten Adressen enthalten. Bei einer Ausführungsform sorgt das Mittel für das gezielte Bereitstellen von Konnektivität für vollständige Konnektivität zwischen dem Mittel für das Decodieren und dem Mittel für die Arbitrierung, während das Mittel für das gezielte Bereitstellen von Konnektivität bei anderen Ausführungsformen für eine unvollständige Konnektivität zwischen dem Mittel für das Decodieren und dem Mittel für die Arbitrierung sorgt.

[0021] Bei der Datenlogik kann das Mittel für das gezielte Bereitstellen von Konnektivität ein Mittel für das Empfangen von Schreibdaten aus den mehreren Master-Vorrichtungen und das gezielte Zuführen der Schreibdaten zu vorgesehenen Slave-Vorrichtungen enthalten und ein Mittel für das Empfangen von Lesedaten aus den mehreren Slave-Vorrichtungen und das gezielte Zuführen der Lesedaten zu vorgesehenen Master-Vorrichtungen. Bei einer Ausführungsform werden das Mittel für das Empfangen der Schreibdaten und das Mittel für das Empfangen der Lesedaten von dem Mittel für die Arbitrierung gesteuert.

[0022] Das Mittel für das gezielte Bereitstellen von Konnektivität kann für eine vollständige oder unvollständige Konnektivität zwischen dem Mittel für das Empfangen von Schreibdaten und den mehreren Master-Vorrichtungen sorgen. Außerdem kann das Mittel für das gezielte Bereitstellen von Konnektivität für eine vollständige oder unvollständige Konnektivität zwischen dem Mittel für das Empfangen von Lesedaten und den mehreren Slave-Vorrichtungen sorgen.

[0023] Die vorliegende Erfindung kann vorteilhafterweise in bekannten PLD-Software-Tools enthalten sein. Bei einer Ausführungsform kann ein Kerngenerator, der durch Tools auf Systemebene aktiviert wird, einen von Dritten bereitgestellten programmierbaren Schnittstellenkern der vorliegenden Erfindung aufnehmen. Alternativ dazu kann der Kerngenerator einen programmierbaren Schnittstellenkern verwenden, der in einem Satz Bibliotheken bereitgestellt wird, die Bestandteil des Kerngenerators sind. Wenn der programmierbare Schnittstellenkern ausgewählt worden ist, kann er einem PLD-Software-Tool zur Verfügung gestellt werden. Diese Software kann ebenfalls von einem Anwender eine Struktur auf oberster Ebene erhalten. Dieses PLD-Software-Tool erzeugt den Konfigurationsbitstrom, der eine PLD so programmiert, dass sie die von dem programmierbaren Schnittstellenkern und der Struktur auf oberster Ebene vorgesehenen Funktionen bereitstellt.

[0024] Es ist wichtig, dass der programmierbare Schnittstellenkern der vorliegenden Erfindung mehrere Compliance-Ebenen aufweisen kann. Ein Kern kann beispielsweise nur eine bestimmte Funktionalität bei der PLD zulassen. Bei einer Ausführungsform ist beispielsweise eine Teilmenge der Funktionen des Prozessor-Local-Busses (PLB) zugelassen. Bei anderen Ausführungsformen, wie den Ausführungsformen, die bestehendes IP oder nicht parametrisiertes IP enthalten, könnte die vorliegende Erfindung die Teilmenge auf die Funktionalität zurechtschneiden, die von diesem IP gefordert wird (oder auf eine vorgegebene Compliance-Ebene auf der Grundlage dieses IP).

[0025] Gemäß der vorliegenden Erfindung wird ein Verfahren für das Bereitstellen einer Schnittstelle zwischen einer zentralen Verarbeitungseinheit (CPU) auf einer programmierbaren Logikvorrichtung (PLD) und vom Anwender implementierter Logik auf der PLD angegeben. Das Verfahren umfaßt das gezielte Bereitstellen von Konnektivität zwischen mehreren von der Logik bereitgestellten Master-Vorrichtungen und mehreren von der Logik bereitgestellten Slave-Vorrichtungen, wobei zur CPU mindestens eine Master-Vorrichtung gehört. Zu dem Schritt des gezielten Bereitstellens von Konnektivität im Verlauf der Adreß/Steuerlogik gehört das Decodieren von Adressen aus den Master-Vorrichtungen mit Hilfe von Adreßdecodierern und das Arbitrieren von Transaktionen zwischen den mehreren Master-Vorrichtungen und den mehreren Slave-Vorrichtungen mit Hilfe von Zugriffsarbitern und der decodierten Adressen. Bei einer Ausführungsform sorgt der Schritt des gezielten Bereitstellens von Konnektivität für vollständige Konnektivität zwischen den Adreßdecodierern und den

Zugriffsarbitern. Bei einer weiteren Ausführungsform sorgt der Schritt des gezielten Bereitstellens von Konnektivität für eine unvollständige Konnektivität zwischen den Adreßdecodierern und den Zugriffsarbitern.

[0026] Bei dem Verfahren der vorliegenden Erfindung gehört zu dem Schritt des gezielten Bereitstellens von Konnektivität in der Datenlogik das Empfangen von Schreibdaten aus den mehreren Master-Vorrichtungen und das gezielte Zuführen der Schreibdaten zu vorgesehenen Slave-Vorrichtungen unter Verwendung eines ersten Satzes Multiplexer sowie das Empfangen von Lesedaten aus den mehreren Slave-Vorrichtungen und das gezielte Zuführen der Lesedaten zu vorgesehenen Master-Vorrichtungen unter Verwendung eines zweiten Satzes Multiplexer. Der Schritt des gezielten Bereitstellens von Konnektivität kann für eine vollständige oder unvollständige Konnektivität zwischen dem ersten Satz Multiplexer und den mehreren Master-Vorrichtungen sorgen. Der Schritt des gezielten Bereitstellens von Konnektivität kann ebenso für eine vollständige oder unvollständige Konnektivität zwischen dem zweiten Satz Multiplexer und den mehreren Slave-Vorrichtungen sorgen.

[0027] Bei dem Verfahren der vorliegenden Erfindung gehört zu dem Schnitt des gezielten Bereitstellens von Konnektivität das Pipelining von Transaktionen zwischen den mehreren Master-Vorrichtungen und den mehreren Slave-Vorrichtungen. Dieses Pipelining kann vorteilhafterweise in der Adreß/Steuerlogik sowie der Datenlogik bereitgestellt werden, wodurch sich bei beiden die Leistung verbessert.

[0028] Gemäß der vorliegenden Erfindung wird ein Verfahren für das Bereitstellen einer Schnittstelle zwischen einer zentralen Verarbeitungseinheit (CPU) auf einer programmierbaren Logikvorrichtung (PLD) und vom Anwender implementierter Logik auf der PLD angegeben. Das Verfahren umfaßt das Auswählen eines Kerns aus einer Bibliothek, wobei der Kern gezielt für Konnektivität zwischen mehreren von der Logik bereitgestellten Master-Vorrichtungen und mehreren von der Logik bereitgestellten Slave-Vorrichtungen sorgt, wobei zur CPU mindestens eine Master-Vorrichtung gehört. Zu dem Verfahren gehört des weiteren das Einstellen mindestens eines Parameters in dem Kern, was die Konnektivität beeinflußt.

[0029] Bei einer weiteren Ausführungsform wird der Informationsgehalt jedes Kerns, der für den Aufbau des Systems benötigt wird, zur Analyse der Daten in einer zentralen Einrichtung gesammelt. Diese Daten stehen dem Anwender zur Verfügung, damit er bei der Instanziierung von Kernen oder der Parametrisierung Alternativen wählen kann. Eine bestimmte Wahl bei der Parametrisierung des Kerns kann beispielsweise zu einer unterschiedlichen Leistung des Gesamtsystems führen.

[0030] Die vorliegende Erfindung bietet im Vergleich zu den üblichen integrierten CPUs in der ASIC-Umgebung eine Anzahl wesentlicher Vorteile. Die vorliegende Erfindung sorgt erstens für beträchtliche Flexibilität, indem sie Funktionen bereitstellt, die auf der Grundlage der Bedürfnisse des Anwenders und/ oder der zugehörigen Struktur parametrisiert werden können. Diese parametrisierten Funktionen können in programmierbaren Ressourcen auf der PLD implementiert werden, wodurch sie jederzeit modifiziert werden können. Zweitens können aufgrund dieser programmierbaren Ressourcen nur die Ressourcen implementiert werden, die tatsächlich für die programmierbare Schnittstelle benötigt werden, wodurch es dem Anwender möglich wird, die Verwendung der restlichen PLD zu optimieren. So können beispielsweise sowohl die Adreß/Steuerlogik als auch die Datenlogik „getrimmt" werden. Anders ausgedrückt können Wege (und die mit ihnen verbundene Logik), die von der Struktur des Anwenders nicht benötigt werden, eliminiert werden, wodurch sich die Signalgeschwindigkeit erhöht und die Siliziumfläche verringert. Bei einem weiteren Beispiel kann die Anzahl der Master- oder Slave-Vorrichtungen erhöht werden, da zusätzliche Fläche zur Verfügung steht. Drittens kann die vorliegende Erfindung auf effiziente Weise die Funktionen des Prozessor-Local-Busses einschränken, wodurch die PLD dem Leistungsniveau eines ASIC nahekommen beziehungsweise dieses sogar übersteigen kann.

KURZE BESCHREIBUNG DER FIGUREN

[0031] Fig. 1 zeigt eine vereinfachte schematische Darstellung eines Mikroprozessors mit einer zentralen Verarbeitungseinheit, einem Prozessor-Local-Bus und einem chipinternen peripheren Bus.

[0032] Fig. 2 stellt ein vereinfachtes Blockdiagramm eines konfigurierbaren Logikbausteins bei dem von der Inhaberin der vorliegenden Erfindung vertriebenen FPGA Virtex dar.

[0033] Fig. 3 stellt einen IP-Prozeßaufbau für die Verwendung eines oder mehrerer Kerne in einem FPGA dar.

[0034] Fig. 4 stellt ein FPGA-Grundsystem mit integriertem Prozessor gemäß der vorliegenden Erfindung dar.

[0035] Fig. 5A zeigt eine vereinfachte schematische Darstellung eines 4×4-Koppelpunkt-Schalters gemäß der vorliegenden Erfindung.

[0036] Fig. 5B zeigt eine ausführlichere schematische Darstellung einer Implementierung des Koppelpunkt-Schalters aus Fig. 5A in der Adreßlogik.

[0037] Fig. 5C zeigt eine ausführlichere schematische Darstellung einer Implementierung des Koppelpunkt-Schalters aus Fig. 5A in der Datenlogik.

[0038] Fig. 5D stellt eine Ausführungsform des Koppelpunkt-Schalters aus Fig. 5B dar, bei der die Adreß/Steuerlogik getrimmt ist.

[0039] Fig. 5E stellt eine Ausführungsform des Koppelpunkt-Schalters aus Fig. 5C dar, bei der die Datenlogik getrimmt ist.

[0040] Fig. 6 stellt eine Übersicht über die für eine Master-Vorrichtung und den programmierbaren Schnittstellenkern der vorliegenden Erfindung bereitgestellten Anschlüsse dar und die Signale, die zwischen diesen Anschlüssen übertragen werden könnten.

[0041] Fig. 7 stellt ein Zeitdiagramm eines Masters während einer 8-Datenwörter-Schreiboperation dar, der eine 8-Datenwörter-Leseoperation folgt.

[0042] Fig. 8 stellt eine Übersicht über die für eine Slave-Vorrichtung und den programmierbaren Schnittstellenkern der vorliegenden Erfindung bereitgestellten Ports dar und die Signale, die zwischen diesen Ports übertragen werden könnten.

[0043] Fig. 9 stellt ein Zeitdiagramm eines Slaves während einer 8-Datenwörter-Schreiboperation dar.

[0044] Fig. 10 stellt ein Zeitdiagramm eines Slaves während einer 8-Datenwörter-Leseoperation dar.

[0045] Die Fig. 11A und Fig. 11B stellen eine Ausführungsform der Adreß/Steuerlogik in einem programmierbaren Schnittstellenkern gemäß der vorliegenden Erfindung dar, wobei der Weg verschiedene mit dem Kern verbundene Adreß/Steuersignale enthält.

[0046] Die Fig. 12A und Fig. 12B stellen eine Ausführungsform der mit der Schreibdatenlogik gemäß der vorliegenden Erfindung verbundenen Schaltungen dar, wobei die Logik verschiedene mit dem Kern verbundene Schreibsignale enthält.

[0047] Die Fig. 13A und Fig. 13B stellen eine Ausfuhrungsform der mit der Lesedatenlogik gemäß der vorliegenden Erfindung verbundenen Schaltungen dar, wobei die Logik verschiedene mit dem Kern verbundene Lesesignale enthält.

[0048] Fig. 14 stellt eine Ausführungsform der Schnittstelle der Daten-Cache-Speichereinheit oder der Befehls-Cache-Speichereinheit zum Prozessor-Local-Bus dar.

[0049] Fig. 15 stellt die Schnittstelle eines Block-RAM (BRAM) mit Dual-Port-Funktionalität zu dem programmierbaren Schnittstellenkern der vorliegenden Erfindung dar.

[0050] Fig. 16 zeigt eine schematische Darstellung einer Ausführungsform eines Bridge-out-Moduls auf hoher Ebene in dem programmierbaren Schnittstellenkern gemäß der vorliegenden Erfindung.

[0051] Fig. 17 zeigt eine schematische Darstellung einer Ausführungsform eines Bridge-in-Moduls auf hoher Ebene in dem programmierbaren Schnittstellenkern gemäß der vorliegenden Erfindung.

[0052] Die Fig. 18A und Fig. 18B stellen die Adreß/Steuerlogik be-ziehungsweise die Datenlogik einer Architektur mit gemeinsam genutztem Bus dar.

[0053] Die Fig. 19A und Fig. 19B stellen die Adreß/Steuerlogik beziehungsweise die Datenlogik einer Architektur mit Hybridkoppelpunkt/gemeinsam genutztem Bus gemäß einer Ausführungsform der vorliegenden Erfindung dar.

AUSFÜHRLICHE BESCHREIBUNG DER FIGUREN

[0054] Das folgende Inhaltsverzeichnis soll dem Leser einen Überblick über die Strukturierung des Themas bieten.

INHALTSVERZEICHNIS

I. TERMINOLOGIE UND RICHTLINIEN

[0055] Die vorliegende Erfindung wird unter Verwendung bestimmter Terminologie und Richtlinien beschrieben, mit denen Fachleute vertraut sein mögen oder auch nicht. Aus diesem Grund werden die nachfolgenden Definitionen, Signal- und Nummerierungsrichtlinien sowie Timing-Richtlinien genauer beschrieben, um dem Leser die vorliegende Erfindung besser verständlich zu machen.

A. DEFINITIONEN

[0056] Die in dieser Beschreibung verwendeten Akronyme, Abkürzungen und Fachbegriffe werden im Anschluß an diesen Abschnitt in alphabetischer Reihenfolge definiert. Es sei darauf hingewiesen, daß einige Akronyme, Abkürzungen und Fachbegriffe eventuell in der ausführlichen Beschreibung definiert werden.

BEAR:	Busfehler-Adreßregister (Bus Error Address Register)
BESR:	Busfehler-Syndromregister (Bus Error Syndrome Register)
BRAM:	Block-RAM (Block Random Access Memory (RAM)) auf einer programmierbaren Logikvorrichtung
Burst:	eine Transaktion, bei der mehr als eine Dateneinheit übertragen und die in der Regel für das schnelle Bewegen eines Datenblocks zwischen einem Master und einem Slave verwendet wird
Cache-Speicher:	kleiner, schneller Speicher in der CPU zum Speichern von Dateninhalt einschließlich Befehle und/oder Daten, auf den in der letzten Zeit zugegriffen wurde
Kern:	vorstrukturierter Abschnitt einer programmierbaren Logikvorrichtung, der eine bestimmte Funktion bereitstellt und in der Regel mit Hilfe einer vorgegebenen Anzahl physischer Ressourcen auf der programmierbaren Logikvorrichtung implementiert wird
CPU:	zentrale Verarbeitungseinheit, die in der Regel eine Steuereinheit und eine Rechen- und Logikeinheit (Arithmetic and Logic Unit = ALU) enthält
DCU:	Daten-Cache-Speichereinheit (Data Cache Unit)
DDR:	doppelte Datenrate (Double Data Rate)
HSBI:	Hochgeschwindigkeits-Busschnittstelle, bei der hier allgemein eine Vorrichtung mit Hochgeschwindigkeits-Master-Port auf dem PIC gemeint ist
ICU:	Befehls-Cache-Speichereinheit (Instruction Cache Unit)
IP:	Struktur für die Integration in eine integrierte Schaltung, insbesondere ein FPGA
Little	Endian: Architektur, bei der, wenn man von einem 16- oder 32-Bit-Datenwort ausgeht, Byte an niedrigeren Adressen geringerwertig sind
ISA:	Befehlssatzarchitektur (Instruction Set Architecture)
OPB:	chipinterner peripherer Bus (On-chip Peripheral Bus)
PIC:	programmierbarer Schnittstellenkern (Programmable Interface Core), der zwischen einer CPU und programmierbaren Logikvorrichtungsressourcen gemäß der vorliegenden Erfindung bereitgestellt wird
Pipelining:	Prozeß, in dem eine Master-Vorrichtung mit der Ausführung eines zweiten Buszyklus beginnen kann, bevor der erste Buszyklus abgeschlossen worden ist, d.h. gleichzeitiges Verarbeiten mehrerer Buszyklen, die sich in unterschiedlichen Stadien befinden
PLB:	Prozessor-Local-Bus
PowerPC:	RISC-Mikroprozessor, der einer Norm entspricht, die von IBM, Motorola und Apple Computer gemeinsam entworfen wurde und eine gemeinsame Befehlssatzarchitektur (Instructions Set Architecture = ISA) festlegt
Register:	Hochgeschwindigkeits-Speicherbereich auf dem FPGA
SDRAM:	synchroner dynamischer RAM (DRAM) der mit höheren Taktgeschwindigkeiten laufen kann als herkömmlicher Speicher und sich selbst mit dem Bus einiger CPUs synchronisieren kann
Single	Data Beat: nicht burstartiger Transfer, d.h. ein einziges Datenelement in einem Taktzyklus über den Bus
Datenwort:	steht für eine Datenmenge von 32 Bit

B. SIGNALBEZEICHNUNGS- UND NUMMERIERUNGSRICHTLINIEN

[0057] Zur Beschreibung von Merkmalen der vorliegenden Erfindung wird auf verschiedene Signale verwiesen. Gemäß der hier verwendeten Richtlinie beginnen die Signalbezeichnungen in den Fig. 6 und Fig. 8 mit einem Präfix, der die Richtung des Signalflusses angibt sowie die übertragende Vorrichtung und die Vorrichtung, die das Signal empfängt. Die Signalbezeichnungen in anderen Figuren können mit einem Präfix beginnen, der nur die übertragende Vorrichtung angibt. Die Bezeichnungen der Vorrichtungen werden der Einfachheit und Übersichtlichkeit halber abgekürzt. Auf den Präfix folgt ein Unterstrich (_) und dann die Bezeichnung des Signals selbst. Der Suffix „_n" bezeichnet ein Active-Low-Signal. Die Signalbezeichnung PIC2M_foo beschreibt beispielsweise das Signal „foo", das von dem programmierbaren Schnittstellenkern (PIC) gesteuert und von einer Master-Vorrichtung empfangen wird.

[0058] N-Bit-Busbits sind von rechts (0) nach links (N-1) nummeriert. Es sei angemerkt, daß diese Nummerierungsrichtlinie unabhängig von Kompatibilitätsproblemen bei verschiedenen proprietären CPUs nicht die PIC-ISA ändert oder die Software-Kompatibilität der CPU beeinflußt.

C. TIMING-RICHTLINIEN

[0059] Um eine hohe Leistung erzielen zu können, muß das Timing berücksichtigt werden. Fachleute wissen, dass es für eine bestimmte Implementierung einer Struktur häufig ein spezifisches Timing gibt. Daher unterstützen die hier im Sinne von Beispielen angeführten Timing-Bezugswerte zwar das Pipelining der vorliegenden Erfindung, sie betreffen jedoch nicht das Timing aller Strukturen.

[0060] Als Timing-Informationen können die Begriffe „frühes", „mittleres" und „spätes" verwendet werden. Bei der hier verwendeten Richtlinie eignet sich frühes Timing am besten und spätes Timing am wenigsten.

[0061] Wenn ein Eingangssignal frühes Timing aufweist, dann steht dem Anwender für das Verarbeiten der Daten ein ganzer Taktzyklus zur Verfügung, bevor er sie zwischenspeichert. Ein frühes Timing für Eingangssignale bedeutet, daß das Signal von einem Registerausgang angesteuert wird. Wenn ein Ausgangssignal frühes Timing aufweist, dann kann der Anwender den ganzen Taktryklus nutzen, bevor er die Daten absendet. Ein frühes Timing für Ausgangssignale bedeutet, daß das Signal den Eingang eines Registers ansteuert.

[0062] Wenn ein Eingangssignal mittleres Timing aufweist, dann steht dem Anwender für das Verarbeiten der Daten ungefähr ein halber Taktryklus zur Verfügung, bevor er sie zwischenspeichert. Ein mittleres Timing für Eingangssignale bedeutet, daß das Signal von einem Registerausgang durch eine geringe Menge Logik oder Leitwege (Routing) zum Anwender angesteuert wird. Wenn ein Ausgangssignal mittleres Timing aufweist, dann kann der Anwender ungefähr den halben Taktzyklus nutzen, bevor er die Daten absendet. Ein mittleres Timing für Ausgangssignale bedeutet, daß das Signal eine geringe Menge Logik oder Leitwege ansteuert, bevor es am Eingang eines Registers eintrifft.

[0063] Wenn ein Eingangssignal spätes Timing aufweist, dann steht dem Anwender für das Verarbeiten der Daten ein geringer Bruchteil des Taktryklus zu Verfügung, bevor er sie zwischenspeichert. Ein spätes Timing für Eingangssignale be-deutet, daß das Signal von einem Registerausgang über eine wesentliche Menge Logik und Leitwege angesteuert wird. Somit steht dem Anwender ein kleines Fenster zur Verfügung, in dem er die Daten vor ihrer Benutzung zwischenspeichern kann. Wenn ein Ausgangssignal spätes Timing aufweist, dann kann der Anwender einen geringen Bruchteil des Taktzyklus nutzen, bevor er die Daten absendet. Ein spätes Timing für Ausgangssignale bedeutet, daß das Signal den Eingang eines Registers über eine wesentliche Menge Logik oder Leitwege ansteuert. Somit sollte der Anwender diesen Ausgangswert direkt von einem Register aus ansteuern.

[0064] Angesichts der oben genannten Probleme kann die spezielle Implementierung die Mittel, die jedes Signal bereitstellt, erklären. Einige Signale benötigen mehr Zeit (z.B. späte Signale), und dies kann nachteilige Auswirkungen auf die Schalthäufigkeit des Systems haben. Um diese Probleme anzugehen, kann Pipelining implementiert werden, das das Timing von Signalen abschwächt. Diese Pipeline-Register bieten die Möglichkeit, das Signal neu zu synchronisieren, so daß es, wenn auch mit einer Latenz, früher in der Taktperiode vorkommt.

II. ERKENNEN DER UNTERSCHIEDE ZWISCHEN RESSOURCEN

[0065] Wenn man eine zentrale Verarbeitungseinheit (CPU) statt in einen ASIC in eine programmierbare Logikvorrichtung (z.B. ein FPGA) integrieren will, muß man die Unterschiede zwischen diesen beiden Vorrichtungsarten kennen.

[0066] Ein Hauptunterschied zwischen einem FPGA und einem ASIC, der die Leistung beinflussen kann, sind die Logikressourcen. Insbesondere werden die Logikgeneratoren in einem FPGA wie oben beschrieben in der Regel mit Hilfe von Nachschlagetabellen in den CLBs aufgebaut. Somit erfordern eine hohe Eingangsauffächerung oder komplexe kombinatorische logische Funktionen in der Regel mehrere Ebenen von Nachschlagetabellen, wodurch beträchtliche Verzögerungen entstehen. Im Gegensatz dazu kann ein ASIC speziell angepaßte Strukturen für das Ausführen bestimmter logischer Funktionen bieten, wodurch die mehreren Logikebenen eliminiert werden.

[0067] Ein weiterer Hauptunterschied zwischen einem FPGA und einem ASIC, der die Leistung beinflussen kann, sind die Leitwegressourcen. Und zwar leitet ein FPGA Signale sowohl über programmierbare als auch über fest zugeordnete (beispielsweise Übertragungskettenlogik) Signalflußkanäle. Im Gegensatz dazu bietet ein ASIC speziell angepaßte Signalwege und eliminiert dadurch die Unkosten für die Programmierbarkeit. Um an Geschwindigkeiten heranzukommen, die einem ASIC entsprechen, bräuchte ein FPGA somit dickere Spu-

ren und größere Puffer als die, die in der Regel in einem FPGA bereitgestellt werden. Ein Vergrößern dieser Leitwegressourcen könnte eine Vergrößerung der Siliziumfläche des FPGA erforderlich machen und dadurch dessen Herstellungskosten auf unerwünschte Weise erhöhen.

[0068] Angesichts der oben beschriebenen Unterschiede können FPGA-Ressourcen, die an eine CPU angekoppelt sind, im Hinblick auf das Erzielen von Leistung und Bandbreite bei einer Hochgeschwindigkeitsstruktur beträchtliche Hindernisse darstellen. Gemäß der vorliegenden Erfindung, und wie nachfolgend ausführlich beschrieben wird, kann der Aufbau des FPGA auf vorteilhafte Weise wirksam eingesetzt werden, um sowohl standardmäßige als auch erweiterte Transaktionen, an denen die CPU beteiligt ist, bereitstellen zu können. Das sich so ergebende System kann derzeitigen ASIC-Systemen nahekommen und sogar über diese hinausgehen, da es statt IC-Technologie Architektur benutzt, um das Problem anzugehen.

III. EINGRENZEN VON PLB-FUNKTIONEN

[0069] Eine typische CPU arbeitet mit einem Satz PLB-Protokolle, wenn sie mit anderen Vorrichtungen kommunizieren will. Ein FPGA mit einem ganzen Satz PLB-Protokolle auszustatten, würde beträchtlich viel Logik erforderlich machen, wodurch sowohl Leistung als auch Fläche negativ beeinflußt werden würden. Deshalb können gemäß eines Merkmals der vorliegenden Erfindung einige der standardmäßigen PLB-Protokolle vorteilhafterweise eliminiert werden, ohne daß die Leistung des FPGA wesentlich beeinflußt wird.

[0070] Allgemein gesagt eliminiert die vorliegende Erfindung die PLB-Protokolle, die eine komplexe Datenwegmanipulation erfordern. Eine komplexe Datenwegmanipulation verwendet eine beträchtliche Menge an Logik, die über breite Busse und weite Entfernungen verteilt ist. Somit kann eine komplexe Datenwegmanipulation in einem FPGA beträchtliche Ressourcen für sich beanspruchen, während sie nur eine begrenzte Funktionalität für den IP-Anwender hinzufügt.

[0071] Die vorliegende Erfindung eliminiert auch die PLB-Protokolle, die ein asynchrones Handshaking erfordern. Bei einem asynchronen Handshaking legt ein Signal in der Regel große Entfernungen zurück (d.h. Wege vom Master zum Slave und zurück zum Master) und durchläuft eine beträchtliche Menge Logik, und das alles in einem einzigen Taktzyklus. Somit muß das asynchrone Handshaking, wie auch die komplexe Datenwegmanipulation, beträchtliche Ressourcen für sich in Anspruch nehmen, damit es diese Timing-Aufgabe erfüllen kann.

[0072] Durch das fehlende Unterstützen diverser standardmäßiger PLB-Protokolle lassen sich verschiedene Vorteile erzielen. Zum ersten kann die vorliegende Erfindung kritische Wege in der Datenlogik wesentlich vereinfachen. Die Vereinfachung der Datenweglogik ist wichtig, da diese breiten Busse im FPGA weite Entfernungen zurücklegen. Anders ausgedrückt läßt sich durch das Verringern der Logik im Datenweg vorteilhafterweise die Signalgeschwindigkeit erhöhen. Zum zweiten kann die vorliegende Erfindung kritische Wege in der Steuerlogik wesentlich vereinfachen, wodurch die Signalgeschwindigkeit weiter erhöht wird. Zum dritten können Transaktionen durch das Entfernen dieser PLB-Protokolle deterministischer werden. Beispielsweise kann Logik in der Regel vereinfacht werden, wenn ein Master für eine bestimmte Transaktion einen kleineren Satz möglicher Antworten von einem Slave erwarten kann. Es ist klar, daß die Logik, je mehr unterschiedliche Antworten oder Aktionen ein Master oder ein Slave aufweisen kann, umso komplexer sein muß, damit sie diese Antworten oder Aktionen unterbringen kann.

[0073] Es sei angemerkt, daß das Eliminieren von PLB-Protokollen eine direkte Auswirkung auf die Anzahl der PLB-Funktionen hat, die mit der CPU ausgeführt werden können. Die vorliegende Erfindung behandelt dieses Problem jedoch, wie nachfolgend ausführlich beschrieben wird.

A. PLB-TEILMENGENFUNKTIONEN

[0074] Gemäß der vorliegenden Erfindung konzentriert sich eine Teilmenge der in dem FPGA implementierten PLB-Funktionen auf die Arten der Transaktionen, die die CPU in der Regel durchführt, wodurch die Leistung dieses kritischen Bestandteils optimiert wird. Insbesondere erfolgen die meisten Datentransfers auf dem PLB von Vorrichtung zu Speicher oder von Speicher zu Vorrichtung und nicht von Vorrichtung zu Vorrichtung. Somit ist Speicherleistung am PLB sehr gefragt. Es ist wichtig, daß die meisten FPGA-Anwenderspeicherschnittstellen ähnliche Transaktionen ausführen wie die CPU. Folglich kann eine Teilmenge der PLB-Funktionen vorteilhafterweise die unterschiedlichen Transaktionsarten unterstützen, die am wahrscheinlichsten auf dem FPGA auftreten. Eine Analyse hinsichtlich der Integration verschiedener PLB-Funktionen in ein FPGA folgt anschließend.

1. ARBITRIERUNGSSTEUERFUNKTIONEN

[0075] Adreßdecodier-Funktion: Das Implementieren einer Adreßdecodierung in Slaves bei einem standardmäßigen PLB erfolgt derzeit über einen gemeinsam genutzten Bus (Shared Bus). Bei einem gemeinsam genutzten Bus wird die von einem Master angeforderte Adresse allen Slaves auf dem Bus zugeführt. Die Slaves decodieren die Adresse, und der ausgewählte Slave leitet dann eine Adreßbestätigung zurück an den anfordernden Master. Dieser Prozeß erzeugt leider eine beträchtliche Verzögerung. Daher wird gemäß der vorliegenden Erfindung das Adreßdecodieren verteilt und in einem programmierbaren Schnittstellenkern (Programmable Interface Core = PIC) durchgeführt (Beschreibung siehe Fig. 5A–Fig. 5E). Aufgrund des die Transaktion weiterleitenden Charakters dieses Kerns ist dem Slave ein Adreßtreffer (Hit) sicher, wenn ihm eine Transaktion präsentiert wird. Auf diese Weise verringert die vorliegende Erfindung die derzeit in den Slaves bereitgestellte Adreßdecodierlogik beträchtlich, was das Entwickeln und Bauen von Slaves vereinfacht. Es sei angemerkt, daß bei einem Hybridsystem ein zusätzlicher Adreßdecodierer in einem Slave bereitgestellt werden kann, wenn mehrere Slaves einen Bus gemeinsam nutzen. Bei diesem Hybridsystem ist die Anzahl der Slaves, die einen Bus gemeinsam nutzen, relativ gering, wodurch die Adreßdecodierlogik vereinfacht wird.

[0076] Busfehler- (Adreß- oder Datenfehler-)Funktion: Die Busfehler-Funktion wird sowohl vom standardmäßigen PLB als auch vom PIC der vorliegenden Erfindung unterstützt.

[0077] Abbruch-Funktion: Die Unterstützung einer Abbruch-Funktion bei einem Transaktionen weiterleitenden System ist schwierig, weil die Transaktion zu einem Slave weitergeleitet werden kann, bevor es überhaupt dazu kommt, daß ein Abbruchsignal überprüft wird. Es sei angemerkt, daß bei einem standardmäßigen PLB eine Transaktion nicht mehr abgebrochen werden kann, wenn ein Slave die Anforderung erst einmal bestätigt hat. Darüber hinaus wäre für das Leiten eines Abbruchbefehls in den PIC zusätzliche Logik notwendig, was negative Auswirkungen auf das Timing des Systems hätte. Daher erzeugen Masters bei der vorliegenden Erfindung in der Regel keine Abbruchsignale.

[0078] Wenn ein Master ein Abbruchsignal erzeugt, kann er ein Schnittstellenmodul benutzen, das die Anforderung sofort erkennt, sie bestätigt und direkt an den entsprechenden Slave weiterleitet. Dieses Schnittstellenmodul befindet sich auf dem Weg zwischen dem Master und dem Port zum PIC. Somit bleibt die allgemeine Regel, daß Masters keine Abbruchsignale erzeugen, erhalten.

[0079] Es sei angemerkt, daß das Unterstützen von Abbruchsignalen bei einem System mit Pipelining zu einer gewissen Leistungssteigerung führen kann, wenn sich der Code des Anwenders verzweigt oder unterbrochen wird. Insbesondere können, wenn Abbrüche unterstützt werden und Masters Zeit zum Überprüfen von Abbrüchen bekommen, unnötige Transaktionen frühzeitig abgebrochen werden, wodurch die Ausnutzung der Datenlogik auf vorteilhafte Weise reduziert wird. Sich verzweigender oder unterbrochener Code beseitigt den Informationsbedarf, der vom Master derzeit eingefordert wird. Wenn der Master Transaktionen abbrechen kann, nachdem er festgestellt hat, daß die Daten nicht gebraucht werden, dann erhöht sich die Leistung. Dieser Nutzen kann jedoch durch die Komplexität und die niedrigere Taktfrequenz für den Rest des Systems wieder aufgehoben werden.

[0080] Priorisierte Arbitrierungsfunktion: Der Aufbau kombinatorischer Logik, die auf der Grundlage einer dynamischen Priorität zwischen Masters arbitriert, würde zahlreiche logische Ebenen erfordern, die die Leistung bei einem FPGA beträchtlich verlangsamen könnten. Daher ist bei einer Ausführungsform der vorliegenden Erfindung eine Arbitrierung im Rundlauf (Round-Robin) oder mit fester Priorität vorgesehen. Bei anderen Ausführungsformen kann eine vom Anwender auszuwählende Arbitrierung bereitgestellt werden.

[0081] Wiederholungsfunktion: Bei einer Shared-Bus-Architektur wird in der Regel eine Wiederholungsfunktion (manchmal auch als Rearbitrierungsfunktion bezeichnet) benötigt, um zu verhindern, daß sich mehrere Masters gegenseitig blockieren. Gemäß dem PIC der vorliegenden Erfindung können jedoch mehrere Transaktionen gleichzeitig ablaufen, wodurch die Notwendigkeit von Wiederholungen wesentlich verringert oder sogar ganz beseitigt wird. Es sei angemerkt, daß die Implementierung einer Wiederholungsfunktion in einem Transaktionen weiterleitenden System das Leiten von Signalen vom Slave zum Master erforderlich machen würde, was die Komplexität von Logik in dem Arbiter auf unerwünschte Weise erhöht.

[0082] Es sei angemerkt, daß eine Wiederholungsfunktion in einem Hybridsystem mit einem gemeinsam genutzten Bus bereitgestellt werden kann. Bei einem Hybridsystem könnte der Zugriffsarbiter so modifiziert werden, daß er eine Wiederholungsfunktion aktiviert, wodurch es nicht mehr notwendig wäre, das Wiederholungssignal zum Master zurückzuleiten.

[0083] Busspen-Funktion: Eine Busspen-Funktion stellt sicher, daß nur ein Master auf einen bestimmten Slave zugreifen kann. Eine Busspen-Funktion kann jedoch die Arbitrierungslogik in dem PIC beeinflussen, die sich auf dem kritischen Timing-Weg befindet. Somit kann die Busspen-Funktion aktiviert werden, wenngleich dies wahrscheinlich auch mit einem gewissen Leistungsverlust einhergeht.

[0084] Adressen-Pipelining-Funktion: Eine Adressen-Pipelining-Funktion wird sowohl beim standardmäßigen PLB als auch beim PIC der vorliegenden Erfindung unterstützt. Pipelining kann die Systemleistung beträchtlich verbessern, da es den gleichzeitigen Ablauf mehrerer Transaktionen ermöglicht. Es sei darauf hingewiesen, daß eine Code-Verzweigung die Vorteile des Pipelining untergraben kann.

[0085] Transfersart-Funktion: Zu einer Transferart-Funktion bei einem standardmäßigen PLB kann beispielsweise direkter Speicherzugriff (Direct Memory Access = DMA) und Simultanbetrieb gehören. Im allgemeinen erfordert jede Transferart eine Kombination aus verschiedenen Protokollen. Daher kann das Bereitstellen mehrerer Transferarten ein FPGA-System wesentlich komplexer machen. Gemäß der vorliegenden Erfindung sind Übertragungen auf einfache Speichertransfers (Umspeicherungen) beschränkt, was die Logik und die Komplexität des FPGA vereinfacht.

2. DATENTRANSFERFUNKTIONEN

[0086] Einzeldatenimpulstransferfunktion: Eine Einzeldatenimpulstransferfunktion, d.h. die Übertragung einer einzelnen Dateneinheit, ist eine Grundoperation der CPU und wird daher von der vorliegenden Erfindung unterstützt. Es sei angemerkt, daß es sich bei dieser Transferfunktion um entweder eine Schreib- oder eine Leseoperation handeln und diese ein beliebiges Bytefreigabemuster enthalten kann. Jede Transaktion wird maximal eine Datenmenge übertragen, die der Größe der Busbreite entspricht. Es sei darauf hingewiesen, daß sich diese Funktion vom Burst-Modus unterscheidet.

[0087] Burst-von-unbestimmter-Länge-Funktion: Durch das Bereitstellen einer Burst-von-unbestimmter-Länge-Funktion auf Adreß- und Datenwegen mit Pipelining, wie dies bei der vorliegenden Erfindung geschieht, entstehen beträchtliche Leistungsprobleme. Der Master würde beispielsweise einen oder mehrere Zyklen benötigen, um das Ende eines Burst an den angeforderten Slave zu übermitteln. Im Verlauf dieser Zeit könnte der Slave jedoch zu viele Daten senden. Darüber hinaus kann es sich für die Adreßdecodierlogik als schwierig erweisen, rechnerisch zu bestimmen, ob ein Burst von unbestimmter Länge eine Adreßgrenze in dem zugeordneten Zeitraum überschreitet. Deshalb werden Bursts von unbestimmter Länge von der vorliegenden Erfindung nicht unterstützt.

[0088] Burst-Beenden-Funktion: Ein standardmäßiger PLB erlaubt dem Slave jederzeit, einen Burst von festgelegter Länge abzubrechen. Der Master kann jedoch wieder einen oder mehrere Zyklen benötigen, um dem Slave ein Beenden-Signal zu übermitteln. Im Verlauf dieses Zeitraums könnte der Slave jedoch zu viele Daten senden. Deshalb unterstützt die vorliegende Erfindung die Burst-Beenden-Funktion bei einer Ausführungsform nicht. Stattdessen erfordert die vorliegende Erfindung deterministisches Verhalten, d.h. der Slave schickt die richtige Anzahl Lesebestätigungs- (RdAck) und Schreibbestätigungssignale (WrAck) zurück. Bei einer weiteren Ausführungsform kann der Slave, wenn ein Burst-Beenden-Signal für die Steuerung des Datenflusses verwendet wird, Wartezustände zwischen RdAck- und WrAck-Signale einfügen, um mehr Zeit zur Verfügung zu stellen. Alternativ dazu kann der Slave Lesefehler markieren oder Interrupts erzeugen, wenn ein Problem auftritt.

[0089] Burst-von-bestimmter-Länge-Funktion: Durch das Bereitstellen eines Burst von bestimmter Länge, d.h. zwischen 2 und 16 Transfers, vereinfacht sich zwar die Struktur des Masters, es ist aber wiederum zusätzliche Logik für den Slave notwendig, damit eine Überschreitung der Adreßgrenze oder Seitengrenze erfaßt werden kann. Deshalb wird diese Funktion bei der vorliegenden Erfindung bei einer Ausführungsform nicht unterstützt, und stattdessen wird für eine Cache-Line-Ausrichtung gesorgt. Bei dieser Ausfuhrungsform kann der PIC Bursts mit festgelegter Länge in Zweierpotenzen durchführen, die auf diese Größe ausgerichtet sind (d.h. einen Cache-Line-Transfer). Bei einer weiteren Ausführungsform wird die Burst-von-bestimmter-Länge-Funktion unterstützt, wobei der Anwender die Adreßkennzeichner definiert und das Master/Slave-IP konstruiert, das diese Kennzeichner unterstützt. Bei einer weiteren Ausführungsform wird die Burst-von-bestimmter-Länge-Funktion forciert als zur Burstlänge ausgerichtete Transfers implementiert, wodurch keine Adreßgrenzenüberprüfungen mehr ausgeführt werden müssen.

[0090] Burst-Größe-Funktion: Ein Verändern der Burst-Größe erhöht die Komplexität von Datenwegen im Slave. Der Slave müßte insbesondere Datenwege implementieren, um mit jeder beliebigen Anzahl von Daten-

breiten umgehen zu können. Bei einer Ausführungsform kann die Burst-Größe auf die Datenwegbreite eingestellt werden.

[0091] Cache-Line-Transfer-Funktion: Cache-Line-Transfers, d.h. atomare Daten-Bursts von bestimmter Länge und Ausrichtung, vereinfachen die Struktur der Slaves im Hinblick auf Adreß- oder Seitengrenzenausrichtung. Cache-Line-Transfers werden gemäß der vorliegenden Erfindung unterstützt. Bei einer Ausführungsform der vorliegenden Erfindung werden Cache-Line-Transfers von 2, 4, 8, 16 und 32 Datenwörtern bereitgestellt. Es sei darauf hingewiesen, daß diese Größen vorteilhafterweise DDR SDRAM unterstützen.

[0092] Dynamische Busabmessungsfunktion: Die dynamische Busabmessungsfunktion erfordert Datenspiegelungs- und -lenkungslogik, was die Komplexität des FPGA beträchtlich erhöhen kann. Darüber hinaus müssen sowohl Masters als auch Slaves die Transferbreiten kennen, damit sie ihre interne Logik für die richtige Anzahl Datentransferzyklen einstellen können. Deshalb unterstützt der PIC gemäß der vorliegenden Erfindung keine Masters und Slaves mit unterschiedlichen Breiten. Stattdessen sind die Transferbreiten über das gesamte System hinweg festgelegt, wodurch sich die Komplexität beim FPGA verringert. Bei einer Ausführungsform kann parametrisiertes IP auf 32- oder 64-Bit-Transfers eingestellt werden. Der Auswirkung der dynamischen Busabmessung läßt sich mit Hilfe von Bytefreigabesignalen und Software-Steuerung Rechnung tragen.

B. PLB-OBERMENGENFUNKTIONEN

[0093] Gemäß der vorliegenden Erfindung stellen eine Anzahl implementierter Funktionen Obermengen der standardmäßigen PLB-Funktionen dar. Die vorliegende Erfindung unterstützt beispielsweise Datenwege mit doppelter Datenrate (DDR), wobei der Datenweg mit der doppelten Taktfrequenz des Adreßweges laufen kann. Zusätzlich dazu enthält die vorliegende Erfindung einen dynamischen Schreib-Bytefreigabemechanismus, damit Schreibtransaktionen in der gesamten Verbindungsstruktur optimiert werden können. Insbesondere kann die vorliegende Erfindung für jedes Byte übertragener Schreibdaten ein zusätzliches Bit bereitstellen, so daß jedes Datenbyte selektiv übersprungen werden kann. So kann der dynamische Schreib-Bytefreigabemechanismus auf effiziente Weise Byte-Muster schreiben, wodurch die Systemleistung bei nicht zusammenhängenden Speicherschreiboperationen verbessert wird. Dies ist besonders vorteilhaft bei Systemen, die DMA verwenden. Insbesondere würde eine DMA-Maschine keine komplexe Logik mehr benötigen, um schlecht ausgerichtete Daten sequentiell zu durchlaufen. Stattdessen kann die DMA-Maschine einfach das richtige Bytefreigabemuster bereitstellen, um den Inhalt des Transfers unterzubringen. Des weiteren wird die DMA-Maschine nicht mehr so viele Buszyklen für den Transfer ihrer Daten verbrauchen, wodurch dem System mehr verfügbare Leistung zur Verfügung steht. Und schließlich benutzt der PIC der vorliegenden Erfindung eine Transaktionen weiterleitende Architektur, die Pipelining unterstützt und eine höhere Gesamtleistung des Systems ermöglicht. Diese Obermengenfunktionen erhöhen die IP-Leistung bei dem FPGA beträchtlich.

C. SIGNALE: ABBILDEN (MAPPING) VON PLB AUF PIC

[0094] Um die oben beschriebenen PLB-Funktionen bereitstellen zu können, werden verschiedene Signale zwischen dem PLB und den Masters/Slaves des Systems übertragen. Diese PLB-Signale werden nachfolgend aufgeführt, um das Abbilden (Mapping) auf die vorliegende Erfindung zu zeigen. Tabelle 1 führt beispielsweise verschiedene PLB-Signale aus der Perspektive eines Masters auf, und sie gibt an, ob diese Signale bei einer einfachen CPU benötigt werden und bei einem FPGA mit akzeptabler Leistung implementiert werden können.

TABELLE 1

SIGNALBEZEICHNUNG	RICHTUNG	VON CPU BENÖTIGT	IM FPGA
PLB-Anforderungskennzeichner			
SYS_plbClk	Eingabe	Ja	Ja
SYS_plbReset	Eingabe	Ja	Ja
PLB_MnAddrAck	Eingabe	Ja	Ja
PLB_MnRearbitrate	Eingabe	Nein	Ja (gemeinsam genutzter Bus)
PLB_MnSSize	Eingabe	Nein	Nein
PLB_MnBusy	Eingabe	Ja	Ja
PLB_MnErr	Eingabe	Ja	Ja
PLB_pendReq	Eingabe	Nein	Ja (feststehender Wert)
PLB_pendPri	Eingabe	Nein	Ja
PLB_reqPri	Eingabe	Nein	Ja (feststehender Wert)
Mn_request	Ausgabe	Ja	Ja
Mn_priority	Ausgabe	Nein	Ja
Mn_busLock	Ausgabe	Nein	Ja
Mn_RNW	Ausgabe	Ja	Ja
Mn_BE	Ausgabe	Ja (kein Burst)	Ja (kein Burst oder Bursts mit festgelegter Länge)
Mn_size	Ausgabe	Ja (kein Burst oder Cache-Line)	Ja (kein Burst, Bursts mit festgelegter Länge oder Cache-Line)
Mn_type	Ausgabe	Nein	Nein
Mn_Msize	Ausgabe	Nein	Nein
Mn_compress	Ausgabe	Ja	Ja
Mn_guarded	Ausgabe	Ja	Ja
Mn_ordered	Ausgabe	Nein	Ja
Mn_locker	Ausgabe	Nein	Ja
Mn_abort	Ausgabe	Ja	Nein
Mn_Abus	Ausgabe	Ja	Ja
PLB-Schreibdatenbus			
PLB_MnWrDAck	Eingabe	Ja	Ja
PLB_MnWrBTerrn	Eingabe	Nein	Ja
Mn_wrBurst	Ausgabe	Nein	Nein
Mn_wDBus	Ausgabe	Ja	Ja
PLB-Lesedatenbus			
PLB_MnRdAck	Eingabe	Ja	Ja
PLB_MnRdBTerm	Eingabe	Nein	Ja
PLB_MnRdWdAddr	Eingabe	Ja	Ja
PLB_MnRdDBus	Eingabe	Ja	Ja
Mn_rdBurst	Ausgabe	Nein	Nein
Obermengensignale			
SYS_plbClk2x	Eingabe	Nein	Ja
PLB_MnWrAck	Eingabe	Nein	Ja

[0095] Tabelle 2 führt verschiedene PLB-Signale aus der Perspektive eines Slave auf, und sie gibt an, ob diese Signale bei einer einfachen CPU benötigt werden und bei einem FPGA mit akzeptabler Leistung implementiert werden können.

TABELLE 2

SIGNALBEZEICHNUNG	RICHTUNG	IN CPU BENÖTIGT	IM FPGA
PLB-Transferkennzeichner			
SYS_plbClk	Eingabe	Ja	Ja
SYS_plbReset	Eingabe	Ja	Ja
PLB_PAValid	Eingabe	Ja	Ja
PLB_busLock	Eingabe	Nein	Ja
PLB_pendReq	Eingabe	Nein	Ja
PLB_pendPri	Eingabe	Nein	Ja (feststehender Wert)
PLB_reqPri	Eingabe	Nein	Ja (feststehender Wert)
PLB_masterID	Eingabe	Nein	Ja
PLB_RNW	Eingabe	Ja	Ja
PLB_BE	Eingabe	Ja (kein Burst)	Ja (kein Burst oder Bursts mit festgelegter Länge)
PLB_size	Eingabe	Ja (kein Burst oder Cache-Line)	Ja (kein Burst, Bursts mit festgelegter Länge oder Cache-Line)
PLB_type	Eingabe	Nein	Nein
PLB_Msize	Eingabe	Nein	Ja (feststehender Wert)
PLB_compress	Eingabe	Ja	Ja
PLB_guarded	Eingabe	Ja	Ja
PLB_ordered	Eingabe	Nein	Ja
PLB_lockErr	Eingabe	Nein	Ja
PLB_abort	Eingabe	Nein	Ja (feststehender Wert)
PLB_Abus	Eingabe	Ja	Ja
Sl_addrAck	Ausgabe	Ja	Ja
Sl_wait	Ausgabe	Nein	Nein
Sl_Ssize	Ausgabe	Nein	Nein
Sl_rearbitrate	Ausgabe	Nein	Ja (gemeinsam genutzter Bus)
Sl_Mbusy	Ausgabe	Ja	Ja
Sl_Merr	Ausgabe	Ja	Ja
PLB-Adreß-Pipelining			
PLB_SAValid	Eingabe	Ja (Leistung)	Ja
PLB_rdprim	Eingabe	Ja (Leistung)	Ja
PLB_wrPrim	Eingabe	Ja (Leistung)	Ja
PLB-Schreibdatenbus			
PLB_wrDBus	Eingabe	Ja	Ja
PLB_wrBurst	Eingabe	Nein	Nein
Sl_wDAck	Ausgabe	Ja	Ja
Sl_wrComp	Ausgabe	Ja	Ja
Sl_wrBTerm	Ausgabe	Nein	Ja
PLB-Lesedatenbus			
PLB_rdBurst	Eingabe	Nein	Nein
PLB_Sl_rdDBus	Ausgabe	Ja	Ja
PLB_rdWdAddr	Ausgabe	Ja	Ja

IV. FPGA-IMPLEMENTIERUNG

[0096] Gemäß einer Ausführungsform der vorliegenden Erfindung sorgt ein programmierbarer Schnittstellenkern (PIC) für die Kommunikation zwischen einer zentralen Verarbeitungseinheit (CPU) und anderen FPGA-in-

ternen und FPGA-externen Vorrichtungen. Aufgrund der oben beschriebenen, von der vorliegenden Erfindung bestimmten Teilmenge der PLB-Funktionen kann der PIC zum effizienten Implementieren dieser Funktionen vorteilhafterweise standardmäßige FPGA-Ressourcen benutzen, d.h. Nachschlagetabellen, Multiplexer und Flip-Flops. Die vorliegende Erfindung nutzt sowohl das Pipelining als auch logische Funktionen, die sich problemlos in diesen Ressourcen abbilden lassen.

A. ZENTRALE VERARBEITUNGSEINHEIT (CPU)

[0097] Fig. 4 stellt ein FPGA **400** mit einer zentralen Verarbeitungseinheit (CPU) **401** dar, die aus nicht programmierbaren Ressourcen auf dem FPGA **400** gebildet ist. Bei einer Ausführungsform enthält die CPU **401** eine IBM PowerPC **405** CPU. Die Erfindung läßt sich jedoch in gleichem Maße auf jede beliebige CPU anwenden. Darüber hinaus kann die Erfindung mit einer beliebigen Anzahl CPUs verwendet werden. Insbesondere kann die in Fig. 4 dargestellte Architektur für jede CPU auf dem FPGA repliziert werden.

[0098] Bei dieser Ausführungsform kann die CPU **401** direkt mit mehreren chipinternen Speichern, wie beispielsweise dem Block-RAM (BRAM) **408A** und **408B**, kommunizieren. Da jede CPU ihre eigenen Ports und Verbindungen zu solchen chipinternen Speichern definiert, werden diese Einzelheiten hier nicht mit angeführt.

[0099] Die CPU **401** ist auch an einen Interrupt-Controller (INTC) **409** gekoppelt, der Markierungen abfängt und priorisiert, die von einer beliebigen Anzahl der peripheren Module, d.h. den Master-Modulen **406** oder den Slave-Modulen **404**, an die CPU **401** gesendet werden. Somit kann jedes Slave-Modul **404** oder Master-Modul **406** an den Interrupt-Controller **409** gekoppelt werden (Leitungen sind nicht gezeigt). Diese Markierungen stehen für Ereignisse, die mit den peripheren Modulen in Verbindung stehen und die sofortige Aufmerksamkeit der CPU **401** erfordern. Auf diese Weise wird die CPU **401** von der Durchführung ständiger Überprüfungen der peripheren Module befreit, was sicherstellt, dass die CPU **401** so effizient wie möglich arbeiten kann. Eine Markierung von einem peripheren Modul könnte beispielsweise die CPU **401** darüber in Kenntnis setzen, daß ein Anwender die „Eingabe"-Taste auf einer Tastatur gedrückt hat. Fachleuten ist allgemein bekannt, wie diese Markierungen erzeugt werden, und daher wird dies hier nicht ausführlich beschrieben. Für die Reaktion auf die von einem peripheren Modul erzeugten Interrupts benutzt die CPU **401** den PIC **402** und den OPB **403** (die beide nachfolgend ausführlich beschrieben werden).

[0100] Es sei angemerkt, daß die CPU **401** eine Struktur enthalten kann, die das Koppeln der fest zugeordneten Pinbelegung der CPU **401** mit der FPGA-Struktur erleichtert. Die vorliegende Erfindung kann mit zahlreichen CPUs mit unterschiedlichem Aufbau arbeiten.

[0101] Das FPGA **400** enthält des weiteren mehrere (nicht gezeigte) Datenwege, die den Transfer von Daten von mehreren Hochgeschwindigkeitsvorrichtungen **410** sowie von Master/Slave-Modulen **404**/**406** zur CPU **401** ermöglichen. Daher sollen die in Fig. 4 gezeigten Wege Transaktionswege gemäß der vorliegenden Erfindung angeben.

B. PROGRAMMIERBARER SCHNITTSTELLENKERN (PROGRAMMABLE INTERFACE CORE = PIC)

[0102] Gemäß der vorliegenden Erfindung benutzt die CPU **401** für das Bereitstellen von Hochleistungskommunikation mit anderen Vorrichtungen (sowohl auf dem FPGA **400** als auch chipextern) einen programmierbaren Schnittstellenkern (PIC) **402**. Der programmierbare Schnittstellenkern **402** wird, wie der Name vermuten läßt, mit Hilfe der von einem FPGA-Software-Tool (siehe FPGA-Software-Tool **306** in Fig. 3) erzeugten Konfigurationsbits auf dem FPGA **400** implementiert. Bei einer optimierten Ausführungsform enthält der programmierbare Schnittstellenkern **402** einen Koppelpunkt-Schalter, der es jeder Master-Vorrichtung (wie beispielsweise der CPU **401** oder der Hochgeschwindigkeits-Busschnittstelle (HSBI) **407**) gezielt ermöglicht, mit einer beliebigen Slave-Vorrichtung **410** zu kommunizieren. Darüber hinaus kann bei einem Koppelpunkt-Schalter eine weitere Master-Vorrichtung gleichzeitig mit einer weiteren Slave-Vorrichtung kommunizieren, während die erste Transaktion abläuft. Auf diese Weise kann die vorliegende Erfindung eine Hochleistungsschnittstelle zwischen Master-Vorrichtungen und Slave-Vorrichtungen sowohl auf dem FPGA **400** als auch chipextern bereitstellen.

[0103] Bei der in Fig. 4 gezeigten Ausführungsform würde der programmierbare Schnittstellenkern **402** einen 4×4-Koppelpunkt-Schalter mit 4 Eingängen (Master) × 4 Ausgängen (Slave) für das Verbinden der Master-Vorrichtungen mit den Slave-Vorrichtungen verwenden. Insbesondere können die Master-Vorrichtungen die Befehls-Cache-Speichereinheit (Instruction Cache Unit = ICU) der CPU **401**, die Daten-Cache-Speichereinheit (Data Cache Unit = DCU) der CPU **401**, die Hochgeschwindigkeits-Busschnittstelle (High Speed Bus Interface

= HSBI) **407** und ein OPB-Bridge-in-Modul (OPB/BI) enthalten, während Slave-Vorrichtungen eine DDR-SDRAM-Controller-Vorrichtung **410A**, einen der chipexternen Speicher **410B**–**410D**, einen chipinternen Speicher **410E** (bei einer Ausfuhrungsform einen BRAM) und ein OPB-Bridge-out-Modul (OPB/BO) enthalten können. Bei anderen Ausführungsformen mit N Master-Vorrichtungen und M Slave-Vorrichtungen kann der programmierbare Schnittstellenkern **402** als N×M-Koppelpunkt-Schalter konfiguriert sein. Es ist jedoch wichtig, daß die Implementierung eines Koppelpunkt-Schalters aufgrund seiner im Vergleich zu anderen Schalterarten erhöhten Konnektivität sorgfältig evaluiert werden muß, damit verfügbare Ressourcen auf dem FPGA **400** optimiert werden. Diese Evaluierung wird nachfolgend ausführlich beschrieben.

[0104] Fig. 5A stellt einen vereinfachten 4×4-Koppelpunkt-Schalter **500** dar, der als Bestandteil eines programmierbaren Schnittstellenkerns bereitgestellt wird, wobei der Koppelpunkt-Schalter **500** vier Master-Vorrichtungen **501A**–**501D** und vier Slave-Vorrichtungen **502A**–**502D** gezielt verbindet. Bei einer Ausfuhrungsform kann die Größe des Koppelpunkt-Schalters **400** auf einen 16×16-Schalter erweitert werden. Es sei angemerkt, daß die Logikressourcen in der Regel die Größe des Koppelpunkt-Schalters einschränken, wenngleich auch Leitwegressourcen diese Auswirkung haben können. Somit hängt die maximale Größe des Koppelpunkt-Schalters vom FPGA ab. Bei einer Ausfuhrungsform kann die maximale Größe so berechnet werden, daß sie die Größe N×M und die für die Erzeugung des PIC verwendeten Parameter berücksichtigt. Der Koppelpunkt-Schalter **500** enthält zwei separate Strukturen für die Adreßwege und die Datenwege der Transaktion. Bei einem tatsächlichen FPGA können sich diese Strukturen, die unter Bezugnahme auf die Fig. 5B (**500A**) und Fig. 5C (**500B**) beschrieben werden, auf überlappenden programmierbaren Ressourcen befinden. Bei anderen Ausführungsformen werden die bei der Adreßlogik und der Datenlogik der Transaktion verwendeten Strukturen aus sich nicht überlappenden programmierbaren Ressourcen gebildet.

[0105] Fig. 5B zeigt eine ausführlichere schematische Darstellung einer Implementierung des Koppelpunkt-Schalters in der Adreßlogik (nachfolgend: Koppelpunkt-Schalter **500A**). Bei dieser Implementierung enthält der Koppelpunkt-Schalter **500A** N Adreßdecodierer **511** (wobei N der Anzahl der Master-Vorrichtungen **501** entspricht) und M Zugriffsarbiter **512** (wobei M der Anzahl der Slave-Vorrichtungen **502** entspricht). Zu den Master-Vorrichtungen **501A**–**501D** gehören eine Daten-Cache-Speichereinheit (DCU), eine Befehls-Cache-Speichereinheit (ICU), eine Hochgeschwindigkeits-Busschnittstelle (HSBI) und ein Bridge-in-Modul für den chipinternen peripheren Bus (OPB/BI). Zu den Slave-Vorrichtungen **502A**–**502D** gehören eine DDR-Vorrichtung, ein statischer RAM (Static Random Access Memory = SRAM), ein Block-RAM (BRAM) und ein OPB-Bridge-out-Master OPB/BO.

[0106] Hinsichtlich der OPB/BI- und OPB/BO-Module ist anzumerken, daß, wenn die CPU **401** (Fig. 4) mit einem Slave-Modul **404** oder einem Master/Slave-Modul **405** (nachfolgend erläutert) kommunizieren muß, ein OPB-Bridge-out-Modul OPB/BO in dem programmierbaren Schnittstellenkern (PIC) **402** den Befehl der CPU zum OPB **403** leitet. Aus der Perspektive des PIC **402** fungiert dieses OPB/BO-Modul jedoch als Slave-Vorrichtung. Andererseits wird, wenn ein Master-Modul **406** oder ein Master/Slave-Modul **405** einen Befehl an die Vorrichtung **410** ausgibt, dieser Befehl von einem Bridge-in-Modul OPB/BI im programmierbaren Schnittstellenkern **402** empfangen. Aus der Perspektive des PIC **402** fungiert das OPB/BI daher scheinbar als Master. Deshalb können OPB/BO und OPB/BI als Brückenmodule in dem programmierbaren Schnittstellenkern **502** bezeichnet werden, die sowohl Master- als auch Slave-Funktionalität aufweisen. OPB/BO und OPB/BI werden unter Bezugnahme auf die Fig. 16 und Fig. 17 ausführlicher erläutert.

[0107] Bei dem Koppelpunkt-Schalter **500A** empfängt ein Adreßdecodierer **511** eine Adresse von seiner entsprechenden Master-Vorrichtung **501** und decodiert dann die Adresse, um zu ermitteln, auf welche Slave-Vorrichtung **502** die Master-Vorrichtung zugreifen möchte. Nach dem Decodieren leitet der Adreßdecodierer **511** die Adresse an den mit der angeforderten Slave-Vorrichtung **502** verbundenen Zugriffsarbiter **512** weiter. Es sei daran erinnert, daß möglicherweise mehrere Master-Vorrichtungen **501** auf eine Slave-Vorrichtung **502** zugreifen möchten. Deshalb wird der Zugriffsarbiter **512** dazu verwendet, unter den anfordernden Master-Vorrichtungen Prioritäten festzulegen. Zu einigen der bekannten Algorithmen, die von den Zugriffsarbitern **512** verwendet werden könnten, gehören der Rundlauf auf einer Ebene (Single-Level Round Robin), der Rundlauf auf mehreren Ebenen (Multi-Level Round Robin), „am längsten nicht verwendet" (Least Recently Used = LRU) und die auf der Priorität basierenden Ansätze (nachfolgend unter Bezugnahme auf den OPB-Busarbiter **503A** ausführlicher erläutert). Es sei angemerkt, daß zusätzliche Informationen von der Master-Vorrichtung **501**, einschließlich der Information, ob es sich bei der Transaktion um eine Lese- oder eine Schreiboperation handelt, und der Größe des Datentransfers, gemeinsam mit der decodierten Adresse zu der Slave-Vorrichtung **502** übertragen werden.

[0108] Die Verwendung des Adreßdecodierers **511** und des Zugriffsarbiters **512** bei dem Koppelpunkt-Schal-

ter **500A** ermöglicht es den Master-Vorrichtungen **501**, mit einer weiteren Transaktion zu beginnen, bevor die vorangehende Transaktion abgeschlossen worden ist. Anders ausgedrückt kann der Adreßdecodierer **511**, wenn die decodierte Adresse einem Zugriffsarbiter **512** zugeführt worden ist, mit dem Decodieren der nächsten Adresse beginnen. Diese Art Prozeß wird als „Transaktionsweiterleitung" bezeichnet. Die vorliegende Erfindung verwendet die Transaktionsweiterleitung auf vorteilhafte Weise dazu, die Leistung des Koppelpunkt-Schalters zu erhöhen.

[0109] Bei einer Ausführungsform der vorliegenden Erfindung kann dieses Transaktionsweiterleitungsmerkmal noch erweitert werden, wenn mehrere Register **513** (zum Zwecke der Übersichtlichkeit ist nur eines beschriftet) in die Adreßlogik aufgenommen werden. In Fig. 5B können diese Register vor den Adreßdecodierern **511**, hinter den Adreßdecodierern **511** und vor den Zugriffsarbitern **512** und/oder hinter den Zugriffsarbitern **512** und vor den Slave-Vorrichtungen **502** angeordnet werden. Es sei angemerkt, daß sich zusätzliche Register an beliebiger Stelle in der Adreßlogik befinden können (jeder Adreßweg sollte jedoch ein Register an der gleichen Stelle haben, damit ein einheitliches Timing auf den Adreß/Steuerwegen sichergestellt werden kann). Bei der vorliegenden Erfindung können diese Register **513** für das Pipelining für die Adreß- und Steuerinformationen sorgen, was es dem anfordernden Master ermöglicht, eine weitere Transaktion auszulösen, während die andere Transaktion noch verarbeitet wird. Daher kann das Pipelining der vorliegenden Erfindung auf diese Weise die Adreß/Steuer-Gesamtbandbreite des Systems erhöhen, wodurch sich die Geschwin-digkeit der Adreßlogik für mehrfache Transaktionen erhöht.

[0110] Wenn ein Zugriffsarbiter **512** feststellt, daß eine bestimmte Slave-Vorrichtung **502** für eine Transaktion verfügbar ist, kommuniziert er mit Strukturen in der Datenlogik, damit die entsprechenden Verbindungen hergestellt werden, wie unter Bezugnahme auf Fig. 5C beschrieben wird. Fig. 5C zeigt eine ausführlichere schematische Darstellung einer Implementierung des Koppelpunkt-Schalters für die Datenlogik (nachfolgend: Koppelpunkt-Schalter **500B**). Bei dieser Implementierung enthält der Koppelpunkt-Schalter **500B** N Schreibmultiplexer **514** (wobei N der Anzahl der Slave-Vorrichtungen **502** entspricht, in die bei einer Schreiboperation geschrieben werden kann) und M Lesemultiplexer **515** (wobei M der Anzahl der Master-Vorrichtungen **501** entspricht, die bei einer Leseoperation Daten empfangen können). Die Multiplexer **514A**–**514D** und **515A**–**515D** werden von Zugriffsarbitern **512A**–**512D** gesteuert (Fig. 5B).

[0111] Ähnlich wie die Adreßlogik kann der Koppelpunkt-Schalter **500B** Pipelining aufweisen, für das die Register **513** sorgen. Bei einer Ausführungsform befinden sich diese Register **513** zwischen den Master-Vorrichtungen **501** und den Schreibmultiplexern **514** und zwischen den Slave-Vorrichtungen **502** und den Lesemultiplexern **515**. Es sei angemerkt, daß sich zusätzliche Register an beliebiger Stelle in der Datenlogik befinden können. Es sollte jedoch jeder Datenweg an der gleichen Stelle ein Register haben, damit ein einheitliches Timing auf den Datenwegen sichergestellt werden kann. Bei der vorliegenden Erfindung können diese Register **513** für das Pipelining für die übertragenen Daten sorgen, was es entweder dem anfordernden Master oder dem Slave, auf den zugegriffen wird, ermöglicht, eine weitere Lese- oder Schreibtransaktion auszulösen, während die andere Lese- oder Schreibtransaktion noch verarbeitet wird. Daher kann das Pipelining der vorliegenden Erfindung auf diese Weise ebenso die Daten-Gesamtbandbreite des Systems erhöhen, wodurch sich der Durchsatz der Datenlogik für mehrfache Transaktionen erhöht.

[0112] Die vorliegende Erfindung kann wie oben beschrieben vorteilhafterweise sowohl bei der Adreßals auch bei der Datenlogik Transaktionsweiterleitung verwenden. Pipelining ermöglicht eine Optimierung der Transaktionsweiterleitung. Insbesondere können Pipeline-Register, da eine Master-Vorrichtung nicht direkt mit einer Slave-Vorrichtung kommuniziert, an mehreren Stellen in der Adreß/Datenlogik bereitgestellt werden. Wenn eine Master-Vorrichtung eine Adresse abgesendet und von dem Slave eine Bestätigung empfangen hat, könnte die Master-Vorrichtung somit den Rest der Informationen (sowohl Steuer- als auch Dateninformationen), die mit dieser Transaktion verbunden sind, über eine Leitung leiten (Pipelining). Das Pipelining hilft dadurch, daß der Master, wenn die Datenbestätigungssignale vom Slave erst einmal bei ihm eingegangen sind, die Daten zuführen kann, ohne darauf warten zu müssen, daß der Slave die Daten tatsächlich empfangen hat. Die Master-Vorrichtung kann zu diesem Zeitpunkt eine weitere Transaktion auslösen. Das Ausmaß des Pipelining bestimmt die Anzahl der Transaktionen, die gleichzeitig aktiv sein können.

[0113] Es sei angemerkt, dass bei der Gesamtleistung des Systems auch die Latenz (d.h. die Verzögerung, die mit diesem Pipelining verbunden ist) mit berücksichtigt werden muß. Bei einer Ausführungsform können die Register beispielsweise mit Hilfe von Flip-Flops implementiert werden, die ein Taktsignal benötigen. Dieses Taktsignal kann auf der Grundlage der Zeit eingestellt werden, die die Adreßdecodierer und die Zugriffsarbiter für das Ausführen ihrer jeweiligen Funktionen benötigen. Es sei angemerkt, daß Pipelining in unmittelbarem Zusammenhang mit der Frequenz steht. Das heißt, je mehr Pipelining, desto höher ist die bereitgestellte Fre-

quenz. Das Erhöhen der Frequenz erhöht allerdings auch auf unerwünschte Weise die Latenz. Wie nachfolgend noch ausführlicher beschrieben wird, findet die vorliegende Erfindung folglich einen Mittelweg zwischen Band-breite (wie sie durch das Pipelining bereitgestellt wird) und Latenz.

[0114] Ein weiterer Vorteil besteht darin, daß die vorliegende Erfindung die in jeder Slave-Vorrichtung bereitgestellte Logik beträchtlich verringern kann. Insbesondere können Slave-Vorrichtungen, die in Verbindung mit der vorliegenden Erfindung mit Master-Vorrichtungen kommunizieren, die für das Decodieren von Adressen benötigten komplexen Schaltungen eliminieren. Folglich vereinfacht die vorliegende Erfindung die Aufgabe des Konstruierens, Implementierens und Herstellens von Slave-Vorrichtungen.

[0115] Im Gegensatz dazu würde eine Master-Vorrichtung in einem System mit gemeinsam genutztem Bus eine Adresse an alle Slave-Vorrichtungen senden. Bei diesem System würde somit jede Slave-Vorrichtung die Logik für das Decodieren der Adresse und das Ermitteln, ob die Adresse zu dieser Slave-Vorrichtung gehört, enthalten. Wenn die Adresse decodiert worden ist, würde die entsprechende Slave-Vorrichtung ein Signal an die Master-Vorrichtung zurücksenden, die wiederum den Rest der die Transaktion betreffenden Informationen an die identifizierte Slave-Vorrichtung sendet.

[0116] Wie sich der obigen Beschreibung entnehmen läßt, kann ein System mit gemeinsam genutztem Bus das Ausmaß des Pipelining, das in diesem System enthalten sein kann, in wesentlichem Maße einschränken. Wenn beispielsweise eine Master-Vorrichtung die Adresse abgesendet hat, müssen alle Slave-Vorrichtungen die Adresse decodieren, was andere, nicht vorgesehene Slave-Vorrichtungen davon abhält, mit anderen Transaktionen mit anderen Master-Vorrichtungen fortzufahren. Das Shared-Bus-System aus dem Stand der Technik stellt daher natürlich eine geringere Leistung zur Verfügung als der Koppelpunkt-Schalter der vorliegenden Erfindung. Es sei angemerkt, daß bei einem (nachfolgend beschriebenen) Hybridsystem die Anzahl der Slaves auf einem gemeinsam genutzten Bus beschränkt sein kann. Auf diese Weise läßt sich die Zeit für das Senden einer Adresse und das Warten auf eine Antwort von den Slaves auf ein Minimum beschränken. Anders ausgedrückt können die Vorrichtungen, die eine große Bandbreite benötigen, durch diejenigen ausgeglichen werden, die die zusätzlichen Latenzen aushalten, die eine Konfiguration mit gemeinsam genutztem Bus mit sich bringt, wodurch ein bestimmtes System optimiert wird.

[0117] Das Konfigurieren eines Koppelpunkt-Schalters in dem FPGA als Schaltstruktur der vorliegenden Erfindung bietet verschiedene Vorteile. Zum einen sperrt der Koppelpunkt-Schalter nicht, wodurch er für einen Eingabewert einen unverstellten Weg zum Ausgang sicherstellt. Dieser Aspekt ist bei der Pipelining-Architektur der vorliegenden Erfindung besonders vorteilhaft. Zum anderen ermöglicht der mit Hilfe von FPGA-Ressourcen implementierte Koppelpunkt-Schalter ein schnelles gezieltes Ändern einer Verbindung. Es sei angemerkt, daß das Ändern einer solchen Verbindung, und folglich das Ändern eines Weges, die Verbindungen für keinen der anderen Wege beeinflußt. Und drittens kann der Koppelpunkt-Schalter programmierbar getrimmt werden, um die auf dem FPGA verwendeten Ressourcen auf ein Minimum zu reduzieren.

[0118] Fig. 5D stellt beispielsweise eine Ausführungsform des Koppelpunkt-Schalters **500A** dar, bei der die Adreß/Steuerlogik getrimmt ist. Bei dieser Ausführungsform hat der Anwender festgelegt, daß ein Teil der Adreß/Steuerlogik nicht benutzt wird. Es sei angemerkt, daß Software dem Anwender dabei helfen kann anzugeben, welche Logik getrimmt werden kann. Der Anwender muß jedoch in der Regel die Grundparameter bereitstellen, damit die Software eine solche Festlegung treffen kann. Die vorliegende Erfindung eliminiert die ungenutzte Logik, so daß die mit dieser Logik verbundenen Ressourcen für andere Bestandteile der Struktur des Anwenders verwendet werden können. Alternativ dazu kann die Struktur des Anwenders selbst dann verkleinert werden, wenn ihre anderen Bestandteile nicht die getrimmte Logik verwenden, wobei eventuell sogar ein kleineres FPGA verwendet werden kann oder bei einem FPGA mit der gleichen Größe mehr Masters beziehungsweise Slaves implementiert werden können. In Fig. 5D wird die folgende Adreß/Steuerlogik eliminiert: ICU zu OPB/BO, HSBI zu BRAM, HSBI zu OPB/BO, OPB/BI zu BRAM und OPB/BI zu OPB/BO.

[0119] Fig. 5E stellt als weiteres Beispiel eine Ausführungsform des Koppelpunkt-Schalters **500B** dar, bei der die Datenlogik getrimmt ist. Bei dieser Ausführungsform hat der Anwender festgelegt, daß ein Teil der Datenlogik nicht benutzt wird. Die vorliegende Erfindung kann die ungenutzte Logik vorteilhafterweise eliminieren. In Fig. 5E wird die folgende Schreibdatenlogik eliminiert: HSBI zu BRAM und OPB/BO, ICU zu jeder Slave-Vorrichtung (d.h. die ICU ist eine schreibgeschützte Vorrichtung) und OPB/BI zu BRAM und OPB/BO. Außerdem wird bei dieser Ausführungsform auch die folgende Leselogik eliminiert: BRAM zu HSBI und OPB/BI sowie OPB/BO zu HSBI und OPB/BI.

[0120] Gemäß der vorliegenden Erfindung kann das Trimmen von Wegen (und damit verbundener Logik) die

restlichen Schaltungen vereinfachen. In Fig. 5E haben beispielsweise die Schreibmultiplexer **514C** und **514D** jeweils nur einen Eingang. Somit könnten diese Multiplexer beispielsweise als einfache Durchgangstransistoren implementiert werden. Andere Multiplexer, wie beispielsweise die Lesemultiplexer **515C** und **515D**, ließen sich ebenfalls vereinfachen. Das Vereinfachen von Logik kann die Geschwindigkeit und andere Leistungsparameter des Systems verbessern. Folglich kann die Möglichkeit, programmierbare Ressourcen gemäß der vorliegenden Erfindung zu trimmen, zu beträchtlichen Verbesserungen bei der Leistung und der Fläche des FPGA führen.

[0121] Bei einer Ausführungsform der Erfindung ist der Koppelpunkt-Schalter **500A**/**500B** so ausgelegt, dass die Adreß/Steuerlogik mit CLK und die Datenlogik mit CLK2X läuft. Insbesondere kann ein Zugriffsarbiter **512**, wenn er feststellt, daß eine Slave-Vorrichtung **502** für eine Transaktion zur Verfügung steht, die notwendige Logik für diese Transaktion erzeugen, indem er den entsprechenden Schreibmultiplexer **514** oder Lesemultiplexer **515** steuert. Somit kann die Schreib- beziehungsweise die Leseoperation an dieser Stelle ohne weitere Verzögerung stattfinden, da keine zusätzliche Logik oder Wegübergänge nötig sind. Daher kann bei einer Ausführungsform die Adreß/Steuerlogik mit 133 MHz und die Datenlogik mit 266 MHz laufen.

[0122] Es ist wichtig, daß der Steuerweg bei einem Dual-Clock-System mit der Hälfte der nominalen Datentaktfrequenz arbeitet und so diesem Weg bevorzugt Zeit zur Verfügung gestellt wird, damit er seine Aufgabe erfüllen kann. Des weiteren ist aufgrund des Verhältnisses von 1/2:1 die Datenlatenz beträchtlich reduziert, und die Frequenz der Datenübernagungen kann erhöht werden.

[0123] Es sei angemerkt, daß der Koppelpunkt-Schalter **500A**/**500B** zusätzliche Leitungen enthalten kann, die der Übertragung eines oder mehrerer bestimmter Signale zu einer Master-Vorrichtung **501** dienen. Diese zusätzlichen Leitungen könnten unter anderem ein Bestätigungssignal übertragen. Bei einer Ausführungsform kann die Transaktionsweiterleitung beispielsweise durch ein Bestätigungssignal ausgelöst werden, das der anfordernden Master-Vorrichtung **501** von einem Zugriffsarbiter **512** zugeführt wird. Das Pipelining der Steuersignale und -daten wird insbesondere ausgelöst, nachdem die anfordernde Master-Vorrichtung **501** das Bestätigungssignal empfangen hat.

C. MASTER- & SLAVE-MODULE

[0124] Gemäß der vorliegenden Erfindung können sowohl Master-Module **406** als auch Slave-Module **404**, die bei Systemen des Standes der Technik in der Regel als periphere Vorrichtungen vorliegen, entweder als Teil der Anwenderstruktur auf oberster Ebene oder als Kerne auf dem FPGA angeordnet werden. Bei einer Ausführungsform der vorliegenden Erfindung kann das FPGA **400** ein Hybridmodul **405** enthalten, das sowohl als Master als auch als Slave fungiert. Im speziellen kann das Master/Slave-Hybridmodul **405** Transaktionen auslösen und zu bestimmten Zeiten die Steuerung eines chipinternen peripheren Busses (OPB) **403** übernehmen, und zu anderen Zeiten kann es Transaktionen akzeptieren und dem OPB **403** Daten zuführen. Bei einer Netzwerkanwendung beispielsweise kann das Hybridmodul **405**, wenn es als Slave ein Datenpaket empfängt, das Paket mit Hilfe von DMA (Direct Memory Access = direkter Speicherzugriff) als Master in den Speicher weiterleiten. Bei einer weiteren Ausführungsform ist die Master-Funktionalität in das Slave-Modul integriert.

[0125] Da die vorliegende Erfindung eine Parametrisierung vereinfacht, kann die Anzahl der Master- und der Slave-Module vorteilhafterweise sogar dynamisch geändert werden. Folglich verwendet die vorliegende Erfindung nur die Ressourcen auf dem FPGA, die für die Implementierung der Struktur des Anwenders benötigt werden.

D. CHIPINTERNER PERIPHERER BUS (OPB)

[0126] Gemäß der vorliegenden Erfindung enthält der chipinterne periphere Bus (On-chip Peripheral Bus = OPB) **403** eine Verbindungsinfrastruktur, die die Kommunikation zwischen dem programmierbaren Schnittstellenkern **402** und den Modulen **404**/**406** vereinfacht. Der OPB **403** benutzt eine Shared-Bus-Architektur, die mehrere periphere Vorrichtungen mit geringerer Leistung unterstützt; d.h. die Module **404**/**406**. Diese Module **404**/**406** könnten beispielsweise serielle Anschlüsse und Ethernet-Verbindungen verwenden. Bei einer Ausfuhrungsform können die Module **404**/**406** der OPB-Spezifikation IBM CoreConnect™ entsprechen.

[0127] Es ist wichtig, daß der OPB **403** aus den programmierbaren Leitwegressourcen gebildet werden kann, die auf dem FPGA **400** bereitgestellt werden. Insbesondere können die Leitungen des Busses aus den programmierbaren Leitwegressourcen gebildet werden, während die Schaltlogik aus den programmierbaren Ressourcen in den CLBs gebildet werden kann. Auf diese Weise kann der OPB **403** überall auf dem FPGA **400**

positioniert werden. Darüber hinaus kann der OPB **403** so konfiguriert werden, daß er eine beliebige Anzahl Module **404**/**406** aufnehmen kann. Anders ausgedrückt stellt die vorliegende Erfindung sicher, daß, wenn der Anwender nur drei Master-Module **406** und ein Slave-Modul **404** auf dem FPGA **400** anordnet, nur Ports für diese vier Module erzeugt werden. Somit ist der OPB **403** gemäß der vorliegenden Erfindung vollständig parametrisierbar.

[0128] Der OPB **403** enthält Schaltlogik, die es einem einzelnen Master-Modul **406** ermöglicht, mit einem Slave-Modul **404** oder einem Master/Slave **405** zu kommunizieren. Insbesondere müssen aufgrund der Shared-Bus-Architektur andere Master-Module **406** oder der Master/Slave **405** bei jeder aktiven Transaktion warten, bis sie ihre Transaktionen mit ihren entsprechenden Slave-Modulen **404** beginnen können. Ein OPB-Arbiter **403A** ordnet diesen Transaktionen Prioritäten zu.

[0129] Gemäß einer Ausführungsform der vorliegenden Erfindung ist der OPB-Arbiter **403A** ein Kern, der so konfiguriert werden kann, daß er eine beliebige Art Arbitrierungsverfahren bereitstellt, und sogar so konfiguriert werden kann, daß er sich dynamisch ändert, wenn sich das FPGA **400** in Betrieb befindet. Es sei angemerkt, daß der OPB-Arbiter **403A** wie der OPB **403** vollständig parametrisierbar ist. Daher verwendet der OPB-Arbiter **403A** nur die programmierbaren Ressourcen auf dem FPGA **400**, die für die Arbitrierung einer vom Anwender bestimmten Anzahl Master-Module **406** benötigt wird. Im allgemeinen verwendet der OPB-Arbiter **403A** einen vorgegebenen Algorithmus, um festzulegen, welches Master-Modul **406** die Steuerung des OPB **403** übernehmen kann und für wie lange.

[0130] Die Arbitrierung muß die konkurrierenden Ziele Fairness und Priorität miteinander vereinen. Fairness erfordert, daß kein Master-Modul **406** den OPB **403** für sich allein beanspruchen darf. Priorität erfordert jedoch, daß unter bestimmten Umständen vorgegebene Master-Module **406** den OPB **403** häufiger benutzen müssen, um zeitkritische Ziele erreichen zu können. Zu einigen typischen Algorithmen, die von dem OPB-Arbiter **403A** benutzt werden können, gehören der Rundlauf auf einer Ebene (Single-Level Round Robin), der Rundlauf auf mehreren Ebenen (Multi-Level Round Robin), „am längsten nicht verwendet" (Least Recently Used = LRU) und die auf der Priorität basierenden Ansätze.

[0131] Beim Rundlauf auf einer Ebene (Single-Level Round Robin) wird eine kleine Zeiteinheit, d.h. ein Quantum, definiert. Alle (mit bestimmten Masters **306** verbundenen) Transaktionen werden in eine zirkuläre Warteschlange gestellt. Der OPB-Arbiter **403A** folgt der Warteschlange und vergibt die Verwendung des OPB **403** durch das Master-Modul, damit die Transaktion für einen Zeitraum von einem Quantum erfolgen kann. Eine neue Transaktion würde dann nach der letzten Transaktion in der Warteschlange hinzugefügt werden.

[0132] Wenn die Transaktion vor dem Ende des Quantums endet, gibt das Master-Modul **406** den OPB **403** frei. Wenn die Transaktion am Ende des Quantums jedoch immer noch läuft, wäre das Master-Modul **406** vorbelegt, und die Transaktion würde zum Ende der Warteschlange hinzugefügt. Es sei angemerkt, daß das Master-Modul **406** bei einer anderen Ausführungsform seine Transaktion vollenden darf. Dann kann ein OPB-Arbiter **403A** den OPB **403** der nächsten Transaktion in der Warteschlange zuordnen.

[0133] Beim Rundlauf auf mehreren Ebenen (Multi-Level Round Robin) werden mindestens zwei zirkuläre Warteschlangen gebildet. Wenn man beispielsweise davon ausgeht, daß eine erste und eine zweite Warteschlange gebildet worden sind, so würden Transaktionen, die den OPB **403** häufig benutzen, in die erste Warteschlange aufgenommen und Transaktionen, die den OPB **403** weniger häufig benutzen, in die zweite Warteschlange. Transaktionen in der zweiten Warteschlange hätten dann, wenn sie sich in der gleichen Warteschlange befinden, gleichberechtigt Zugriff auf den OPB **403**. Die Transaktionen in der zweiten Warteschlange hätten jedoch als Gruppe gleichberechtigt Zugriff auf den OPB **403** wie jede Transaktion in der ersten Warteschlange. Anders ausgedrückt bilden die Transaktionen der zweiten Warteschlange effektiv eine „Super-Transaktion", die als den Transaktionen in der ersten Warteschlange gleichwertig betrachtet wird. Somit würde für jede Runde der Transaktionen in der ersten Warteschlange eine Transaktion in der zweiten Warteschlange ausgeführt. Bei dieser Herangehensweise würde das Master-Modul **406**, wenn die Transaktion vor dem Ende des Quantums endet, den OPB **403** freigeben. Wenn die Transaktion am Ende des Quantums jedoch immer noch läuft, wäre das Master-Modul **406** vorbelegt, und die Transaktion würde zum Ende der entsprechenden Warteschlange hinzugefügt.

[0134] Bei dem Least-Recently-Used-Ansatz (LRU; am längsten nicht verwendet) wird eine beliebige Warteschlange gebildet. Der OPB-Arbiter **403A** würde zunächst der Warteschlange folgen und jede Transaktion zum Ende kommen lassen, bevor er dem nächsten Master-Modul **406** in der Warteschlange die Steuerung des OPB **403** überläßt. Wenn der OPB-Arbiter **403A** jedoch von einem Master-Modul **406** eine Anforderung für die Bus-

steuerung empfängt, das in der Warteschlange nicht an der nächsten Stelle steht, dann würde er die Steuerung des OPB **403** (nach Beendigung der laufenden Transaktion) an das Master-Modul **406** übergeben, das den OPB **403** am längsten nicht verwendet hat.

[0135] Bei einem Ansatz auf Prioritätsbasis schließlich würde der OPB-Arbiter **403A** die Bussteuerung ausschließlich auf der Grundlage der Priorität der damit verbundenen Transaktion festlegen, die von dem Master-Modul **406** ausgeführt wird. Bei diesem Ansatz würde jede Transaktion mit höherer Priorität abgeschlossen werden, bevor die Transaktionen mit niedrigerer Priorität ausgelöst werden dürften. Bei einer Ausführungsform enthält der Ansatz auf Prioritätsbasis das Erteilen von Priorität für den Master im FPGA mit dem größten Einfluß auf die Leistung.

E. HOCHGESCHWINDIGKEITS-BUSSCHNITTSTELLE

[0136] Gemäß einer Ausführungsform der vorliegenden Erfindung enthält das FPGA **400** eine Hochgeschwindigkeits-Busschnittstelle (HSBI) **407**, die über die Busse **407A** und **407B** große Datenmengen zwischen dem programmierbaren Schnittstellenkern **402** und einer (nicht gezeigten) chipexternen Vorrichtung übertragen kann. Beispielhafte chipexterne Vorrichtungen könnten Gigabit I/O, Rapid I/O, PCI-X oder andere bekannte Hochleistungsprotokolle verwenden, um mit der Schnittstelle **407** zu kommunizieren. Diese Schnittstelle **407** kann in Form eines Kerns bereitgestellt werden. Es sei angemerkt, daß die übertragenen Daten von oder zu chipexternem Speicher (wie beispielsweise einem Flash-Speicher, einem SRAM und/oder einer anderen Art Speicher) oder chipinternem Speicher (wie beispielsweise BRAM) übertragen werden können.

[0137] Bei einer Ausführungsform kann die HSBI **407** für nicht datenbezogenen Transport von den Bussen **407C** angesteuert werden. Diese Busse ermöglichen der CPU **401** oder anderen Bus-Masters einen lokalisierten Zugriff auf den internen Registersatz der HSBI **407**. Beispielsweise kann die HSBI **407** einen Interrupt für den INTC **409** erzeugt haben, um anzuzeigen, daß sie ein Datenpaket empfangen und dessen Übertragung an eine externe DDR-Speichervorrichtung beendet hat. Die CPU **401** könnte beim Antworten auf den Interrupt den Bus **407A** dazu verwenden, mit der HSBI **407** zu kommunizieren und herauszufinden, worum es bei ihrer Dienstanforderung ging. Der Vorgang der Benutzung von Bus **407A** bedeutet jedoch, daß laufende Datentransportoperationen unterbrochen werden müßten. Dementsprechend gibt es bei einer Ausführungsform sowohl für Master- als auch für Slave-Operationen zum OPB **403** einen Satz tertiärer Busse **407C**. Mit Hilfe des Busses **407C** kann die CPU **401** mit der HSBI **407** kommunizieren, ohne daß sich dies auf den Datentransport auf Bus **407A** auswirkt. Des weiteren braucht die HSBI **407**, wenn sie Daten zu der Vorrichtung **404C** transportieren will, nicht erst auf den PIC **402** zuzugreifen und/oder diese Daten im Speicher zu speichern. Stattdessen könnte die HSBI **407** eine Transaktion auf dem OPB **403** auslösen und die Daten direkt übertragen, wodurch die Daten nicht mehr zweimal zwischengespeichert werden müssen und zu beträchtlichen Leistungsverbesserungen im gesamten System beigetragen wird.

V. FPGA-IMPLEMENTIERUNG: EINZELHEITEN

A. PIC & VORRICHTUNGEN: PORTS, TIMING & AUSFÜHRUNGSFORM

[0138] Master-Vorrichtungen und Slave-Vorrichtungen verwenden eine generische Schnittstelle, die einen stärker modular geprägten Aufbau ermöglicht. Insbesondere benutzen diese Module generische Ports (nachfolgend ausführlich erläutert), die in den programmierbaren Schnittstellenkern „eingesteckt" werden können. Bei der hier angeführten Beschreibung gehört zu dem Begriff „Ports" eine Gruppe Signale, die zwischen den Master/Slave-Vorrichtungen und dem programmierbaren Schnittstellenkern bereitgestellt werden. Aufgrund dieser generischen Ports kann eine beliebige Anzahl Master-Vorrichtungen und Slave-Vorrichtungen in beliebiger Kombination in die vorliegende Erfindung einbezogen werden. Bei einer Ausführungsform sind N Master-Vorrichtungen und M Slave-Vorrichtungen an den programmierbaren Schnittstellenkern gekoppelt, wobei N und M ganze Zahlen (in manchen Fällen die gleiche ganze Zahl) gleich oder größer als Eins sind.

[0139] Es sei darauf hingewiesen, daß sich die vorliegende Erfindung der Bedeutung der Ausgewogenheit zwischen komplexen Protokollen, die mehr Funktionen bereitstellen, und einfachen Protokollen, die schneller laufen können, bewußt ist. Bei der hier ausführlich beschriebenen Ausführungsform wird ein relativ einfaches Protokoll verwendet, wodurch trotzdem die meisten Aufgaben erfüllt werden, das aber auch sehr schnell läuft.

1. KOMMUNIKATION ZWISCHEN MASTER-VORRICHTUNG UND PIC

[0140] Fig. 6 stellt typische Signale dar, die zwischen einer beliebigen Master-Vorrichtung und dem program-

mierbaren Schnittstellenkern übertragen werden können. Um diesen Transfer zu vereinfachen, enthält die Master-Vorrichtung eine Schnittstelle **600A** mit Adreß-Ports **601A**, Schreibdaten-Ports **602A** und Lesedaten-Ports **603A**. Bei einer ähnlichen Konfiguration enthält der programmierbare Schnittstellenkern eine PIC-Schnittstelle **600B** mit Adreß-Ports **601B**, Schreibdaten-Ports **602B** und Lesedaten-Ports **603B**. Bei einer Ausfuhrungsform weist jedes zwischen Ports übertragene Signal (oder jede Gruppe von Signalen) eine damit verbundene Leitung/einen damit verbundenen Bus auf, was die effiziente Übertragung dieses Signals erleichtert. Es sei angemerkt, daß sich die Bezeichnung des Signals bei der hier angeführten Beschreibung auch auf die damit verbundene Leitung/den damit verbundenen Bus beziehen kann. Gemäß der nachfolgend beschriebenen Ausführungsform unterliegen standardmäßige PLB-Protokolle zwar Einschränkungen und bieten dadurch weniger Funktionen, das FPGA kann durch sie jedoch wesentlich schneller laufen.

a. ADRESS-PORTS FÜR MASTERS

[0141] Die Adreß-Ports **601A** können die folgenden Signale ausgeben.

M2PIC_AddrReq

[0142] Diese Statusmeldung gibt an, daß der Master einen Datentransfer anfordern möchte. Es sei angemerkt, daß andere Signale, insbesondere M2PIC_Addr[31:0], M2PIC_AddrQual und M2PIC_RdWr_n, ebenfalls gelten müssen, wenn M2PIC_AddrReq aktiv ist. Bei einer Ausfuhrungsform der vorliegenden Erfindung gibt ein Logisch-Eins-Signal (1) an, daß eine Anforderung ansteht, während ein Logisch-Null-Signal (0) angibt, daß keine Anforderung ansteht. Dieses Signal wird als Signal mit mittlerem Timing bezeichnet und besitzt eine Taktdomäne CLK.

M2PIC_Addr[0:31]

[0143] Dieses 32-Bit-Signal gibt die Adresse des angeforderten Transfers an, d.h. das spezifische Byte im Speicherraum. Wie oben erläutert muß dieses Signal im gleichen Zyklus gelten, in dem M2PIC_AddrReq gilt. Dieser Bus befindet sich auf dem kritischen Weg, da er die Adreßdecodierer für den programmierbaren Schnittstellenkern versorgt (siehe beispielsweise Adreßdecodierer **511** in Fig. 5B). Somit werden bei einer Ausführungsform tatsächlich nur die oberen Bits verglichen, so daß die unteren Bits wie bei mittlerem Timing behandelt werden können. Die oberen Bits geben insbesondere die Adresse des Slave an, während die unteren Bits den beabsichtigten Speicher im Slave angeben. Da das Angeben des Slave bei der vorliegenden Erfindung der Arbitrierung vorausgeht (siehe beispielsweise Zugriffsarbiter **512** in Fig. 5B) und zu jeder Zeit nur eine Transaktion in einem Slave aktiv sein kann, sollte das Lesen der oberen Bits als spätes Timing charakterisiert werden, während die tatsächliche Speicherstelle im Slave als mittleres Timing charakterisiert werden kann.

[0144] Es sei angemerkt, daß der angeforderte Datentransfer vorzugsweise bezüglich der Größe des Transfers adreßausgerichtet ist. Bei einer Übertragung von 8 Datenwörtern sollte sich beispielsweise die Adresse auf einer 8-Datenwörter-Grenze befinden. Anders ausgedrückt könnte eine Übertragung von zwei Datenwörtern bei der logischen Adresse 0, 2, 4 usw. beginnen, während eine Übertragung von 4 Datenwörtern bei einer logischen Adresse 0, 4, 8 usw. und eine Übertragung von 8 Datenwörtern bei einer logischen Adresse 0, 8, 16 beginnen kann. (Als Ausnahme kann beim Spezifizieren eines ersten Transfers des Zieldatenwortes eine nicht ausgerichtete Adresse angegeben werden.) Eine Ausrichtung ist für Vorrichtungen wie den DDR SDRAM notwendig, da diese dafür ausgelegt sind, ausgerichtete Blocks zu übertragen. Dieses Signal besitzt eine Taktdomäne CLK.

M2PIC_RdWr_n

[0145] Dieses Signal stellt eine Lese- oder Schreibmarkierung bereit, die das Adreßanforderungssignal (M2PIC_AddrReq) begleitet. Wie oben beschrieben muß dieses Signal im gleichen Zyklus gelten, in dem M2PIC_AddrReq gilt. Bei einer Ausfuhrungsform gibt ein Logisch-Eins-Signal (1) an, daß eine Leseoperation angefordert wird, während ein Logisch-Null-Signal (0) angibt, daß eine Schreiboperation angefordert wird. Dieses Signal wird als Signal mit mittlerem Timing bezeichnet und besitzt eine Taktdomäne CLK.

M2PIC_AddrQual[16:0]

[0146] Dieses Adreßtransferkennzeichnersignal ist für eine bestimmte CPU typisch, bei der es sich bei einer Ausführungsform um den IBM PowerPC **405** handelt. Wie oben angegeben muß dieses Signal im gleichen Zyklus gelten, in dem M2PIC_AddrReq gilt. Diese Signale werden nicht von dem programmierbaren Schnittstel-

lenkern decodiert oder verarbeitet, sondern lediglich zur entsprechenden Slave-Vorrich-tung geleitet. Dieses Signal wird als Signal mit mittlerem Timing bezeichnet und besitzt eine Taktdomäne CLK. Die Bit-Definitionen für dieses Signal lauten folgendermaßen:

Bit	Bezeichnung	Beschreibung
[16]	zwischenspeicherbares Attribut	Zwischenspeicherbarkeit von Daten (wenn nicht benutzt, auf 0 setzen)
[15]	geschütztes Attribut	Schutzdaten (wenn nicht benutzt, auf 0 setzen)
[14]	anwenderdefiniertes Attribut	anwenderdefiniertes Attribut (wenn nicht benutzt, auf 0 setzen)
[13]	WriteThru	Write-Through-Attribut (wenn nicht benutzt, auf 0 setzen)
[12:11]	Priorität	Transferanforderungspriorität – Je höher die Zahl, desto wichtiger ist die Anforderung. Diese Information wird derzeit in der Slave-Arbitrierlogik nicht benutzt. (Einstellung auf 00 empfohlen)
[10:8)	Transfergröße	000 = 1 Datenwort (1-4 Byte) - 32 Bit 000 = 2 Datenwörter (1-8 Byte) - 64 Bit 001 = Burst aus 4 Datenwörtern 010 = Burst aus 8 Datenwörtern 011 = Burst aus 16 Datenwörtern 100 = Burst aus 32 Datenwörtern 101-111 = reserviert
[7: 0]	Einzelimpuls-transferbytefreigaben	Hierbei handelt es sich um die Bytefreigaben während des Transfers eines einzelnen Datenimpulses (Transfergröße = 000). Bits [7:4] sind bei einem 32-Bit-PIC-System nicht definiert.

[0147] Es sei angemerkt, daß der Anwender bei einem 32-Bit-System möglicherweise den 32-Datenwörter-Burst-Modus nicht unterstützen möchte, da dies Strukturen erfordern könnte, sich nicht effizient in FPGA-Grundelementen abbilden lassen.

[0148] Bytefreigaben werden für Einzelimpulsübertragungen (d.h. ein oder zwei Datenwörter) bereitgestellt, damit Vorrichtungen wie die CPU Leseoperationen von geringerem Ausmaß, wie beispielsweise von einem Byte oder einem halben Datenwort, durchführen können. Diese Funktion ist beispielsweise bei der OPB-Brücke (unter Bezugnahme auf die Fig. 16 und Fig. 17 ausführlich erläutert) besonders wichtig, da einige OPB-Vorrichtungen (insbesondere die Slaves) möglicherweise nur Register mit Bytebreite aufweisen. Bei Speichervorrichtungen wie DRAM oder SRAM gibt es kein Problem beim Einlesen eines Datenwortes oder doppelten Datenwortes, selbst wenn nur ein Byte notwendig ist, und daher können diese Vorrichtungen die Einzelimpulstransferbytefreigabe ignorieren. Diese Bytefreigabebits gelten auch für Schreibvorgänge, selbst wenn die Informationen für die Datenwegbytefreigaben redundant sind.

[0149] Der Adreß-Port **601A** kann auch die folgenden Signale empfangen.

PIC2M_AddrAck

[0150] Dieses Strobe-Signal (aus dem progammierbaren Schnittstellenkern) bestätigt die aktuelle Adresse und ruft die nächste Adreßanforderung ab, sofern es eine gibt. In dem Zyklus nach der Überprüfung von PIC2M_AddrAck werden M2PIC_AddrReq, M2PIC_Addr, M2PIC_RdWr_n und M2PIC_AddrQual aktualisiert. Dieses Signal wird nur für einen CLK-Zyklus pro Adreßanforderung überprüft. Dieses Signal wird als Signal mit

spätem Timing bezeichnet und besitzt eine Taktdomäne CLK.

PIC2M_Busy

[0151] Dieses Signal teilt dem anfordernden Master mit, daß eine seiner Transaktionen in einem Slave abläuft. Dieses Signal geht auf High, wenn der programmierbare Schnittstellenkern sein PIC2M_AddrAck ausgibt, und zurück auf Low, wenn ein Slave sein Signal S2PIC_RdXferDone oder S2PIC_WrXferDone (nachfolgend ausführlich erläutert) überprüft hat, um anzu-geben, daß der Datentransfer vom Slave abgeschlossen worden ist. Bei einer Ausführungsform werden das Signal PIC2M_Busy und das Signal M2PIC_AddrReq vom Master einem ODER-Gatter zugeführt. Auf diese Weise kann der Master feststellen, wann er all seine Transaktionen über den programmierbaren Schnittstellenkern abgeschlossen hat. Dieses Signal unterstützt den Befehl „Sync" oder „EIEIO" (Enforce Instruction Execution In Order = Befehlsausführung in Reihenfolge durchsetzen) in der CPU. Bei einer Ausführungsform zeigt eine logische 1 an, daß der programmierbare Schnittstellenkern mit einem Datentransfer beschäftigt ist, und eine logische 0 zeigt an, daß es keine laufenden Transaktionen gibt. Es sei angemerkt, daß ein Slave das Signal PIC2M_Busy aktiv lassen kann, nachdem er S2PIC_RdXferDone/S2PIC_WrXferDone überprüft hat, wenn er die Transaktion nicht intern abgeschlossen hat. Dieses Signal wird als Signal mit mittlerem Timing bezeichnet und besitzt eine Taktdomäne CLK.

PIC2M_Err

[0152] Dieses Signal teilt einem Master mit, daß eine seiner laufenden Transaktionen einen Fehler verursacht hat. Bei Lesefehlern wäre dieses Signal mit PIC2M_RdDataAck (nachfolgend erläutert) aktiv. Bei Schreibfehlern kann dieses Signal während oder nach der Transaktion über den programmierbaren Schnittstellenkern überprüft werden. Insbesondere kann das Fehlersignal nur dann überprüft werden, wenn eine Schreibtransaktion über den programmierbaren Schnittstellenkern abgeschlossen worden ist, solange das Signal PIC2M_Busy aktiv ist. Zu diesem Zeitpunkt kann der anfordernde Master die Fehlerregister, d.h. BEAR und BESR, des Slave untersuchen, um festzustellen, worum es sich bei dem Fehler gehandelt hat. Dieses Signal wird als Signal mit spätem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

b. SCHREIB-PORTS FÜR MASTERS

[0153] Gemäß der vorliegenden Erfindung muß sich ein Master sicher sein, daß er alle Schreibdaten, die er übertragen will, liefern kann. Es gibt keine Datenflußsteuersignale, so daß ein Master den Fluß der übertragenen Daten nicht verlangsamen oder unterbrechen kann. Eine Slave-Vorrichtung darf jedoch Wartezustände zwischen Datentransfers forcieren, indem sie Datenbestätigungssignale (d.h. PIC2M_WrDataAck) zurückhält. Angesichts dieser Einschränkungen können Master-Schreib-Ports **602A** die folgenden Signale ausgeben.

M2PIC_Data

[0154] Dieses Signal liefert die Schreibtransferdaten. Im allgemeinen stehen die ersten zu übertragenden Schreibdaten zu dem Zeitpunkt zur Verfügung, zu dem die entsprechende Adreßanforderung die Möglichkeit hat, den Slave zu erreichen. Bei einer Ausführungsform besteht eine Latenz von zwei CLK-Zyklen (oder vier CLKX2-Zyklen) für Adreßdaten, die über Pipelining durch den programmierbaren Schnittstellenkern geleitet werden. Somit müßten Schreibdaten zwei CLK-Zyklen nach der Ausgabe einer Adreßanforderung bereitstehen. Es sei darauf hingewiesen, daß bei einem speziell angepaßten System, bei dem den Slaves eine vorgegebene Mindestantwortzeit für Schreibvorgänge zur Verfügung steht, die Verfügbarkeit von Schreibdaten um diesen Betrag verzögert sein kann.

[0155] Bei einer Ausführungsform enthalten die Schreibtransferdaten 64 Bits. Bei anderen Ausführungsformen können jedoch weniger oder mehr Bits verwendet werden. Auf jeden Fall kann der Bus, der die Schreibtransferdaten mit sich führt, auf die entsprechende Breite verkleinert/vergrößert werden. Es sei angemerkt, daß dieser Bus möglicherweise eine lange Strecke über den programmierbaren Schnittstellenkern zurücklegt, und somit ist sein Timing kritisch. Dieses Signal wird daher als Signal mit spätem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

M2PIC_DataBE[7:0]

[0156] Dieses Schreibtransferdatenbytefreigabesignal weist das gleiche Timing wie das Signal M2PIC_Data auf und begleitet die Schreibdaten als Bytefreigabekennzeichner. Bei einer Ausführungsform ist ein beliebiges Bytefreigabemuster zugelassen, einschließlich verschachtelter Bits oder „alle Bits aus". So könnte beispiels-

weise ein Hochgeschwindigkeits-Master den Transfer eines 16-Datenwörter-Bursts auslösen, alle Bytefreigaben im ersten Datentransfer ausschalten und praktisch einen 14-Datenwörter-Burst in einer Operation statt in 3 Operationen durchführen (8-Datenwörter-Burst + 4-Datenwörter-Burst + 1 doppeltes Datenwort). Dieses Signal wäre in einem 32-Bit-PIC-System 4 Bits breit. Dieses Signal wird daher als Signal mit spätem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

M2PIC_DataQual[N:0]

[0157] Dieses Signal ermöglicht das Übertragen zusätzlicher Informationen mit den Schreibdaten und/oder Schreibbytefreigaben. Bei einer Ausführungsform könnten beispielsweise Paritäts- oder Fehlerkorrektur-Codebits mit den Schreibdaten übertragen werden. Es sei angemerkt, daß sich jedes Signal M2PIC_DataQual mit den einzelnen Daten ändern kann. Dieses Signal wird als Signal mit spätem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

PIC2M_WrDataAck[7:0]

[0158] Dieses Strobe-Signal bestätigt die Schreibdaten und läßt zu, daß der nächste Teil Daten übertragen wird. In dem Zyklus nach der Überprüfung des Signals PIC2M_WrDataAck müssen die eventuell vorhandenen nächsten Daten angesteuert werden. Jedes PIC2M_WrDataAck-Bit (bei dieser Ausführungsform insgesamt 8 Bits) bestätigt die Daten für seine entsprechende Bytespur. Auf diese Weise kann ein Slave mit geringerer Breite oder ein langsamerer Slave ein paar Bytespuren gleichzeitig bestätigen. Es sei jedoch angemerkt, daß die Transaktion für alle Bytespuren bestätigt wird, obwohl das Bytefreigabebit nicht für diese Bytespur eingestellt ist. Auf diese Weise weiß der Master, daß die einzelnen Daten ordnungsgemäß geschrieben worden sind. Das dynamische Wesen des Signals PIC2M_WrDataAck ist vorteilhaft, da es ermöglicht, daß Daten bei Übertragungen nicht zusammenhängend sein müssen. Bei einer Ausführungsform werden die PIC2M_WrDataAck-Bits in der Reihenfolge ansteigender Byteadressen (d.h. von Bit [7] bis [0] oder von [0] bis [7], je nach System) durchlaufen. Dieses Signal wäre in einem 32-Bit-System 4 Bits breit. Dieses Signal wird als Signal mit spätem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

c. LESE-PORTS FÜR MASTERS

[0159] Bei einer Leseoperation muß sich eine Master-Vorrichtung sicher sein, daß sie all die Daten, deren Übertragung sie anfordert, auch empfangen kann. Es gibt keine Datenflußsteuersignale, so daß ein Master den Fluß der übertragenen Daten nicht verlangsamen oder unterbrechen kann. Eine Slave-Vorrichtung darf jedoch Wartezustände zwischen Datentransfers forcieren, indem sie Datenbestätigungssignale (d.h. PIC2M_RdDataAck) zurückhält.

[0160] Der Lese-Port **603A** kann die folgenden Signale empfangen.

PIC2M_Data

[0161] Zu diesem Signal gehören die zu übertragenden Lesedaten. Bei einer Ausführungsform gehören 64 Datenbits dazu. Bei anderen Ausführungsformen können weniger oder mehr Bits übertragen werden. Auf jeden Fall kann der diesen Signalbus übertragende Bus auf die entsprechende Größe verkleinert/vergrößert werden. Da dieser Bus möglicherweise eine lange Strecke über den programmierbaren Schnittstellenkern zurücklegt, ist sein Timing kritisch. Dieses Signal wird deshalb als Signal mit spätem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

PIC2M_DataQual

[0162] Dieses Signal betrifft Lesedatentransferkennzeichner. Bei einer Ausfuhrungsform lauten die Bit- Definitionen folgendermaßen:

Bit	Bezeichnung	Beschreibung
[4:0]	Lesedatenwortadresse	Gibt an, welches Datenwort oder doppelte Datenwort in einem Cache-Line-Transfer gesendet wird. Das Bit [0] ist in einem System mit einem 64-Bit-Datenweg immer 0.

[0163] Es sei angemerkt, daß die CPU bei einer Ausführungsform Transfers mit einer Größe von bis zu 32 Datenwörtern anfordern kann. Somit sind 5 Bits „Lesedatenwortadresse" notwendig, um die CPU zu unterstüt-

zen (logisch gesehen kann dieses Signal keine Transfers definieren, die größer als 32 Datenwörter sind). Dieses Signal wird als Signal mit spätem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

PIC2M_RdDataAck[7:0]

[0164] Dieses Lese-Strobe-Signal qualifiziert die Lesedaten, die empfangen werden. Jedes PIC2M_RdDataAck-Bit bestätigt einzeln die Daten für seine entsprechende Bytespur, wodurch ein Slave mit geringerer Breite oder ein langsamerer Slave ein paar Bytespuren gleichzeitig bestätigen kann. Der Slave stellt sicher, daß alle Bytespuren bestätigt werden, selbst wenn das Bytefreigabebit nicht für diese Bytespur eingestellt ist. Zusätzlich dazu kann der Slave die PIC2M_RdDataAck-Bits in der Reihenfolge ansteigender Byteadressen (dies könnte von Bit [7] bis [0] oder von [0] bis [7] erfolgen, je nachdem, ob es sich um ein Big-Endian- oder ein Little-Endian-System handelt) durchlaufen. Dieses Signal wäre demzufolge in einem 32-Bit-PIC-System 4 Bits breit. Dieses Signal wird als Signal mit spätem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

2. TIMING DER MASTER-VORRICHTUNG

[0165] Fig. 7 stellt ein Zeitdiagramm für eine Master-Vorrichtung dar, die eine 8-Datenwörter-Schreiboperation erzeugt und dann eine 8-Datenwörter-Leseoperation. Bei der Schreiboperation überprüft der Master eine Adreßanforderung (m_addr_req) zum Zeitpunkt t1. Zur gleichen Zeit gibt der Master an, daß es sich bei der Transaktion um eine Schreiboperation handelt (m_addr_r_wn geht auf Low über), und sendet die Schreibadresse (m_addr[31:0]) sowie die Größe des Transfers und sämtliche anderen Kennzeichner (m_addr_qual[]).

[0166] Nach einer Latenz von einem Taktryklus (clk) (Latenz kann bei anderen Ausführungsformen verschieden sein) empfängt der Master zum Zeitpunkt t3 eine Adreßbestätigung (m_addr_ack) von dem programmierbaren Schnittstellenkern, die es dem Master ermöglicht, zum Zeitpunkt t5 mit einer weiteren Transaktion zu beginnen. Es sei angemerkt, daß der Master zum Zeitpunkt t1 ebenfalls die Schreibdaten (D1w) sendet, aber mit dem Senden der restlichen Schreibdaten (D2w, D3w und D4w) zu den Zeitpunkten t3, t4 und t5 (Latenz von einem DDR-Taktzyklus (clk2x)) wartet, bis er zum Zeitpunkt t3 die Adreßbestätigung (m_addr_ack) empfangen hat. (Bei dieser Ausführungsform entspricht ein Datenwort 32 Bits. Somit werden bei einem 64-Bit-Datenweg nur 4 64-Bit-Transfers benötigt.)

[0167] In Fig. 7 folgt auf die Schreiboperation sofort die Leseoperation. Deshalb bleibt die Adreßanforderung (m_addr_req) bei logisch 1. Der Master muß jedoch angeben, daß es sich bei der Transaktion um eine Leseoperation handelt. Somit geht m_addr_r_wn zum Zeitpunkt t5 auf High über. Entsprechend dem Pipelining bei einer Ausführungsform der vorliegenden Erfindung kann der Master die Leseadresse (m_addr[31:0]) und den Adreßkennzeichner (m_addr_qual[]) zum Zeitpunkt t5 senden, d.h. bevor die Schreiboperation zum Zeitpunkt t7 abgeschlossen worden ist. Die Master-Vorrichtung empfängt die Lesedaten (D1r, D2r, D3r und D4r) und die Lesekennzeichner (Q1r, Q2r, Q3r und Q4r), was zum Zeitpunkt t8 beginnt und zum Zeitpunkt t12 beendet wird.

3. KOMMUNIKATION ZWISCHEN SLAVE-VORRICHTUNG UND PIC

[0168] Eine Slave-Vorrichtung empfängt Befehle zum Datenübertragen und führt die von den Master-Vorrichtungen angeforderte Lese- oder Schreiboperation durch. **Fig.** 5 stellt typische Signale dar, die zwischen einer Slave-Vorrichtung und dem programmierbaren Schnittstellenkern übertragen werden können. Um diesen Transfer zu erleichtern, enthält die Slave-Vorrichtung eine Schnittstelle **800A** mit den Adreß-Ports **801A**, den Schreibdaten-Ports **802A** und den Lesedaten-Ports **803A**. Bei einer ähnlichen Konfiguration enthält der programmierbare Schnittstellenkern eine PIC-Schnittstelle **800B** mit den Adreß-Ports **801B**, den Schreibdaten-Ports **802B** und den Lesedaten-Ports **803B**. Es sei angemerkt, daß einige Signale denen für die Master-Ports (Fig. 6) gleichen, wobei nur ihre Richtungen umgekehrt sind.

a. ADRESS-PORTS FÜR SLAVES

[0169] Die Adreß-Ports **801A** können die folgenden Signale empfangen.

PIC2S_AddrReq

[0170] Diese Statusmeldung gibt an, ob eine Adreßanforderung vom Master ansteht. Die Adresse (PIC2S_Addr) und Adreßkennzeichner (PIC2S_AddrQual, PIC2S_RdWr_n) müssen ebenso gelten, wenn PIC2S_AddrReq aktiv ist. Bei einer Ausführungsform gibt eine logische 1 an, daß eine Anforderung ansteht,

und eine logische 0 gibt an, daß keine Anforderung ansteht. Dieses Signal wird als Signal mit frühem Timing bezeichnet und besitzt eine Taktdomäne CLK.

PIC2S_RdWr_n

[0171] Dieses Signal, das das Adreßanforderungssignal begleitet, gibt an, ob es sich bei der Anforderung um eine Lese- oder eine Schreiboperation handelt. Dieses Signal muß im gleichen Zyklus gelten wie PIC2S_AddrReq. Bei einer Ausführungsform steht eine logische 1 für eine Leseoperation (READ) und eine logische 0 für eine Schreiboperation (WRITE). Dieses Signal wird als Signal mit frühem Timing bezeichnet und besitzt eine Taktdomäne CLK.

PIC2S_Addr[31:0]

[0172] Dieses Signal liefert die Adressen für den angeforderten Transfer. Insbesondere weist der Adreßwert auf ein bestimmtes Byte im Speicherraum hin, und er muß im gleichen Zyklus gelten wie PIC2S_AddrReq. Es sei angemerkt, daß die Master Adreßwerte liefern, die hinsichtlich der Größe des Datentransfers ausgerichtet sind. Dieses Signal wird als Signal mit frühem Timing bezeichnet und besitzt eine Taktdomäne CLK.

PIC2S_AddrQual[20:0]

[0173] Dieses Adreßtransferkennzeichnersignal ist für die bestimmte CPU typisch, bei der es sich bei einer Ausführungsform um den IBM PowerPC **405** handelt. Wie oben angegeben müssen diese Adreßkennzeichnerinformationen im gleichen Zyklus gelten wie das Signal PIC2S_AddrReq. Dieses Signal wird als Signal mit frühem Timing bezeichnet und besitzt eine Taktdomäne CLK. Die Bit-Definitionen lauten folgendermaßen:

Bit	Bezeichnung	Beschreibung
[20:17]	Master-ID	Dieses Signal gibt die ID-Nummer des Masters an, der momentan auf den Slave zugreift. Diese Information kann beispielsweise für das Optimieren der Leistung oder das Modifizieren des Verhaltens für bestimmte Masters nützlich sein. Der Anwender muß Kompatibilitätsprobleme berücksichtigen, wenn dieses Signal benutzt wird, weil die Lage der Masters systemspezifisch ist.
[16]	zwischenspeicherbares Attribut	Zwischenspeicherbarkeit von Daten (wenn nicht benutzt, auf 0 setzen)
[15]	geschütztes Attribut	Schutzdaten (wenn nicht benutzt, auf 0 setzen)
[14]	anwenderdefiniertes Attribut	anwenderdefiniertes Attribut (wenn nicht benutzt, auf 0 setzen)
[13]	WriteThru	Write-Through-Attribut (wenn nicht benutzt, auf 0 setzen)
[12:11]	Priorität	Dieses Signal liefert eine Transferanforderungspriorität. Bei einer Ausführungsform gilt: Je höher die Zahl, desto wichtiger ist die Anforderung.
[10:8]	Transfergröße	000 = 1 Datenwort (1-4 Byte, 32-Bit-System) 000 = 1 doppeltes Datenwort (1-8 Byte, 64-Bit-System) 001 = Burst aus 4 Datenwörtern 010 = Burst aus 8 Datenwörtern 011 = Burst aus 16 Datenwörtern 100 = Burst aus 32 Datenwörtern 101-111 = reserviert
[7:0]	Einzelimpuls-transferbytefreigaben	Hierbei handelt es sich um die Bytefreigaben während des Transfers eines einzelnen Datenimpulses (Transfergröße = 000). Bits [7:4] sind bei einem 32-Bit-System nicht definiert.

[0174] Bytefreigaben werden für Einzelimpulstransfers bereitgestellt, damit Vorrichtungen wie die CPU Lesevorgänge von geringerer Größe, wie von Bytes und halben Datenwörtern, durchführen können. Dies ist beispielsweise besonders bei der OPB-Brücke wichtig, da einige OPB-Vorrichtungen möglicherweise nur Register mit Bytebreite aufweisen. Für Speichervorrichtungen wie DRAM oder SRAM gibt es kein Problem beim Einlesen eines Datenwortes oder doppelten Datenwortes, selbst wenn nur ein Byte notwendig ist, so daß diese Vorrichtungen die Einzelimpulstransferbytefreigabe ignorieren können. Diese Bytefreigabebits gelten ebenso für Schreibvorgänge, obwohl die Informationen für die Datenwegbytefreigaben redundant sind.

[0175] Die Adreß-Ports **801A** können ebenso die folgenden Signale ausgeben.

S2PIC_AddrAck

[0176] Dieses Strobe-Signal bestätigt die Adresse und ruft die nächste Adreßanforderung ab, sofern es eine gibt. In dem Zyklus nach der Überprüfung des Signals S2PIC_AddrAck werden die Signale PIC2S_AddrReq, PIC2S_Addr, PIC2S_RdWr_n und PIC2S_AddrQual auf die nächste Adreßanforderung aktualisiert, sofern es eine gibt. Dieses Strobe-Signal wird nur für einen CLK-Zyklus pro Adreßanforderung überprüft. Dieses Signal wird als Signal mit frühem Timing bezeichnet und besitzt eine Taktdomäne CLK.

[0177] Ein Slave, der Adreß-Pipelining unterstützen möchte, kann die Adreßinformationen umgehend bestätigen, so daß der Kern gemäß der vorliegenden Erfindung die nächste Adresse abrufen kann. Ein Slave kann eine zweite Adreßanforderung erst dann bestätigen, wenn er den mit der ersten Anforderung verbundenen Datentransfer abgeschlossen hat. Anders ausgedrückt kann der Slave die Pipeline-Adreßanforderung für das Vorbereiten des nächsten Transfers verwenden, darf jedoch die zweite Adresse nicht vorzeitig bestätigen.

[0178] Eine Ausnahme von dieser allgemeinen Regel besteht darin, daß ein Slave umgehend eine zweite Anforderung bestätigen darf, wenn diese einen anderen Daten-Port verwendet. Ein Slave kann beispielsweise einen Lesetransfer bestätigen, dann einen Schreibtransfer bestätigen und sich dann eine dritte Adresse ansehen.

S2PIC_Busy[15:0]

[0179] Dieses Signal teilt dem Master mit, daß eine seiner Transaktionen in einem Slave abläuft. Der Slave ist insbesondere dafür verantwortlich, das Belegt-Signal (Busy) zu dem entsprechenden Master zurückzuleiten, der die Transaktion ausgelöst hat. Der Verbindungskern der vorliegenden Erfindung kann dieses Belegt-Signal automatisch für den Master überprüfen, bis der Slave ein Signal S2PIC_RXferDone oder S2PIC_WrXferDone ausgibt (nachfolgend erläutert). Alternativ dazu kann der Slave weiterhin das Belegt-Signal nach dem Signal S2PIC_Wr/RdXferDone senden, um dem Master mitzuteilen, daß die Transaktion im Slave noch nicht abgeschlossen ist (d.h. die Transaktion steht bei einem anderen Bus noch an oder befindet sich noch in einer Warteschlange im Slave). Dieses Signal unterstützt den Befehl „Sync" oder „EIEIO" (Enforce Instruction Execution In Order = Befehlsausführung in Reihenfolge durchsetzen) in der CPU. Bei einer Ausführungsform zeigt eine logische 1 an, daß der Slave noch mit einem Datentransfer beschäftigt ist, und eine logische 0 zeigt an, daß bei dem Slave keine laufenden Transaktionen anliegen. Dieses Signal wird als Signal mit mittlerem Timing bezeichnet und besitzt eine Taktdomäne CLK.

S2PIC_Err[15:0]

[0180] Dieses Signal teilt dem Master mit, daß eine seiner laufenden Transaktionen einen Fehler verursacht hat. Der Slave ist insbesondere dafür verantwortlich, das Fehler-Signal (Err) zu dem entsprechenden Master zurückzuleiten, der die Transaktion ausgelöst hat. Bei Lesefehlern wäre dieses Signal zusammen mit dem Signal S2PIC_RdDataAck aktiv. Bei Schreibfehlern kann dieses Signal (wenn das Signal S2PIC_Busy aktiv bleibt) während oder nach der Transaktion über den Kern jederzeit überprüft werden. Slaves, die Fehler erzeugen können, können Fehlerregister (wie beispielsweise BEAR und BESR) enthalten, die es dem Master ermöglichen, die Ursache des Fehlers zu finden. Dieses Signal wird als Signal mit spätem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

b. SCHREIB-PORTS FÜR SLAVES

[0181] Die Schreibdaten-Ports **802A** können die folgenden Signale empfangen.

PIC2S_WrDataReq

[0182] Der Slave wartet darauf, daß dieses Signal in den aktiven Zustand übergeht, bevor er beginnt anzufordern, daß die Schreibdaten vom Master übertragen werden. Da ein Master sicherstellen muß, daß er alle Daten liefern kann, die er schreiben will, ändert sich dieses Signal nicht mitten in einem Datentransfer. Wenn dieses Signal in den aktiven Zustand übergegangen ist, bleibt es auf High, bis der Slave das Signal S2PIC_WrXferDone überprüft. Ein Slave kann das Signal S2PIC_WrDataAck überprüfen und in dem gleichen Zyklus mit dem Übertragen von Daten beginnen, in dem das Signal PIC2S_WrDataReq in den aktiven Zustand übergeht. Dieses Signal wird als Signal mit frühem Timing bezeichnet und besitzt eine Taktdomäne CLK.

PIC2S_Data[63:0]

[0183] Zu diesem Signal gehören die Schreibtransferdaten. Es sei darauf hingewiesen, daß zwar 64 Bits angegeben sind, dieser Bus jedoch auf eine Breite von 32 Bit verkleinert werden kann, die in einem 32-Bit-System verwendet werden soll. Dieser Bus legt möglicherweise eine lange Strecke über den Kern zurück, und daher ist sein Timing kritisch. Dieses Signal wird als Signal mit mittlerem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

PIC2S_DataBE[7:0]

[0184] Dieses Signal weist das gleiche Timing wie das Signal PIC2S_Data auf und begleitet die Schreibdaten als Bytefreigabekennzeichner. Es sind alle Bytefreigabemuster zugelassen, einschließlich „verschachtelter" Bits oder „alle Bits aus". Es sei angemerkt, daß dieser Bus bei einem 32-Bit-System auf eine Breite von 4 Bits verkleiner werden kann. Dieser Bus legt wie die Schreibdaten möglicherweise auch eine lange Strecke über den Kern zurück, und daher ist sein Timing ebenfalls kritisch. Dieses Signal wird als Signal mit mittlerem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

PIC2S_DataQual[N:0]

[0185] Dieses Signal wird zwar in der Regel von der CPU nicht benötigt, kann jedoch für den PIC bereitgestellt werden. Eine andere CPU oder ein anderes System könnte beispielsweise von zusätzlichen Informationen profitieren, die zusammen mit den Schreibdaten und/oder Schreibbytefreigabesignalen übertragen werden. Bei einer Ausführungsform könnten zu diesen zusätzlichen Informationen Paritätsbits oder Fehlerkonekturcodes gehören. Dieses Signal wird als Signal mit mittlerem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

[0186] Die Schreibdaten-Ports **802A** können auch die folgenden Signale ausgeben.

Output: S2PIC_WrDataAck[7:0]

[0187] Dieses Strobe-Signal bestätigt die Schreibdaten und läßt zu, daß der nächste Teil Daten übertragen wird. Die Latenz zwischen dem Signal S2PIC_WrDataAck und dem nächsten verfügbaren Datenwort im Signal PIC2S_Data ist systemabhängig. Bei einer Ausführungsform könnte die Standardeinstellung eine Verzögerung von drei CLK2X-Zyklen zwischen dem Signal S2PIC_WrDataAck und den nächsten zu überprüfenden Daten sein. Jedes Bit des S2PIC_WrDataAck-Signals bestätigt einzeln die Daten für seine entsprechende Bytespur, wodurch ein Slave mit geringerer Breite oder ein langsamerer Slave ein paar Bytespuren gleichzeitig bestätigen kann. Es sei darauf hingewiesen, daß der Slave alle Bytespuren bestätigen muß, selbst wenn das Bytefreigabebit nicht für diese Bytespur eingestellt ist. Zusätzlich dazu muß der Slave die S2PIC_WrDataAck-Bits in der Reihenfolge ansteigender Byteadressen überprüfen (dies könnte von Bit [7] bis [0] oder von [0] bis [7] erfolgen, je nachdem, ob es sich um ein Big-Endian- oder ein Little-Endian-System handelt). Eine langsame Slave-Vorrichtung darf auch zwischen Überprüfungen des Signals S2PIC_WrDataAck Lücken einfügen, um die Geschwindigkeit zu steuern, mit der sie Daten überträgt. Dieses Signal wäre demzufolge in einem 32-Bit-System 4 Bits breit. Dieses Signal wird als Signal mit spätem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

S2PIC_WrXferDone

[0188] Der Slave überprüft dieses Signal, um anzuzeigen, daß er den Schreibdatenbus nicht mehr benutzt und alle Daten übertragen hat. Dieses Signal kann im letzten Zyklus des Signals S2PIC_WrDataAck, das der Slave senden muß, überprüft werden. Es sei darauf hingewiesen, daß das Signal S2PIC_WrXferDone über-

prüft werden kann, bevor die entsprechenden Schreibdaten über die Signale PIC2S_Data und PIC2S_DataBE eingetroffen sind. Dieses Signal wird als Sigal mit mittlerem Timing bezeichnet und besitzt eine Taktdomäne CLK.

c. LESE-PORTS FÜR SLAVES

[0189] Die Lese-Ports **803A** können die folgenden Signale empfangen.

PIC2S_RdDataReq

[0190] Der Slave muß darauf warten, daß dieses Signal in den aktiven Zustand übergeht, bevor er mit dem Übertragen der vom Master angeforderten Lesedaten beginnen kann. Da ein Master sicherstellen muß, daß er alle Daten empfangen kann, die er lesen will, ändert sich dieses Signal nicht mitten in einem Datentransfer. Wenn dieses Signal in den aktiven Zustand übergegangen ist, bleibt es auf High, bis der Slave das Signal S2PIC_RXferDone überprüft. Ein Slave darf das Signal S2PIC_RdDataAck überprüfen und in dem gleichen Zyklus mit dem Übertragen von Daten beginnen, in dem das Signal PIC2S_RdDataReq in den aktiven Zustand übergeht. Dieses Signal wird als Signal mit frühem Timing bezeichnet und besitzt eine Taktdomäne CLK.

[0191] Die Lesedaten-Ports **803A** können auch die folgenden Signale ausgeben.

S2PIC_Data[63:0] (CLK2X, mittel)

[0192] Zu diesem Signal gehören die übertragenen Lesedaten. Dieses Signal kann in einem 32-Bit-System auf 32 Bits verkleinert werden. Dieses Signal wird als Signal mit mittlerem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

S2PIC_DataQual[2:0]

[0193] Zu diesem Signal gehören die Lesedatentransferkennzeichner. Bei einer Ausführungsform lauten die Bit-Definitionen folgendermaßen:

Bit	Bezeichnung	Beschreibung
[4:0]	Lesedatenwortadresse	Gibt an, welches Datenwort oder doppelte Datenwort in einem Cache-Line-Transfer gesendet wird. Das Bit [0] ist in einem System mit einem 64-Bit-Datenweg immer 0.

[0194] Bei einer Ausfuhrungsform kann die CPU Transfers mit einer Größe von bis zu 32 Datenwörtern anfordern. Somit werden für die Unterstützung dieser CPU 5 Bits der „Lesedatenwortadresse" benötigt. Daraus folgt logischerweise, daß das Feld „Lesedatenwortadresse" für Transfers mit einer Größe von mehr als 32 Datenwörtern daher nicht definiert ist. Slaves, die keine ersten Transfers des Zieldatenwortes (d.h. die Möglichkeit, Datenwörter in jeder beliebigen Reihenfolge zu übertragen) unterstützen, können die Daten einfach in normaler Reihenfolge übertragen und die „Lesedatenwortadresse"-Bits entsprechend einstellen. Dieses Signal wird als Signal mit mittlerem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

S2PIC_RdDataAck[7:0]

[0195] Dieses Lese-Strobe-Signal qualifiziert die Lesedaten, die empfangen werden. Jedes S2PIC_RdDataAck-Bit bestätigt die Daten für seine entsprechende Bytespur, wodurch ein Slave mit geringerer Breite oder ein langsamerer Slave ein paar Bytespuren gleichzeitig bestätigen kann. Der Slave muß alle Bytespuren bestätigen, selbst wenn das Bytefreigabebit nicht für diese Bytespur eingestellt ist. Zusätzlich dazu muß der Slave die S2PIC_RdDataAck-Bits in der Reihenfolge ansteigender Byteadressen überprüfen (dies könnte von Bit [7] bis [0] oder von [0] bis (0] erfolgen, je nachdem, ob es sich um ein Big-Endian- oder ein Little-Endian-System handelt). Dieses Signal wäre demzufolge in einem 32-Bit-System 4 Bits breit. Dieses Signal wird als Signal mit spätem Timing bezeichnet und besitzt eine Taktdomäne CLK2X.

S2PIC_RdXferDone (CLK, mittel)

[0196] Dieses Signal wird vom Slave überprüft, um anzuzeigen, daß er den Schreibdatenbus nicht mehr benutzt und alle angeforderten Daten übertragen hat. Dieses Signal wird als Signal mit frühem Timing bezeichnet und besitzt eine Taktdomäne CLK.

4. TIMING DER SLAVE-VORRICHTUNG

[0197] Fig. 9 stellt ein Zeitdiagramm eines Slave während einer 8-Datenwörter-Schreiboperation dar. Zum Zeitpunkt t1 empfängt der Slave die Adreßanforderung (s_addr_req), die Adresse (s_addr[31:0]) und den Adreßkennzeichner (s_addr_qual[]). Zum Zeitpunkt t3 erzeugt der Slave die Adreßbestätigung (s_addr_ack), die es dem Master ermöglicht, mit einer weiteren Transaktion fortzufahren, und beginnt mit der Datenschreiboperation D1w. Es sei angemerkt, daß bei dieser Ausführungsform eine Latenz von 2 Taktzyklen (clk) vorgesehen ist, bevor der Slave zum Zeitpunkt t7 Daten D2w schreibt. Diese Latenz basiert auf dem Pipelining und kann bei anderen Ausführungsformen unterschiedlich sein.

[0198] Wenn der Slave zum Zeitpunkt t3 mit der Schreiboperation beginnt, überprüft er eine Schreibbestätigung (s_wdata_ack[0]) für den für das Schreiben der Daten benötigten Zeitraum (d.h. zwei Taktzyklen (clk)). Anders ausgedrückt fordert der Slave alle Daten an und bestätigt die Größe der Schreibtransaktion, bevor er die Daten überhaupt empfangen hat. Auf diese Weise kann der Slave, wenn die restlichen Daten (d.h. D2w-D4w) eingetroffen sind, diese Daten in drei DDR-Zyklen, d.h. zum Zeitpunkt t7–t9, schreiben. Wenn der Slave bestätigt hat, daß die Daten für die Transaktion zum Zeitpunkt t9 bei ihm eingetroffen sind (eine Latenz von einem Taktzyklus (clk)), schickt er ein Signal „Schreibtransfer beendet" (s_wxfer_done) an den Master zurück. Es sei darauf hingewiesen, daß der PIC zu den Zeitpunkten t3-t1 1 eine Schreibdatenanforderung überprüft, was dem Slave den aktiven Schreibzeitraum anzeigt.

[0199] Fig. 10 stellt ein Zeitdiagramm eines Slave während einer 8-Datenwörter-Leseoperation dar. Zum Zeitpunkt t1 empfängt der Slave die Adreßanforderung (s_addr_req), die Adresse (s_addr[31:0]) und den Adreßkennzeichner (s_addr_qual[]). Zum Zeitpunkt t3 erzeugt der Slave die Adreßbestätigung (s_addr_ack), die es dem Master ermöglicht, mit einer wieteren Transaktion fortzufahren. Es sei darauf hingewiesen, daß der Slave, da er nicht auf Schreibdaten von einem Master wartet, nach einer Latenz von 1 Taktzyklus (clk) zum Zeitpunkt t5 umgehend mit seiner Leseoperation beginnen kann.

[0200] Wenn der Slave zum Zeitpunkt t5 mit der Leseoperation beginnt, überprüft er eine Lesebestätigung (s_rdata_ack[0]) für den für das Lesen der Daten benötigten Zeitraum (d.h. zwei Taktzyklen (clk)). Nach einer Latenz von einem Taktryklus (clk) sendet der Slave ein Signal „Lesetransfer beendet" (s_rxfer_done) zum Master zurück. Es sei darauf hingewiesen, daß der Slave zu den Zeitpunkten t3–t9 eine Lesedatenanforderung überprüft, was für den Slave den aktiven Lesezeitraum anzeigt.

B. PROGRAMMIERBARER SCHNITTSTELLENKERN

[0201] Die Fig. 11A und Fig. 11B stellen eine Ausführungsform eines Adreß/Steuerweges bei einem programmierbaren Schnittstellenkern (PIC) einschließlich verschiedener, mit dem PIC verbundener Adreß/Steuersignale dar. In den Fig. 11A und Fig. 11B sind vier Master-Vorrichtungen gezeigt: eine DCU **1101A** (die Daten-Cache-Speichereinheit der CPU), eine ICU **1101B** (die Befehls-Cache-Speichereinheit der CPU), ein OPB-Slave (Bridge-in) **1101C** und eine Hochgeschwindigkeits-Busschnitt-stelle (HSBI) **1101D**. Bei dieser Ausführungsform liefert jeder Master **1101** seine Anforderungen an ein FIFO-Register **1104** (First In First Out = FIFO). Somit werden Anforderungen vom Master **1101** in die Warteschlange im FIFO **1104** eingereiht.

[0202] Schaltungen für das Aktivieren und das Schreiben in Register **1102**, die in der Master-Vorrichtung **1101A** gezeigt sind, können für die Master-Vorrichtungen **1101B**–**1101D** übernommen werden. Insbesondere kann jede Master-Vorrichtung **1101** ein Logikgatter **1102**, beispielsweise ein UND-Gatter, für das Aktivieren mehrerer Flip-Flops **1103** (die der Übersichtlichkeit halber als ein Flip-Flop gezeigt sind) enthalten. Auf der Grundlage der vom Logikgatter **1102** empfangenen Steuersignale wird dem WEN-Anschluß (Write ENable = Schreibfreigabe) des Registers **1104** das entsprechende Steuersignal zugeführt. Wenn das Register **1104** voll ist, überprüft es an seinem FULL-Anschluß (VOLL) das entsprechende logische Signal, das wiederum dem Logikgatter **1102** zugeführt wird, um den WEN des Registers **1104** zu deaktivieren.

[0203] Der Flip-Flop **1103** empfängt ebenfalls die Adresse und die Adreßkennzeichner für die Transaktion. Es sei angemerkt, daß der Flip-Flop **1103** diese Informationen mit DDR, d.h. clk2x, empfangen kann, während das FIFO **1104** diese Informationen in dem normalen FPGA-Takt, d.h. clk, empfängt. Die FIFOs **1104A**–**1104D** liefern ihre jeweiligen Adressen an mehrere Adreßvergleicher **1105A**–**1105D** im PIC **1100**. Jeder Adreßvergleicher **1105** decodiert seine eintreffende Adresse und vergleicht die decodierte Adresse mit seinen gespeicherten Adressen von Slave-Vorrichtungen in dem FPGA.

[0204] Wie bereits beschrieben wurde, identifizieren bei einer Ausführungsform nur die oberen Bits der

Adresse die Slave-Vorrichtung, während die unteren Bits die Speicheradresse in der Slave-Vorrichtung angeben. Auf diese Weise können die Adreßvergleicher **1105** schnell feststellen, welche Slave-Vorrichtung angefordert wird. Wurde ein „Hit" festgestellt, d.h. der angeforderte Slave für die Transaktion identifiziert, werden die Slave-Identifizierungsinformationen mehreren Registern **1106** zugeführt, die die Informationen wiederum zur Arbitrierlogik **1108** übertragen. Es sei angemerkt, daß die mit den Adressen verbundenen Adreßkennzeichner nicht codiert sind und deshalb nicht die Adreßvergleicher **1105** durchlaufen. Bei einer Ausführungsform werden die Adreßkennzeichner von den FIFOs **1104** direkt zu den Registern **1106** übertragen und dann zu einem Multiplexer **1107**.

[0205] Der Multiplexer **1107** wählt gezielt aus, welcher Adreßkennzeichner an einen anderen Satz Register **1109** ausgegeben wird. Es sei darauf hingewiesen, daß die Register **1109** wie die Register **1106** für das Pipelining der vorliegenden Erfindung sorgen können. Andere Ausführungsformen der vorliegenden Erfindung können weniger oder mehr Register enthalten. Der Multiplexer **1107** wird von einem Automaten **1110** gesteuert, der wiederum Eingabewerte von der zirkulären Arbitrierlogik **1108** und vom Multiplexer **1107** empfängt. (Es sei angemerkt, daß der Automat **1110**, wenn er den entsprechenden Master ausgewählt hat, erfahren muß, ob der gewählte Master eine Lese- oder eine Schreiboperation anfordert.) Der Automat **1110** befindet sich ursprünglich im Ruhezustand (IDLE), in dem er den Slave-Vorrichtungen (über die Register **1109**) mitteilt, daß keine Anforderungen anstehen. Wenn er von der Arbitrierlogik **1108** die Anforderung für die ausgewählte Adresse empfangen hat, tritt der Automat **1110** in den Adreßsendezustand ein, in dem der „Kanal" eingestellt wird. Anders ausgedrückt bestätigt der Automat **1110** die gewählte Master-Vorrichtung für die Transaktion und fährt mit dem Senden der Adreßanforderung und der Adreßkennzeichner an die entsprechende Slave-Vorrichtung fort. Wenn die Slave-Vorrichtung die Adreßanforderung empfangen hat, kann sie den Empfang bestätigen, wodurch der Automat **1110** wieder in den Ruhezustand versetzt wird. Auf diese Weise kann der Automat **1110** sich die nächste Transaktion ansehen. Es sei darauf hingewiesen, daß, wenn die nächste Transaktion den Slave betrifft, der derzeit an der aktuellen Transaktion beteiligt ist, der Slave seine Bestätigung erst nach Abschluß der derzeitigen Transaktion sendet.

[0206] Die Fig. 12A und Fig. 12B stellen eine Ausführungsform der mit einem Schreibdatenweg gemäß der vorliegenden Erfindung verbundenen Schaltungen dar. In Fig. 12A sind die gleichen vier Master-Vorrichtungen gezeigt wie in Fig. 11A: DCU **1101A**, ICU **1101B**, OPB-Slave **1101C** und Hochgeschwindigkeits-Busschnittstelle (HSBI) **1101D**. Bei dieser Ausführungsform liefert jeder Master **1101** seine Daten an ein FIFO-Register **1204** (First In First Out = FIFO). Somit werden Anforderungen vom Master **1101** in die Warteschlange im FIFO **1204** eingereiht.

[0207] Schaltungen für das Übertragen von Daten an FIFOs **1204**, die in der Master-Vorrichtung **1101A** gezeigt sind, können für die Master-Vorrichtungen **1101B**–**1101D** übernommen werden. Insbesondere kann jede Master-Vorrichtung **1101** ein Logikgatter **1202**, beispielsweise ein UND-Gatter, für das Aktivieren mehrerer Flip-Flops **1203** (die der Übersichtlichkeit halber als ein Flip-Flop gezeigt sind) enthalten. Auf der Grundlage der vom Logikgatter **1202** empfangenen Steuersignale wird dem WEN-Anschluß (Write ENable = Schreibfreigabe) des FIFO **1204** das entsprechende Steuersignal zugeführt.

[0208] Der Flip-Flop **1203** empfängt ebenfalls die Daten für die Transaktion. Es sei angemerkt, daß der Flip-Flop **1203** diese Informationen mit DDR, d.h. clk2x, empfangen kann, während das FIFO **1204** diese Informationen in dem normalen FPGA-Takt, d.h. clk, empfängt. Die FIFOs **1204A**–**1204D** liefern ihre jeweiligen Daten an mehrere Register **1205** für das Pipelining und dann an einen Multiplexer **1206**. Es sei angemerkt, daß die Implementierung von Flip-Flops im Vergleich zu FIFOs von der Anwendung abhängig sein kann. Wenn der Master beispielsweise mit der doppelten Datenrate von Schreibdaten vollständig mithalten kann, dann können Flip-Flops verwendet werden. Wenn der Master jedoch nicht vollständig mit der doppelten Datenrate mithalten kann, dann können die Schreibdaten zu einem FIFO gesendet werden, wodurch es dem Master ermöglicht wird, die Schreibdaten in seinem eigenen Tempo zu verarbeiten.

[0209] Der Multiplexer **1206** wählt gezielt aus, welche Daten an ein anderes Pipeline-Register **1210** auf dem Datenweg ausgegeben werden. Es sei angemerkt, daß bei einer vollständigen Koppelpunktimplementierung des PIC **1200** für jede Slave-Vorrichtung ein Automat **1208**, ein Multiplexer **1206** und ein Pipeline-Register **1210** repliziert werden können. Der Multiplexer **1206** wird von dem Automaten **1208** (über den Flip-Flop **1209**) gesteuert, der wiederum Eingangswerte von den Slaves empfängt, die angeben, ob dieser Datenweg derzeit benutzt wird. Wenn der Weg benutzt wird (in diesem Fall ein aktives Signal von logisch 1 besteht), dann ist der Datenweg „belegt", und der Automat **1208** gestattet keinem anderen Master, auf dem gleichen Datenweg mit einer Schreiboperation zu beginnen.

[0210] Der Automat **1208** befindet sich ursprünglich im Ruhezustand, in dem er den Slave-Vorrichtungen mitteilt, daß keine Schreibanforderungen anstehen. Es sei angemerkt, daß der Automat **1208**, wenn die Schreibdatenressource belegt ist, weil sie von einem anderen Slave benutzt wird, im Ruhezustand (IDLE) bleiben muß. Nach dem Empfang einer Schreibanforderung tritt der Automat **1208** in einen Übertragungszustand ein, wodurch der Multiplexer **1206** (der zu dem Datenweg zu einer Slave-Vorrichtung gehört) aktiviert wird und Daten sendet. Wenn die Slave-Vorrichtung die Daten empfangen hat, kann sie den Empfang bestätigen, indem sie dem Logikgatter (UND-Gatter) **1211** ein entsprechendes logisches Signal zuführt. Es sei darauf hingewiesen, daß das Logikgatter **1211** und das Pipeline-Register **1212** in der Anzahl der Slave-Vorrichtungen repliziert werden. Das Logikgatter **1211** empfängt auch ein Multiplexerauswahlsignal. Anders ausgedrückt empfängt nur eines der Logikgatter **1211** aktive Signale für sowohl das Multiplexerauswahlsignal als auch die Schreibbestätigung. Die Ausgangswerte der Register **1212** werden dann dem ODER-Gatter **1213** zugeführt, das wiederum das Schreibbestätigungssignal einem der Master **1101** (in diesem Falle der Master-Vorrichtung **1101A**) zuführt. Wenn die Schreibtransaktion abgeschlossen ist, sendet der Slave ein Signal „Schreibtransfer beendet" an den Automaten **1208**, das diesem ermöglicht, wieder in seinen Ruhezustand einzutreten.

[0211] Die Fig. 13A und Fig. 13B stellen eine Ausführungsform der mit einem Lesedatenweg gemäß der vorliegenden Erfindung verbundenen Schaltungen dar. In Fig. 13A sind die gleichen vier Master-Vorrichtungen gezeigt wie in Fig. 11A: DCU **1101A**, ICU **1101B**, OPB-Slave **1101C** und Hochgeschwindigkeits-Busschnittstelle (HSBI) **1101D**. Bei dieser Ausführungsform enthalten die Master-Vorrichtungen **1101A** und **1101B** die Flip-Flops **1301A** beziehungsweise **1301B**, die ihre Lesedaten empfangen, während die Master-Vorrichtungen **1101C** und **1101D** die FIFO-Register **1302C** beziehungsweise **1302D** enthalten, die ihre Lesedaten empfangen. Es sei angemerkt, daß die Implementierung von Flip-Flops im Vergleich zu FIFOs von der Anwendung abhängig sein kann. Wenn der Master beispielsweise mit der doppelten Datenrate von Lesedaten vollständig mithalten kann, dann können Flip-Flops verwendet werden. Wenn der Master jedoch nicht vollständig mit der doppelten Datenrate mithalten kann, dann können die Daten zu einem FIFO gesendet werden, wodurch es dem Master ermöglicht wird, die Lesedaten in seinem eigenen Tempo zu verarbeiten.

[0212] Der Automat **1303** befindet sich ursprünglich im Ruhezustand, in dem er den Slave-Vorrichtungen mitteilt, daß keine Leseanforderungen anstehen. Dieser Zustand wird von mehreren Lesemultiplexerauswahlsignalen bestimmt, die einem ODER-Gatter **1304** zugeführt werden. Handelt es sich bei einem dieser Multiplexerauswahlsignale um ein aktives H-Signal, wodurch angezeigt wird, daß der Datenweg aktiv ist, dann bleibt der Automat **1303** im Ruhezustand (IDLE). Nach dem Empfang einer Leseanforderung und der Bestätigung, daß der Datenweg nicht aktiv ist, tritt der Automat **1303** in einen Übertragungszustand ein, wodurch der Multiplexer **1307** (der zu dem Datenweg zu einer Master-Vorrichtung gehört) aktiviert wird und Daten sendet. Es sei angemerkt, daß der Flip-Flop **1306** für das Pipelining in dem Lesedatenweg sorgt. Um zwischen der Slave-Vorrichtung (nicht gezeigt) und dem Flip-Flop **1306** sowie zwischen dem Automaten **1303** und dem Flip-Flop **1306** eine im wesentlichen gleiche Verzögerung sicherzustellen, wird ein Flip-Flop **1305** in den Steuerweg zum Flip-Flop **1306** eingefügt.

[0213] Wenn die Slave-Vorrichtung die Anforderung empfangen hat, kann sie diese Anforderung bestätigen, indem sie dem Logikgatter (UND-Gatter) **1308** ein entsprechendes logisches Signal zuführt. Es sei darauf hingewiesen, daß das Logikgatter **1308** und das Pipeline-Register **1309** in der Anzahl der Slave-Vorrichtungen repliziert werden. Das Logikgatter **1308** empfängt auch ein Multiplexerauswahlsignal. Anders ausgedrückt empfängt nur eines der Logikgatter **1308** aktive Signale für sowohl das Multiplexerauswahlsignal als auch die Lesebestätigung. Die Ausgangswerte der Register **1309** werden dann dem ODER-Gatter **1310** zugeführt, das wiederum das Lesebestätigungssignal einem der Master **1101** (in diesem Falle der Master-Vorrichtung **1101A**) zuführt. Wenn die Schreibtransaktion abgeschlossen ist, sendet der Slave ein Signal „Schreibtransfer beendet" an den Automaten **1208**, das diesem ermöglicht, wieder in seinen Ruhezustand einzutreten.

[0214] Es ist wichtig und wird von den in den Fig. 11A/Fig. 11B, Fig. 12A/Fig. 12B und Fig. 13A/Fig. 13B beispielhaft dargestellt, daß die vorliegende Erfindung die Menge an Logik zwischen Registern mit Pipelining auf ein Minimum reduziert. Auf diese Weise sind sowohl die Adreß/Steuerwege als auch die Datenwege „ausgeglichen". Anders ausgedrückt sind Logik und Register mit Pipelining auf beiden Wegen gleichmäßig verteilt, wodurch sichergestellt wird, daß Verzögerungen zwischen Registern (oder Logik) im wesentlichen gleich sind. Dieses eingestellte Timing kann wiederum die Logik der Wege vereinfachen.

C. DCU/ICU: SCHNITTSTELLE ZUM PLB

[0215] Fig. 14 stellt eine Ausführungsform der Schnittstelle der Daten-Cache-Speichereinheit (DCU) (oder der Befehls-Cache-Speichereinheit (ICU)) zum PLB dar. Diese Art Schnittstelle ist im Fachgebiet bekannt und

gehört zu der bei der Struktur verwendeten CPU. Es sei darauf hingewiesen, daß die Schnittstelle **1400**, da die vorliegende Erfindung auf jede beliebige CPU anwendbar ist, in Abhängigkeit von der CPU variieren wird. In Fig. 14 ist eine Schnittstelle **1400** für die Verwendung mit der IBM PowerPC **405** CPU gezeigt.

D. BLOCK-RAM: SCHNITTSTELLE ZUM PIC

[0216] Fig. 15 stellt einen Block-RAM (BRAM) **1501** mit zwei Ports A und B dar, wobei jeder Port seine eigene Takterzeugung, Steuerung, Adresse, Schreib/Lesefunktion und Datenbreite aufweist, was unabhängige Lese/Schreibmöglichkeiten schafft. Bei dieser Ausführungsform ist der Port B mit der Anwenderlogik auf dem FPGA gekoppelt und Port A mit einer Schnittstelle **1502**. Über einen endlichen Automaten (Finite State Machine = FSM) **1503** empfängt die Schnittstelle **1502** Signale vom PIC und führt diesem verschiedene Signale zu. Der FSM **1503** wiederum steuert die Schreibfreigabe des Ports A im BRAM **1501**. Die Schnittstelle **1502** enthält des weiteren einen Zähler **1504**, der mehrere Adressen von dem PIC empfängt und auf der Grundlage ihres Zählers eine bestimmte Adresse bereitstellt. Es sei darauf hingewiesen, daß die Lade- und Freigabeeigenschaften dieses Zählers ebenfalls vom FSM **1503** gesteuert werden. Auf diese Weise bestimmt der FSM **1503**, obwohl verschiedene Adressen geladen werden können, wann diese dem BRAM **1501** zugeführt werden. Die Schreib- und Lesedatenanschlüsse von Port A sind an den PIC gekoppelt.

[0217] Der BRAM **1501** ermöglicht sowohl von Port A als auch von Port B aus einen gleichzeitigen Zugriff auf die gleiche Speicherzelle. Wenn also ein Port in eine gegebene Speicherzelle schreibt, kann der andere Port im Verlauf dieser Schreiboperation nicht die gleiche Speicherzelle anwählen. Es sei angemerkt, daß der Anwender zum Zwecke der Konfliktlösung Anwenderlogik und Software entwerfen könnte, die sich mit Problemen von Datenkohärenz und Synchronisation beschäftigt.

E. OPB-BRÜCKENMODULE

1. OPB-BRIDGE-OUT-MODUL

[0218] Das OPB/BO ist ein Bridge-out-Modul, das als Slave-Vorrichtung für den programmierbaren Schnittstellenkern (PIC) und als Master-Vorrichtung für den OPB fungiert. Fig. 16 zeigt eine schematische Darstellung einer Ausfuhrungsform eines Bridge-out-Moduls **1600** auf hoher Ebene, das eine Architektur mit Pipelining verwendet, damit es bei hohen Taktfrequenzen an der OPB-Master-Schnittstelle arbeiten kann. Bei einer Ausführungsform unterstützt das OPB/BO 64-Bit-DDR-Datenwege auf der PIC-Seite und 32-Bit-Bytefreigabetransfers auf der OPB-Seite. Das Bridge-out-Modul **1600** enthält Logik **1601** zum Decodieren von PIC-Übernagungsanforderungen und FIFOs **1602** und **1603** zum Einreihen von Transaktionen in Warteschlangen. Es können insbesondere Schreibtransaktionen mit dynamischen Bytefreigaben in den FIFOs **1602** und **1603** in Warteschlangen eingereiht werden, um die Daten aus dem PIC-Master (nicht gezeigt) zu laden, ohne daß auf den Abschluß von OPB-Transfers gewartet werden muß. Das Bridge-out-Modul **1600** enthält als Bestandteil der Steuer- und Datenweglogik des OPB-Masters **1605** auch mehrere Statusregister. Diese Statusregister können alle OPB-Fehler aufzeichnen, die aufgrund von Transaktionen entstehen, die von den PIC-Masters ausgelöst worden sind. Der OPB-Master **1605** enthält des weiteren die Logik zum ordnungsgemäßen Erzeugen von Fehler- und Belegt-Signalen, die an jeden PIC-Master zurückgehen.

[0219] Bei einer Schreiboperation mit dem Bridge-out-Modul **1600** gibt der programmierbare Schnittstellenkern (PIC) einen Schreibbefehl (eines der Slave-Steuersignale **1606**) aus, der von der Logik **1601** decodiert wird. Wenn die Decodierlogik **1601** feststellt, daß das Bridge-out-Modul **1600** nicht mit der Durchführung einer anderen Transaktion beschäftigt ist, beginnt sie mit dem Übertragen der Daten in das Schreibdaten-FIFO **1603**. Die Decodierlogik **1601** lädt auch die Adreß- und Transferkennzeichner in das Adreß-FIFO **1602**. Das Bridge-out-Modul **1600** blockiert eventuelle weitere Datentransfers, wenn die FIFOs **1602**/**1603** voll werden. Bei einer Ausführungsform ist das Schreibdaten-FIFO **1603** 64 Bits breit, kann aber in Abhängigkeit von der Größe des Transfers mit einer 32 Bit oder 64 Bit breiten Datenmenge beschrieben werden.

[0220] Eine separate Logikgruppe, die mit einem Port des OPB-Masters **1605** verbunden ist, fragt das Adreß-FIFO **1602** nach anstehenden Transaktionen ab. Wenn das Adreß-FIFO **1602** nicht leer ist, wird eine Transaktionsanforderung aus ihm ausgelesen. Die Adresse, Transfergröße und Lese/Schreibinformationen werden zu einem Automaten im OPB-Master **1605** gesendet, der den OPB-Datentransferprozeß beginnt. Die OPB-Transferlogik im OPB-Master **1605** bittet um Buszugriff zum OPB und beginnt mit dem Schreibtransfer, wenn dem OPB-Master **1605** der OPB zugesprochen wird. Wenn es sich bei dem Transfer um einen Burst (d.h. mehr als ein Vollwort) handelt, werden die entsprechenden Signale (wie beispielsweise Bussperr- und sequentielle Adreßsignale) überprüft, damit ein OPB-Schreib-Burst durchgeführt werden kann. Schreibdaten werden

aus dem Schreibdaten-FIFO **1603** übertragen, um dem OPB Daten zuzuführen. Wenn die Übertragung abgeschlossen worden ist, wird das Adreß-FIFO **1602** wieder nach zusätzlichen Transferanforderungen abgefragt. Bestätigungen oder Zeitüberschreitungen werden dem entsprechenden PIC-Master mit dem aktualisierten BEAR/BESR (nicht gezeigt) zurückgemeldet.

[0221] Bei einer Leseoperation mit dem Bridge-out-Modul **1600** wird die Lesetransferanforderung von der Logik **1601** decodiert und dann im Adreß-FIFO **1602** in eine Warteschlange eingereiht (wobei davon ausgegangen wird, daß dieser nicht voll ist). Logik im OPB-Master **1605**, die das Adreß-FIFO **1602** abfragt, erkennt die anstehende Leseanforderung und führt den Lesetransfer über den OPB aus. Soll mehr als ein Vollwort gelesen werden, dann wird eine Burst-Leseoperation durchgeführt. Die Lesedaten werden über den Flip-Flop **1604** zum PIC-Master zurückgesendet. Bestätigungen oder Zeitüberschreitungen werden wieder dem entsprechenden PIC-Master mit dem aktualisierten BEAR/BESR zurückgemeldet.

[0222] Die Logik im OPB-Master **1605** mit der Schnittstelle zum OPB ist vorzugsweise so ausgelegt, daß sie den OPB-Leistungszielen der Struktur gerecht wird. Da der OPB Einzelgang-Handshaking-Protokolle verwendet, ist es insbesondere wichtig, daß Ausgangswerte des OPB-Masters **1605** direkt aus einem Flip-Flop angesteuert werden. Anders ausgedrückt sollte bei der Analyse des ungünstigsten Falls davon ausgegangen werden, daß alle Signale spätes Timing aufweisen und somit vom Register angesteuert werden sollten, um für eine maximale Laufzeit über den OPB zu sorgen. Zusätzlich dazu durchqueren bei einer Ausführungsform die von dem OPB-Master **1605** empfangenen Signale nicht mehr als eine Logikebene, bevor sie wieder registriert werden, wodurch sich die Menge der Zeit auf ein Maximum erhöht, die den Signalen zur Verfügung steht, wenn sie sich über den OPB zu einem Slave-Modul und dann zurück zum Bridge-out-Modul **1600** ausbreiten.

[0223] Bei dem Bridge-out-Modul **1600** können das Adreß- und das Schreibdaten-FIFO **1602**/**1603** so ausgelegt sein, daß sie bei hohen Taktfrequenzen arbeiten. Bei einer Ausführungsform beruhen die FIFO-Strukturen auf Schieberegister-Nachschlagetabellen-FPGA-Grundelementen (Schieberegister-Nachschlagetabelle = Shift Register Look-up Table = SRL) und erfordern eine synchrone Beziehung zwischen den Lese- und den Schreib-Ports. Es sei darauf hingewiesen, daß zur Implementierung eines auf SRL beruhenden FIFO in der Regel ein Vorwärts/Rückwärtszähler gehört, der steuert, auf welchen Schieberegisteranschluß zugegriffen wird. Der Zähler zählt vorwärts, wenn in das FIFO geschrieben, aber nicht in ihm gelesen wird, und wenn in dem FIFO gelesen, aber nicht geschrieben wird, zählt er rückwärts. Anderenfalls wird der Zähler nicht verändert. Da sie einfach sind, unterstützen auf SRL beruhende FIFOs in der Regel keine Schutzmechanismen. Wenn der FIFO im leeren Zustand (EMPTY) gelesen oder im vollen Zustand (FULL) beschrieben wird, kann deshalb ein nicht zu behebender Fehler entstehen. Damit dies nicht geschieht, sollte die Logik **1601** so aufgebaut sein, daß sie sicherstellt, daß die FIFOs **1602**/**1603** ordnungsgemäß benutzt werden.

[0224] Das Bridge-out-Modul **1600** kann wie oben beschrieben Vollwort-Bytefreigabetransfers verwenden. Um eine höhere Leistung und geringere Fläche bereitstellen zu können, kann das Bridge-out-Modul **1600** die Funktion der „dynamischen Busabmessung" bei älteren OPB-Vorrichtungen eliminieren. Deshalb würden die mit dem Bridge-out-Modul **1600** verbundenen OPB-Module bei dieser Ausführungsform das Vollwort-Bytefreigabetransferprotokoll verwenden.

[0225] Angesichts der FPGA-Struktur kann das Bridge-out-Modul **1600** jedoch ohne weiteres individuell angepaßt werden, so daß es verschiedene Kompromisse zwischen Leistung und Logikausnutzung bietet. So kann beispielsweise das Adreß-FIFO **1602** entfernt und das Schreibdaten-FIFO **1603** vereinfacht werden, wenn abgesendete Schreiboperationen nicht unterstützt werden. Eine weitere Reduzierung der Logik kann erfolgen, indem die Logik im OPB-Master **1605**, die Fehler- und Belegt-Signale bearbeitet, vereinfacht wird. Es sei darauf hingewiesen, daß sich durch diese Modifikationen zwar möglicherweise die Leistung verringert, da Transaktionen nicht innerhalb des Bridge-out-Moduls **1600** in eine Warteschlange eingereiht werden können, sich aber dafür die Fläche wesentlich verkleinert.

[0226] Gemäß einer weiteren Ausführungsform der Erfindung werden das Adreß-FIFO **1602** und das Schreibdaten-FIFO **1603** nicht mit SRL-FIFOs konstruiert, sondern in auf Block-RAM (BRAM) beruhende FIFOs umgewandelt, um die Logikausnutzung zu verringern. Darüber hinaus weisen BRAM-FIFOs eine größere Takt-zu-Ausgangsverzögerung auf, wodurch sich potentiell die maximal erreichbare OPB-Taktfrequenz verringert. BRAM-FIFOs können auch für die Implementierung von asynchronen FIFOs verwendet werden, wodurch der OPB-Takt vorteilhafterweise vom Takt des Bridge-out-Moduls **1600** abgekoppelt werden kann.

[0227] Ist eine Abkopplung gewünscht, dann kann die Logik im OPB-Master **1605** auf einfache, unkomplizierte Weise aufgebaut werden, wenn man davon ausgehen kann, daß nicht die höchste Leistung gefragt ist. Bei

langsameren OPB-Taktfrequenzen (wie beispielsweise der Hälfte der Taktfrequenz des Bridge-out-Moduls **1600**) kann insbesondere ein großer Teil der Logik so ausgelegt werden, daß er „Multi-Cycle Paths" (mehrere Taktschritte für eine Operation) enthält oder die Nutzung von mehr Logikebenen gestattet, wodurch die Logikausnutzung der Struktur beträchtlich verringert wird.

2. OPB-BRIDGE-IN-MODUL

[0228] Das OPB/BI-Modul empfängt OPB-Transaktionsanforderungen als Slave und wandelt diese Anforderungen als Master in PIC-Transfers um. Fig. 17 zeigt eine schematische Darstellung einer Ausführungsform eines Bridge-in-Moduls **1700** auf hoher Ebene gemäß der vorliegenden Erfindung. Bei dem Bridge-in-Modul **1700** werden OPB-Schreibdaten in einem Schreibdaten-FIFO **1704** gespeichert, der die Daten zwischenspeichert, um den OPB von den Transaktionen zu entlasten. Bei einer Ausführungsform, bei der Bursts mit unterschiedlicher Länge von den PIC-Slave-Modulen nicht unterstützt werden, können die OPB-Burst-Schreibvorgänge in eine Reihe von Einzelimpuls- oder Cache-Line-Übertragungen über den PIC umgewandelt werden. Der OPB liest das Ergebnis in abgerufenen Daten, die an das PIC-Slave-Modul ausgegeben werden. Burst-Leseanforderungen führen dazu, daß das Bridge-in-Modul **1700** Datenvorabrufe von bis zu 8 Datenwörtern durchführt. Das Bridge-in-Modul **1700** kann ebenfalls die notwendigen Taktdomänen- und Busbreitenumwandlungen durchführen, um Daten zwischen den 64-Bit-DDR-Lese/Schreibdatenwegen des PIC und dem gemeinsam genutzten 32-Bit-Datenweg (keine DDR) des OPB zu transportieren.

[0229] Bei einer Schreiboperation werden Transaktionen über den OPB, die vom OPB-Bridge-in-Modul **1700** empfangen werden, in einem Schreibdaten-FIFO **1704** zwischengespeichert. Das Schreibdaten-FIFO **1704** unterstützt das Absenden von Schreibvorgängen, um zu verhindern, daß der OPB beschäftigt ist, während die Transaktion über den PIC durchgeführt wird. Die Zieladresse der Schreibtransaktionen sowie die Länge von Schreib-Bursts wird in einem separaten Schreib-anforderungs-FIFO **1705** gespeichert. Wenn die Schreibanforderung vollständig in die Warteschlange in dem Bridge-in-Modul **1700** eingereiht worden ist, ist die OPB-Transaktion abgeschlossen, und der OPB steht für weitere Transaktionen zur Verfügung. Wenn das Schreibdaten-FIFO **1703** oder das Schreibanforderungs-FIFO **1705** voll ist, lösen alle zusätzlichen Schreibanforderungen ein Wiederholungssignal aus, das von der Logik **1701** überprüft werden muß.

[0230] Ein Automat in der Adreß-Port-Steuerlogik **1706** fragt das Schreibanforderungs-FIFO **1705** ständig nach anstehenden Schreibtransaktionen ab. Wenn die Steuerlogik **1706** erkennt, daß das Schreibanforderungs-FIFO **1705** nicht leer ist, wird die Schreibtransaktion aus der Warteschlange genommen und verarbeitet. Schreibtransaktionen aus dem OPB können an einer beliebigen Datenwortgrenze beginnen und bis zu 32 Datenwörter lang sein. Somit kann eine Schreibtransaktion in eine Reihe von datenwort- oder cache-line-ausgerichteten Übertragungen umgewandelt werden, die von dem PIC unterstützt werden. Diese Umwandlung erfolgt mit Hilfe von Anforderungsausrichtlogik, die ebenfalls in der Steuerlogik **1706** bereitgestellt wird, sich die Startadresse und Länge eines Burst ansieht und die größte PIC-Transfergröße berechnet, die verwendet werden kann. Der PIC-Transfer kommt in ein Adreß-FIFO **1710**, in dem die Transaktion dann an dem PIC-(Master)-Port vorgelegt werden kann. Eventuell verbleibende Bestandteile des ursprünglichen Schreib-Burst werden schrittweise durch die Steuerlogik **1706** geschickt, bis die ganze Transaktionsreihe in das Adreß-FIFO **1710** geladen worden ist. Es sei angemerkt, daß an diesem Punkt die Schreibdaten in eine Warteschlange eingereiht sind und dem PIC zur Verfügung stehen.

[0231] Lesetransaktionen werden über einen separaten Automaten in der Steuerlogik **1706** bearbeitet, der Leseabrufe aus einem PIC-Slave steuert und die Daten über den OPB zurücksendet. Handelt es sich bei der OPB-Leseanforderung nicht um einen Burst, dann wird eine Leseanforderung für ein einzelnes Datenwort in das Adreß-FIFO **1710** geladen. Handelt es sich bei der OPB-Leseanforderung um eine Burst-Operation, dann ruft das Bridge-in-Modul **1700** bis zur nächsten 8-Datenwörter-Cache-Line-Grenze Daten ab, indem es eine Leseanforderung mit der kleinstmöglichen Größe in das Adreß-FIFO **1710** lädt.

[0232] Die von der Leseanforderung zurückkommenden Daten werden in einem Schieberegister **1709** gespeichert, das den 64-Bit-DDR-Lesedatenweg des PIC unterstützt. Die Daten in dem Schieberegister **1709** werden dann für den entsprechenden 32-Bit-OPB-Datenweg (oben ausführlich erläutert) multiplexiert, und es wird ein entsprechendes Transferbestätigungssignal überprüft. Bei einem Lese-Burst, der über die 8-Datenwörter-Grenze hinausgeht, erfolgt über den PIC eine neue 8-Datenwörter-Cache-Line-Anforderung an die nächstgelegene Speicherstelle, damit der Burst fortgesetzt werden kann. Bei einer Ausführungsform, bei der das Bridge-in-Modul **1700** keine Daten zwischenspeichert, macht jeder Lesevorgang eine neue PIC-Transaktionsanforderung erforderlich, selbst wenn die Daten von einer vorhergehenden Transaktion stammen. Es sei angemerkt, daß Lesefehler von dem PIC dazu führen können, daß das OPB-Fehlerbestätigungssignal über-

prüft wird (nicht gezeigt, aber von der Logik des Schieberegisters **1709** aufgefangen). Bei einer Ausführungsform kann die Logik **1701** eine Zeitüberschreitungs-Unterdrückung überprüfen, während sie darauf wartet, daß die Lesedaten zurückgesendet werden. Diese Eigenschaft ist sinnvoll, da es eine unbestimmte Zeit dauern kann, bis die PIC-Transaktion abgeschlossen ist.

[0233] Die Adreßdecodierlogik, die Bestandteil der Schreibsteuerlogik **1703** und **1707** ist, wird vom PIC dupliziert, so daß das Bridge-in-Modul **1700** nur Transaktionen für Adressen akzeptiert, von denen bekannt ist, daß sie auf der Seite des PIC existieren. Auf diese Weise wird verhindert, daß das Bridge-in-Modul **1700** Adreßfehler erzeugt und mit dem Bridge-out-Modul Rückkopplungsschleifen verursacht (siehe Fig. 16). Darüber hinaus kann durch diese Duplizierung auch der PIC-Verbindungsweg zwischen Bridge-in-Modul und Bridge-out-Modul entfernt werden, wodurch wertvolle Logikressourcen eingespart werden.

[0234] Das Adreß-, das Schreibanforderungs- und das Schreibdaten-FIFO **1710**/**1705**/**1704** können so ausgelegt sein, daß sie bei hohen Taktfrequenzen arbeiten. Bei einer Ausführungsform können diese FIFOs insbesondere auf Schieberegister-Nachschlagetabellen-FPGA-Grundelementen (Schieberegister-Nachschlagetabelle = Shift Register Look-up Table = SRL) beruhen, die eine synchrone Beziehung zwischen den Lese- und den Schreib-Ports erfordern. Da sie einfach sind, unterstützen auf SRL beruhende FIFOs wie oben beschrieben in der Regel keine Schutzmechanismen. Wenn der FIFO im leeren Zustand (EMPTY) gelesen oder im vollen Zustand (FULL) beschrieben wird, kann deshalb ein nicht zu behebender Fehler entstehen. Damit dies nicht geschieht, sollte die Logik **1706**/**1703**/**1707** so aufgebaut sein, daß sie sicherstellt, daß die FIFOs **1710**/**1705**/**1704** ordnungsgemäß benutzt werden.

[0235] Durch die hohe Taktfrequenz des PIC-Master-Ports im Bridge-in-Modul **1700** kann es schwierig werden, die Timing-Anforderungen der Struktur einzuhalten. Deshalb muß Logik, die in der 2X-Taktdomäne operiert oder auf den Weg mit dem einfachen Takt reagiert, sorgfältig strukturiert werden, damit maximal eine Logikebene zwischen Flip-Flops liegt. Diese strukturelle Einschränkung regt zum Einsatz effizienter Logikstrukturen und/oder bestimmter FPGA-Grundelemente an, die bei der Verbesserung des Timings helfen können.

[0236] Bei einer Ausfuhrungsform der vorliegenden Erfindung wird die Lesevorabrufgröße von 8 Datenwörtern auf 16 (oder 32) Datenwörter erhöht, wodurch die Effizienz des Bridge-in-Moduls **1700** verbessert wird, wenn lange OPB-Lese-Bursts üblich sind. Es sei angemerkt, daß es bei kurzen OPB-Burst-Lesevorgängen zu einer größeren Latenz kommen kann. Somit muß der Konstrukteur entscheiden, was die optimale Vorabrufgröße ist. Vorteilhafterweise kann die vorliegende Erfindung mit nur wenig zusätzlicher Logik so geändert werden, daß sie 16- oder 32-Datenwörter-Vorabrufe nutzen kann. Wie oben beschrieben können insbesondere die Lesedaten aus dem PIC in einem 64 Bit breiten SRL-Array gespeichert werden. Da das SRL-Array an sich 16 Register tief ist, besteht bereits genug Speicher für 32 Datenwörter der Lesedaten. In diesem Fall könnte der Anwender somit die Lesedatenmultiplexierlogik ändern und die Leseautomatenlogik so modifizieren, daß längere Leseanforderungen in das Adreß-FIFO **1710** geladen werden.

[0237] Bei einer weiteren Ausfuhrungsform der vorliegenden Erfindung kann das Schreibanforderungs-FIFO **1705** entfernt werden, um die Menge der in der Konstruktion verwendeten Logik zu reduzieren. Das Entfernen dieses FIFO würde jedoch die Schreibabsendefunktion deaktivieren. Ein Deaktivieren der Schreibabsendefunktion könnte aufgrund von Schreibtransaktionen, die von anstehenden Transaktionen über den PIC blockiert werden, potentiell die OPB-Leistung verringern.

[0238] Die Steuerlogik in dem Schreibdaten-FIFO **1704** benutzt eine große Menge redundanter Logik. Diese Redundanz reduziert Verzweigung und ermöglicht es somit dem FIFO **1704**, bis zur gewünschten DDR-Taktfrequenz zu laufen. Bei geringeren Taktfrequenzen kann ein großer Teil dieser redundanten Steuerlogik entfernt werden. Hinzu kommt, daß einige Lesedatenweg-Pipeline-Register, wie beispielsweise die Register **1711**, **1702** und **1708** in Fig. 17, ebenfalls herausgenommen werden könnten, um die Latenz und die Ressourcennutzung zu reduzieren, wenn die zeitlichen Einschränkungen verringert werden.

F. PROGRAMMIERBARE HYBRID-SCHNITTSTELLENKERNE

[0239] Bei einer weiteren Ausführungsform der vorliegenden Erfindung enthält der programmierbare Schnittstellenkern (PIC) zusätzlich zu dem oben ausführlich beschriebenen Koppelpunkt-Schalter Elemente eines gemeinsam genutzten Busses. Die Fig. 18A und Fig. 18B stellen zur Veranschaulichung für den Leser die Adreß/Steuerwege beziehungsweise die Datenwege einer Architektur mit gemeinsam genutztem Bus dar. Wie in Fig. 18A gezeigt ist, könnte ein gemeinsam genutzter Bus **1800A** für den Adreß/Steuerweg durch Bereitstellen eines Arbiters **1803** implementiert werden, der die Adreß/Steuersignale von Master-Vorrichtungen **1801**

empfängt. Wenn der Zugriffsarbiter **1803** festgelegt hat, welche Master-Vorrichtung **1801** dazu ausgewählt werden sollte, ihre Transaktion fortzusetzen, dann wird die Anforderung an alle Slave-Vorrichtungen **1802** weitergeleitet. Jede Slave-Vorrichtung **1802** decodiert die Adreß/Steuerinformationen und ermittelt, ob sie der für diese Transaktion angeforderte Slave ist. Der gemeinsam genutzte Bus **1800B** für den Datenweg könnte in Fig. 18B durch Bereitstellen eines einzigen Schreibmultiplexers **1804** implementiert werden, der die Schreibdaten von den Master-Vorrichtungen **1801** empfängt. Bei einem gemeinsam genutzten Bus empfangen alle Slave-Vorrichtungen **1802** die ausgewählten Schreibdaten, nur eine Slave-Vorrichtung **1802** schreibt die Daten jedoch tatsächlich. Bei einer Leseoperation kann eine Slave-Vorrichtung **1802** ein Logisch-Null-Signal ausgeben, es sei denn, diese Slave-Vorrichtung ist bei der Lesetransaktion aktiv. Auf diese Weise überträgt das ODER-Gatter **1805** die Leseinformationen zu den Lesemultiplexern **1806**. Nicht gezeigte Steuerschaltungen legen fest, welche Schreibdaten von dem Schreibmultiplexer **1804** ausgewählt werden, um die Schreiboperation abzuschließen, sowie welcher Lesemultiplexer **1806** ausgewählt wird, um die Leseoperation abzuschließen.

[0240] Die Fig. 19A und Fig. 19B stellen die Adreß/Steuerwege beziehungsweise die Datenwege einer Architektur mit Hybridkoppelpunkt/gemeinsam genutztem Bus gemäß einer weiteren Ausführungsform der vorliegenden Erfindung dar. Wie in Fig. 19A gezeigt enthält ein Hybridschalter **1900A** für den Adreß/Steuerweg mehrere Adreßdecodierer **1911A**–**1911D**, die jeweils Adreß/Steuerinformationen von den Master-Vorrichtungen **1901A**–**1901D** empfangen. Bei dem Hybridschalter **1900A** können drei Zugriffsarbiter **1912A**–**1912C** von Adreßdecodierern **1911A**–**1911D** decodierte Adressen empfangen. Der Zugriffsarbiter **1912C** legt die Arbitrierung für zwei Slave-Vorrichtungen **1922A** und **1922B** fest. Bei dieser Ausführungsform kann aufgrund des mit dem Zugriffsarbiter **1912C** gekoppelten, gemeinsam genutzten Busses bei einer Transaktion nur eine der Slave-Vorrichtungen **1922A** und **1922B** aktiv sein. Anders als bei den Slave-Vorrichtungen **1902A** und **1902B** müssen die Slave-Vorrichtungen **1922A** und **1922B** Decodierlogik enthalten, um feststellen zu können, welche von ihnen für die Transaktion ausgewählt worden ist. Es sei angemerkt, daß die Pipelining-Register im Shared-Bus-Abschnitt des Hybridschalters **1900A** entfernt worden sind.

[0241] Wie in Fig. 19B gezeigt enthält ein Hybridschalter **1900B** für den Datenweg mehrere Lesemultiplexer **1919**, und zwar einen für jede Master-Vorrichtung **1901**. Es sei angemerkt, daß die Slave-Vorrichtungen **1902C** und **1902D** die Ausgangswerte nicht wie bei einem Koppelpunkt-Schalter (siehe Fig. 5C) direkt den Lesemultiplexern **1901** zuführen, sondern einem Logikgatter (ODER-Gatter) **1930**. Somit kann bei dem Hybridschalter **1900B** zu jeder Zeit nur eine Slave-Vorrichtung **1902C** oder **1902D** den Master-Vorrichtungen **1901** Leseinformationen zuführen. Bei dieser Ausführungsform würde/n die inaktive/n Slave-Vorrichtung/en unter den Slave-Vorrichtungen **1902C** und **1902D** eine logische Null ausgeben. Der Hybridschalter **1900B** enthält ebenfalls mehrere Schreibmultiplexer **1914A**–**1914C**, d.h. minus die Anzahl der Slave-Vorrichtungen **1902**. Somit empfangen die Slave-Vorrichtungen **1902C** und **1902D** beide die vom Schreibmultiplexer **1914C** bereitgestellten Schreibdaten. Die ausgewählte Slave-Vorrichtung **1902** (die durch die vom Zugriffsarbiter **1912C** bereitgestellte Adresse festgelegt wird, siehe Fig. 19A) schreibt die Daten, während die nicht ausgewählte Slave-Vorrichtung **1902** die Daten ignoriert.

[0242] Es ist wichtig, daß sowohl die Master- als auch die Slave-Protokolle im Hybridschalter **1900A**/**1900B** so bleiben können, wie sie oben ausführlich für den Koppelpunkt-Schalter **500A**/**500B** beschrieben worden sind. Bei einer Ausführungsform kann jedoch ein Rearbitrierungs-Port hinzugefügt werden, um zu verhindern, daß der Shared-Bus-Abschnitt des Hybridschalters blockiert wird. Es sei darauf hingewiesen, daß der Hybridschalter **1900A**/**1900B** wie der Koppelpunkt-Schalter getrimmt werden kann, um unnötige Wege zu eliminieren. Somit könnte beispielsweise der mit der Master-Vorrichtung **1901B** (d.h. einer ICU) verbundene Schreibweg getrimmt werden.

[0243] Der Hybridschalter **1900A**/**1900B** kann die Anzahl der benutzten Ressourcen im Vergleich zu einem vollständigen Koppelpunkt-Schalter reduzieren. Insbesondere die Logikgatter, die bei dem Hybridschalter verwendet werden, benutzen weniger Ressourcen als die Multiplexer des Koppelpunkt-Schalters. Dieser Vorteil muß jedoch gegen die potentielle Verringerung der Leistung abgewogen werden. Bei einer Ausführungsform könnte die vorliegende Erfindung festlegen, daß ein erster Satz Slave-Vorrichtungen, der langsamer ist als ein zweiter Satz Slave-Vorrichtungen, so konfiguriert sein könnte, daß er den Shared-Bus-Abschnitt oder den Hybridschalter verwendet, während der zweite Satz Slave-Vorrichtungen so konfiguriert sein könnte, daß er den Koppelpunktabschnitt des Hybridschalters verwendet. Bei einer FPGA-Umgebung können Systemkonstrukteure ohne weiteres die entsprechende Mischung aus Shared-Bus-/Koppelpunkt-Schalter-Ressourcen festlegen, um die Leistung auf ein Maximum zu steigern und die Fläche auf ein Minimum zu verringern.

G. SOFTWARE-TOOLS

[0244] Gemäß der vorliegenden Erfindung wird ein Verfahren zum Erzeugen eines Konfigurationsbitstroms für eine programmierbare Logikvorrichtung (Programmable Logic Device = PLD) angegeben. Dieses Verfahren kann ohne weiteres in das unter Bezugnahme auf Fig. 3 beschriebene System integriert werden. Zu dem Verfahren der vorliegenden Erfindung gehört insbesondere das Aktivieren eines Kerngenerators, das Auswählen eines programmierbaren Schnittstellenkerns mit Hilfe des Kerngenerators und das Bereitstellen des programmierbaren Schnittstellenkerns für ein PLD-Software-Tool, das den Konfigurationsbitstrom erzeugt. Die PLD-Software erhält nicht nur den programmierbaren Schnittstellenkern der vorliegenden Erfindung, sondern kann auch eine Struktur auf oberster Ebene erhalten, die von einem Anwender bereitgestellt wird. Es ist wichtig, dass der programmierbare Schnittstellenkern gezielt mehrere Compliance-Ebenen bereitstellen kann. Der programmierbare Schnittstellenkern kann beispielsweise eine Teilmenge der Funktionen des PLB (Prozessor-Local-Bus) bereitstellen, wie weiter oben ausführlich beschrieben wird. Bei einer Ausführungsform kann der programmierbare Schnittstellenkern die Teilmenge auf die Funktionalität zurechtschneiden, die von der Struktur auf oberster Ebene benötigt wird.

[0245] Die hier erfolgten Beschreibungen der vorliegenden Erfindung dienen nur der Veranschaulichung und nicht der Eingrenzung. Verschiedene Ausführungsformen der vorliegenden Erfindung sind insbesondere weiter oben ausführlich beschrieben worden. Modifikationen an diesen Ausführungsformen werden für Fachleute offensichtlich sein. Aus diesem Grund kann der Schutzumfang der vorliegenden Erfindung nur durch die beiliegenden Ansprüche definiert werden.

Patentansprüche

1. Programmierbare Logikvorrichtung mit:
einer zentralen Verarbeitungseinheit (**401**);
einer mit der zentralen Verarbeitungseinheit (**401**) gekoppelten programmierbaren Schnittstelle (**402**), wobei die programmierbare Schnittstelle (**402**) einen Koppelpunkt-Schalter (**500**) zum Koppeln einer Vielzahl von Vorrichtungen (**404**, **406**, **410**) einschließt, wobei der Koppelpunkt-Schalter (**500**) Adreß/Steuerwege und Datenwege einschließt; und
einer ersten Vorrichtung (**404**, **406**, **410**) entweder zum Liefern von Informationen an die zentrale Verarbeitungseinheit (**401**) oder zum Empfangen von Informationen von dieser über die programmierbare Schnittstelle (**402**).

2. Programmierbare Logikvorrichtung nach Anspruch 1, wobei jede Vorrichtung aus der Vielzahl von Vorrichtungen (**404**, **406**, **410**) eines von folgendem ermöglicht: Liefern von Informationen an die zentrale Verarbeitungseinheit (**401**) über die programmierbare Schnittstelle (**402**), Empfangen von Informationen von der zentralen Verarbeitungseinheit (**401**) über die programmierbare Schnittstelle (**402**) und Kommunizieren mit einer anderen Vorrichtung (**404**, **406**, **410**) über die programmierbare Schnittstelle (**402**).

3. Programmierbare Logikvorrichtung nach Anspruch 1, wobei die Adreß/Steuerwege von einem ersten Satz von programmierbaren Ressourcen gebildet werden und die Datenwege von einem zweiten Satz von programmierbaren Ressourcen gebildet werden.

4. Programmierbare Logikvorrichtung nach Anspruch 1, wobei die Adreß/Steuerwege und die Datenwege anpaßbar sind.

5. Programmierbare Logikvorrichtung nach Anspruch 1, wobei der Koppelpunkt-Schalter (**500**) parametrisierbar ist.

6. Programmierbare Logikvorrichtung nach Anspruch 2, ferner mit einem peripheren Bus (**403**), der mit der programmierbaren Schnittstelle (**402**) gekoppelt ist; wobei ein Brückenmodul den peripheren Bus (**403**) mit der programmierbaren Schnittstelle koppelt.

7. Programmierbare Logikvorrichtung nach Anspruch 6, wobei eine erste Gruppe von Vorrichtungen (**404**, **406**) aus der Vielzahl von Vorrichtungen mit dem peripheren Bus (**403**) gekoppelt ist und mit der programmierbaren Schnittstelle (**402**) über das Brückenmodul gekoppelt ist.

8. Programmierbare Logikvorrichtung nach Anspruch 7, wobei eine zweite Gruppe von Vorrichtungen (**410**) aus der Vielzahl von Vorrichtungen nicht über das Brückenmodul mit der programmierbaren Schnittstelle

(**402**) gekoppelt ist.

9. Programmierbare Logikvorrichtung nach Anspruch 7, wobei die erste Gruppe von Vorrichtungen (**404**, **406**) eine Vielzahl von vom Anwender vorgesehenen Kernen einschließt.

10. Programmierbare Logikvorrichtung nach Anspruch 9, wobei mindestens ein Kern eine Master/Slave- oder eine Master- oder eine Slave-Funktionalität ermöglicht.

11. Programmierbare Logikvorrichtung nach Anspruch 8, ferner mit mindestens einer Speichervorrichtung, wobei die mindestens eine Speichervorrichtung Teil der zweiten Gruppe von Vorrichtungen (**410**) ist.

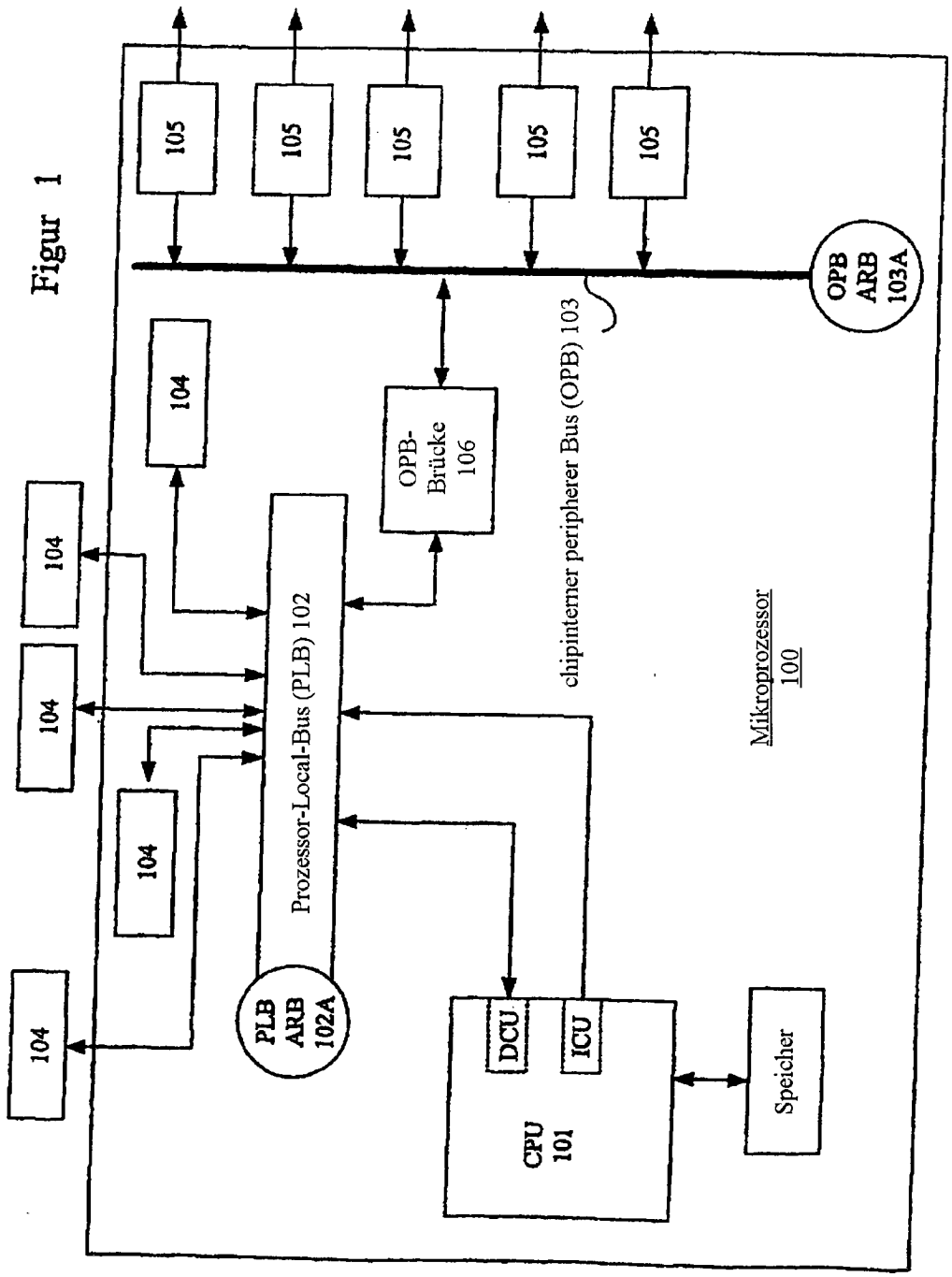
12. Programmierbare Logikvorrichtung nach Anspruch 8, ferner mit einer chipexternen Vorrichtung (**410**), wobei die chipexterne Vorrichtung Teil der zweiten Gruppe von Vorrichtungen ist; wobei die chipexterne Vorrichtung (**410**) eine Vorrichtung mit doppelter Datenrate oder eine Speichervorrichtung einschließt.

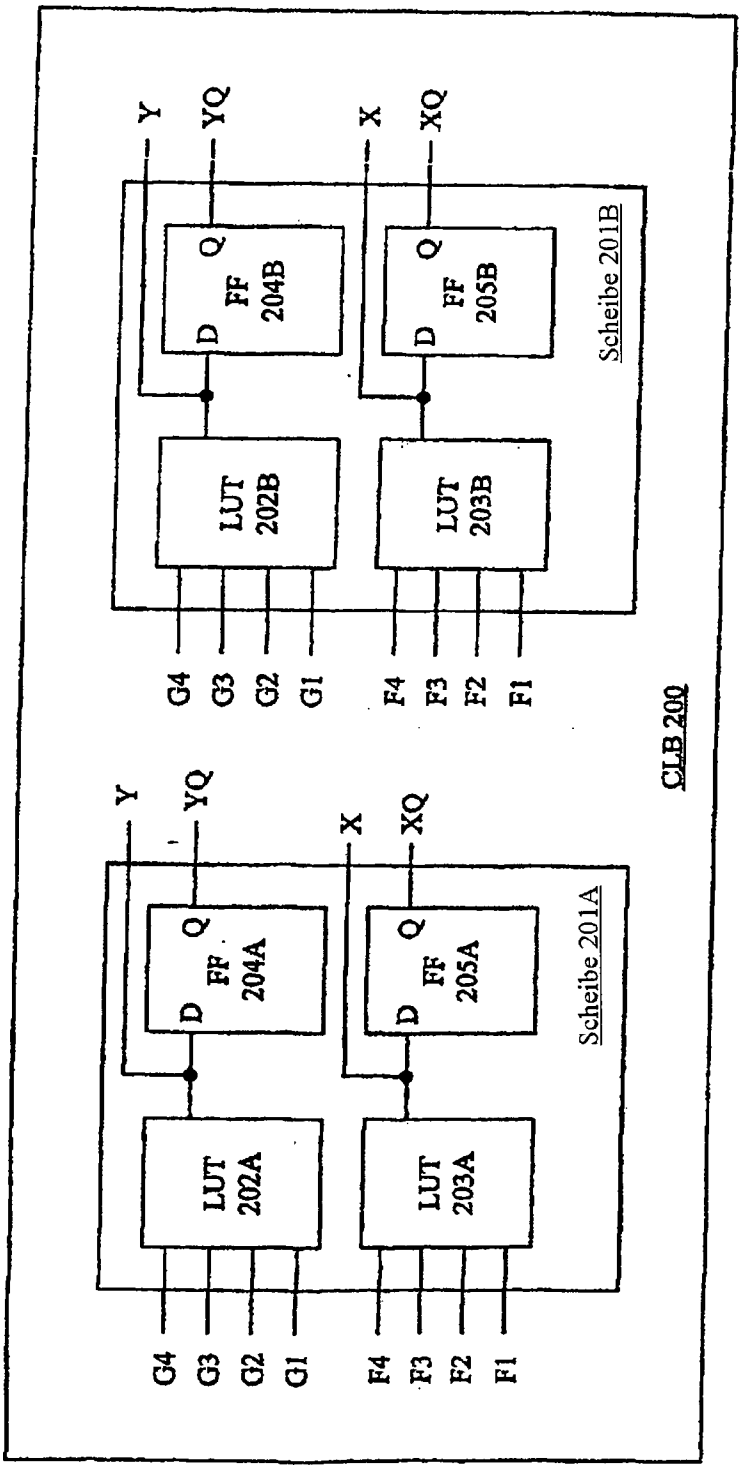
13. Programmierbare Logikvorrichtung nach Anspruch 6, ferner mit einer Hochgeschwindigkeits-Busschnittstelle (**407**), die mit der programmierbaren Schnittstelle (**402**) und dem peripheren Bus (**403**) gekoppelt ist, wobei die Hochgeschwindigkeits-Busschnittstelle (**407**) Kommunikation mit einem chipexternen Vorrichtung ermöglicht.

Es folgen 28 Blatt Zeichnungen

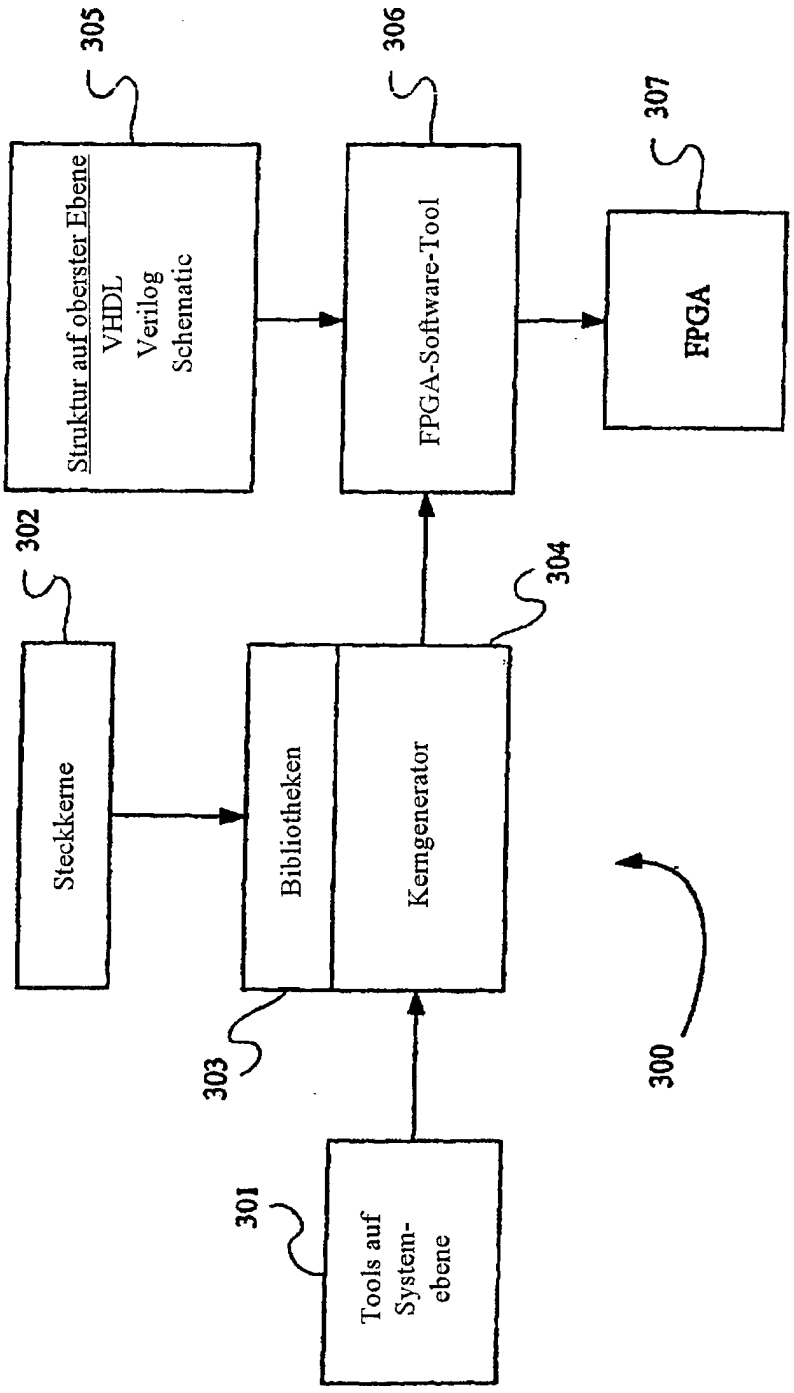
Anhängende Zeichnungen

Figur 1



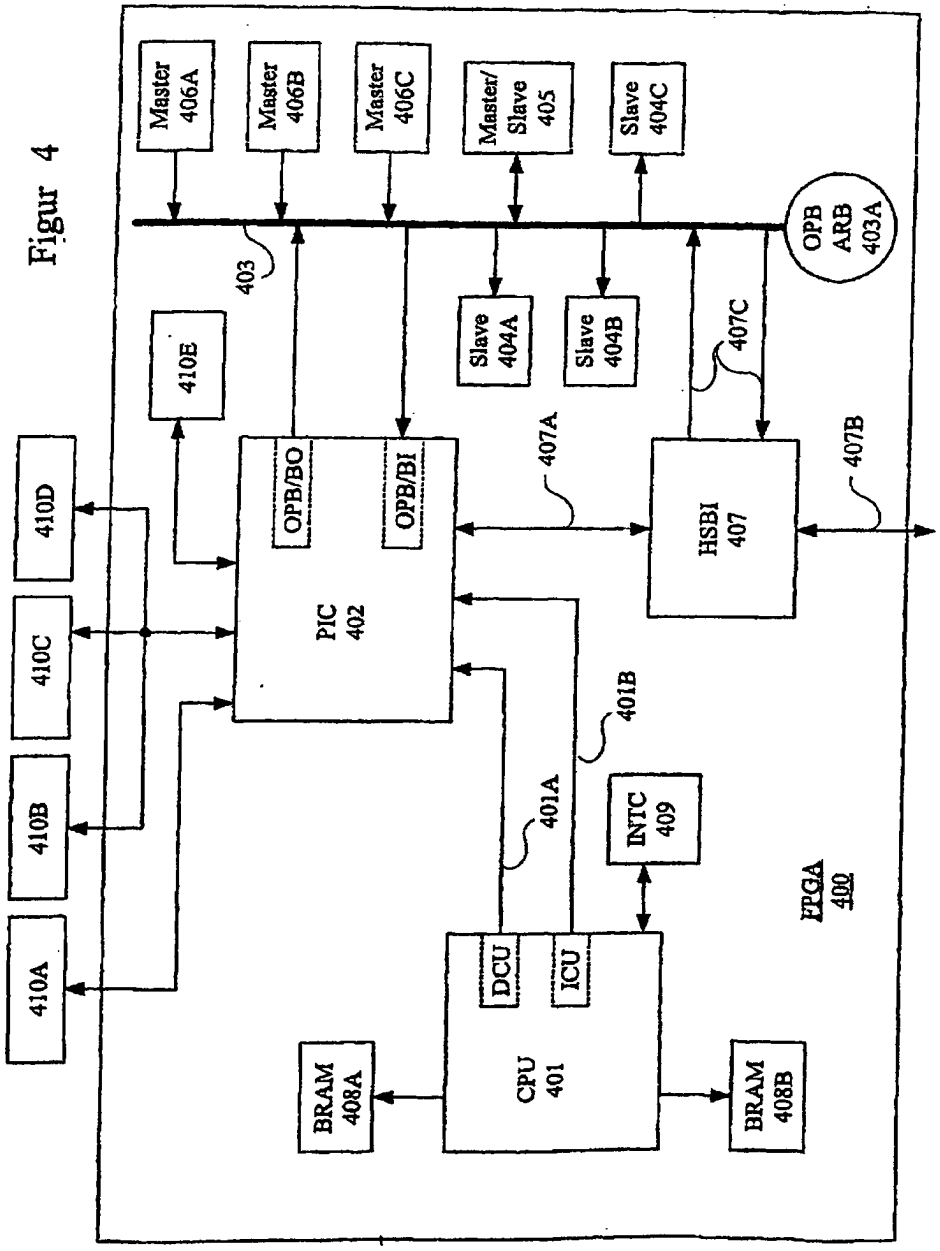

CLB 200
Scheibe 201B
Scheibe 201A
Y
YQ
X
XQ
FF 204B
FF 205B
FF 204A
FF 205A
LUT 202B
LUT 203B
LUT 202A
LUT 203A
G4
G3
G2
G1
F4
F3
F2
F1

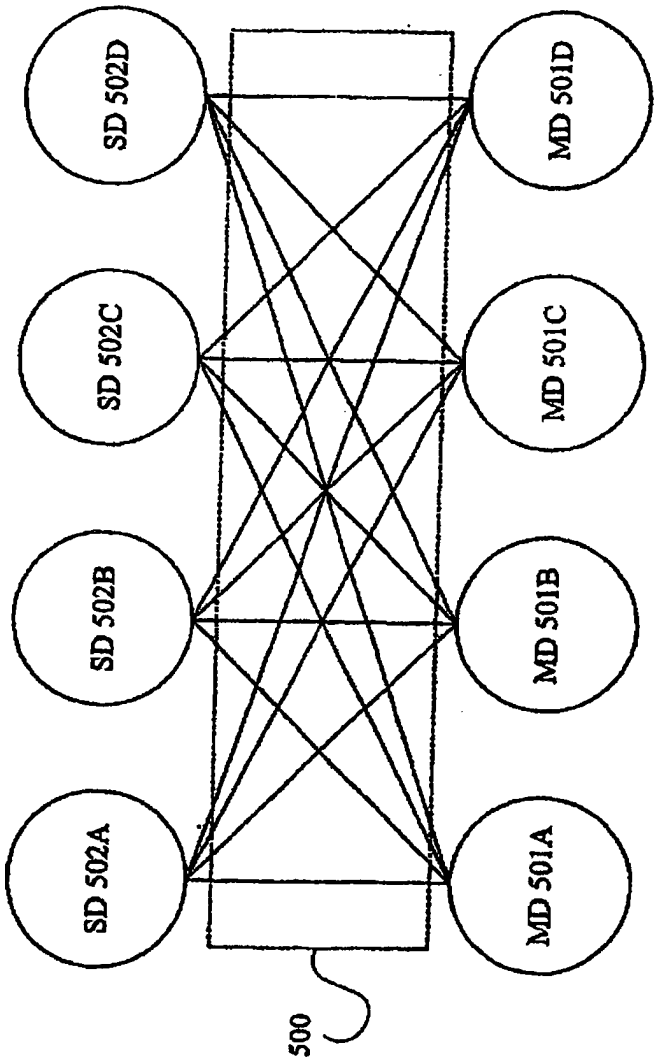
Figur 2


305
306
307
Struktur auf oberster Ebene
VHDL
Verilog
Schematic
FPGA-Software-Tool
FPGA
302
304
Steckkerne
Bibliotheken
Kerngenerator
303
300
301
Tools auf System-ebene

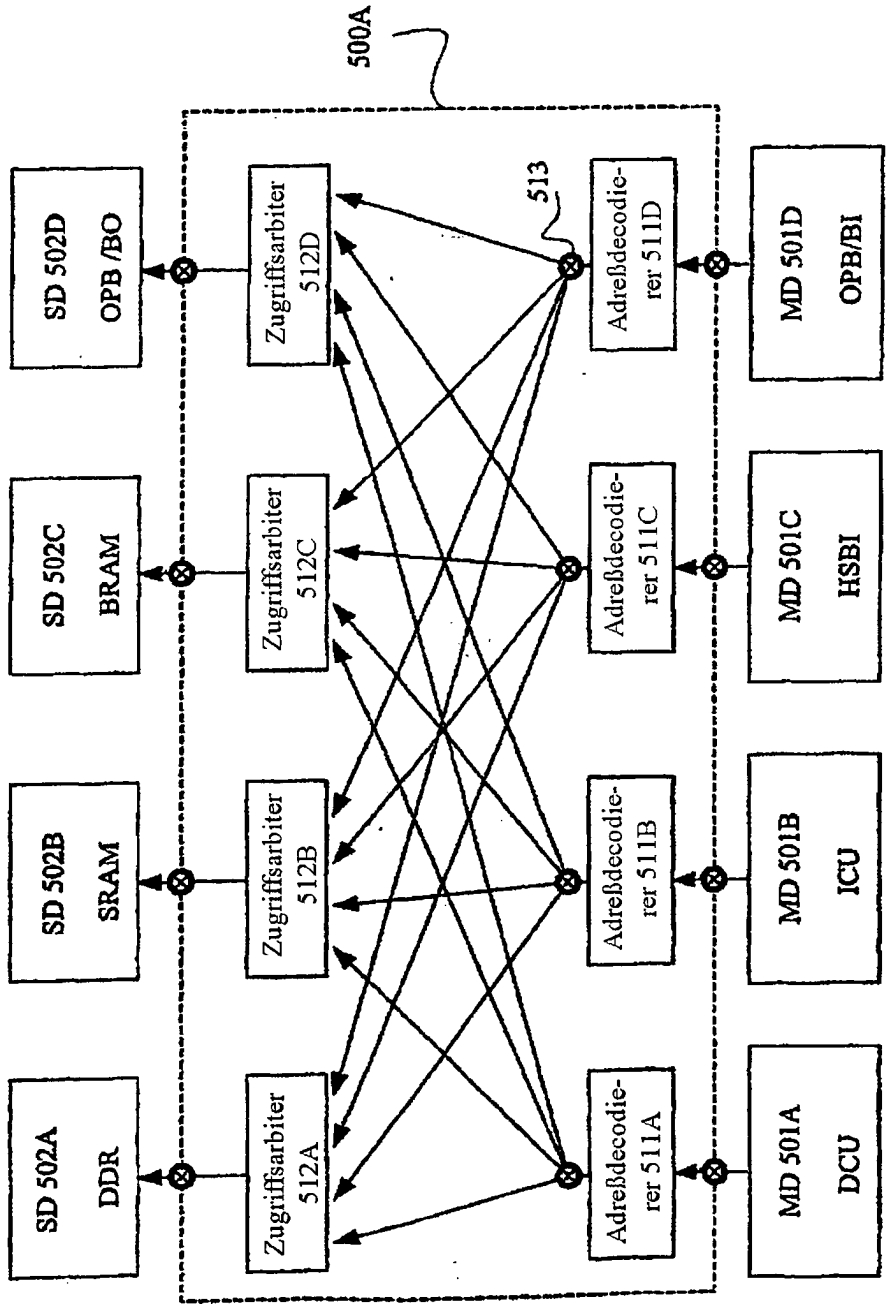
Figur 3

Figur 4

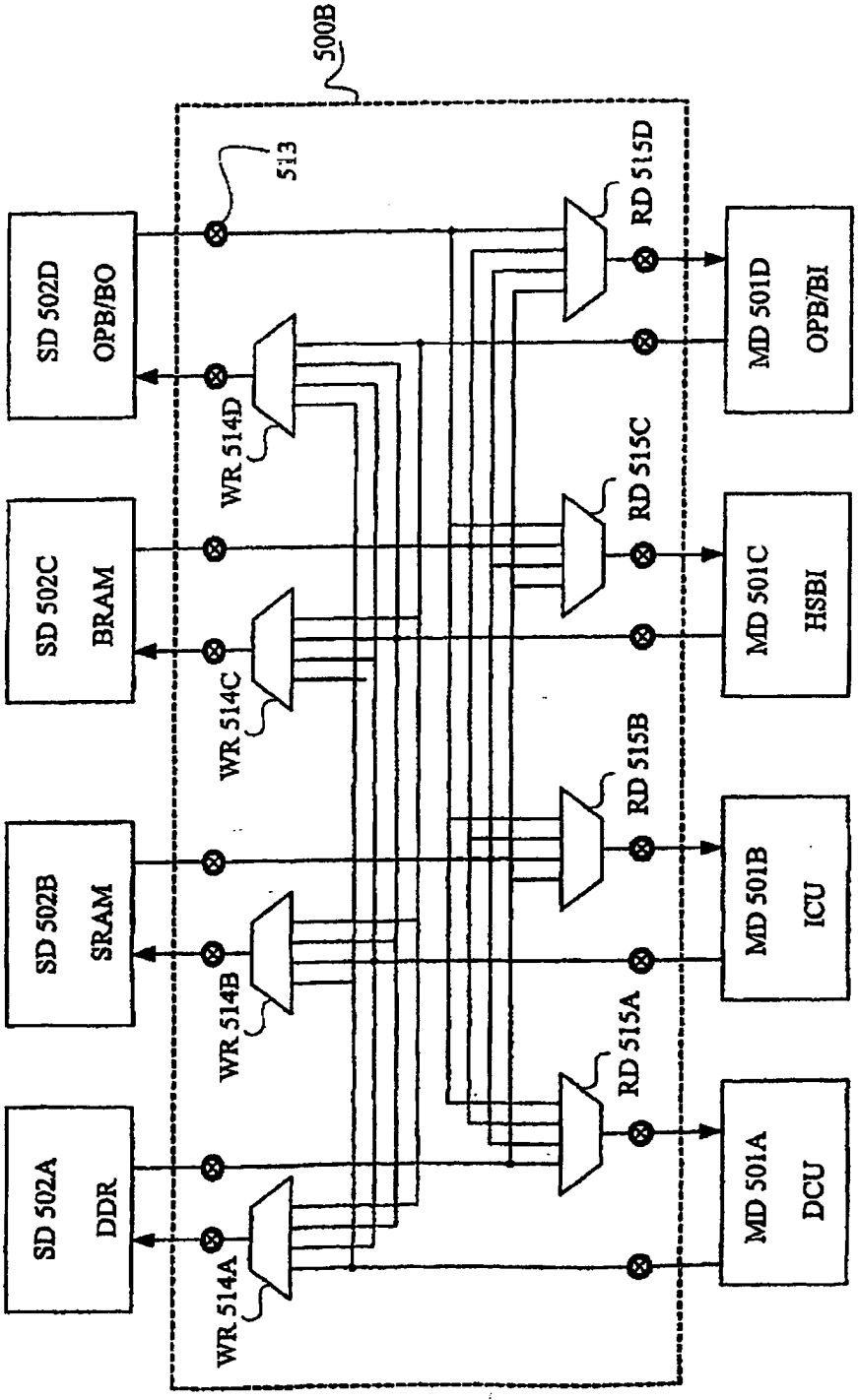

Master 406A
Master 406B
Master 406C
Master/ Slave 405
Slave 404C
OPB ARB 403A
403
Slave 404A
Slave 404B
407C
410E
410D
410C
410B
410A
OPB/BO
OPB/BI
PIC 402
407A
HSBI 407
407B
401B
401A
INTC 409
DCU
ICU
CPU 401
BRAM 408A
BRAM 408B
FPGA 400


SD 502D
SD 502C
SD 502B
SD 502A
MD 501D
MD 501C
MD 501B
MD 501A
500

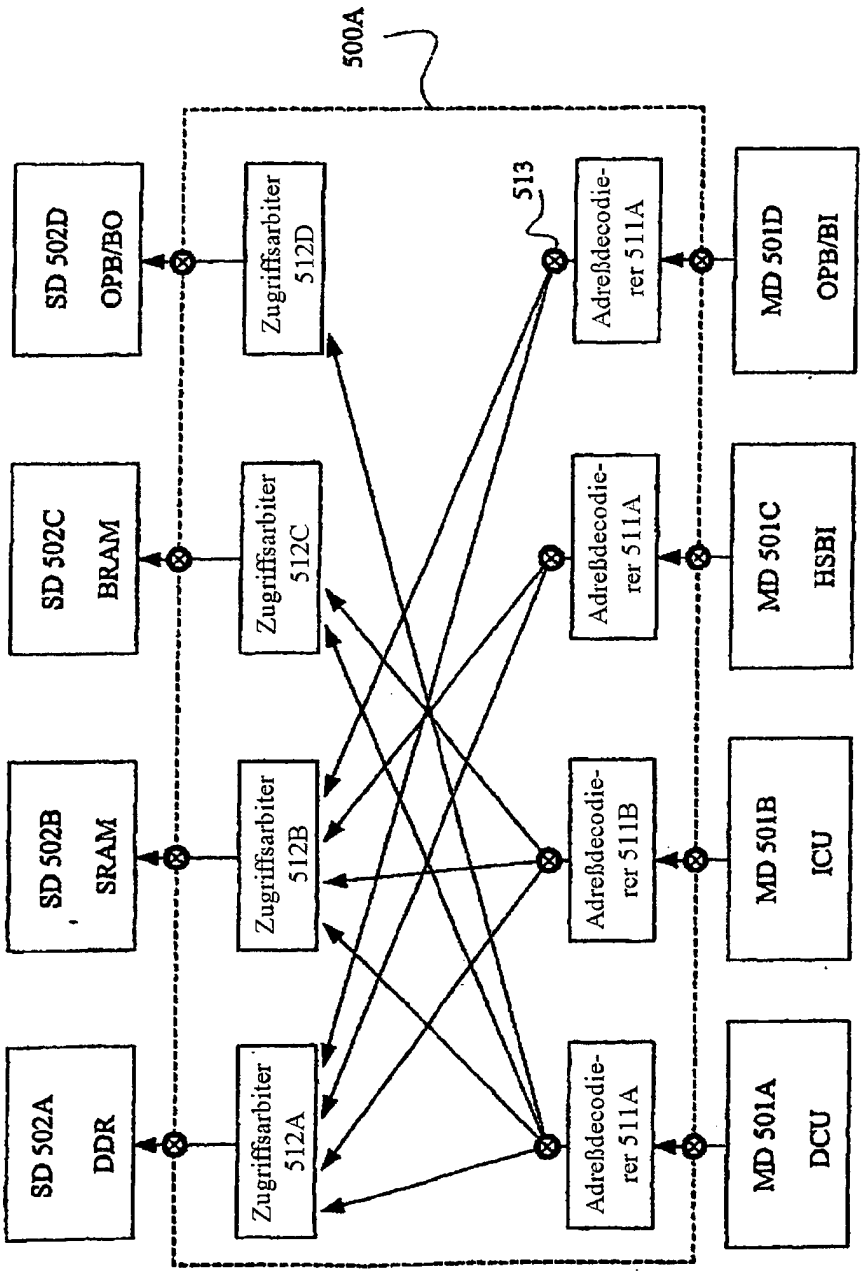
Figur 5A


500A
513
SD 502D
OPB /BO
SD 502C
BRAM
SD 502B
SRAM
SD 502A
DDR
Zugriffsarbiter 512D
Zugriffsarbiter 512C
Zugriffsarbiter 512B
Zugriffsarbiter 512A
Adreßdecodierer 511D
Adreßdecodierer 511C
Adreßdecodierer 511B
Adreßdecodierer 511A
MD 501D
OPB/BI
MD 501C
HSBI
MD 501B
ICU
MD 501A
DCU

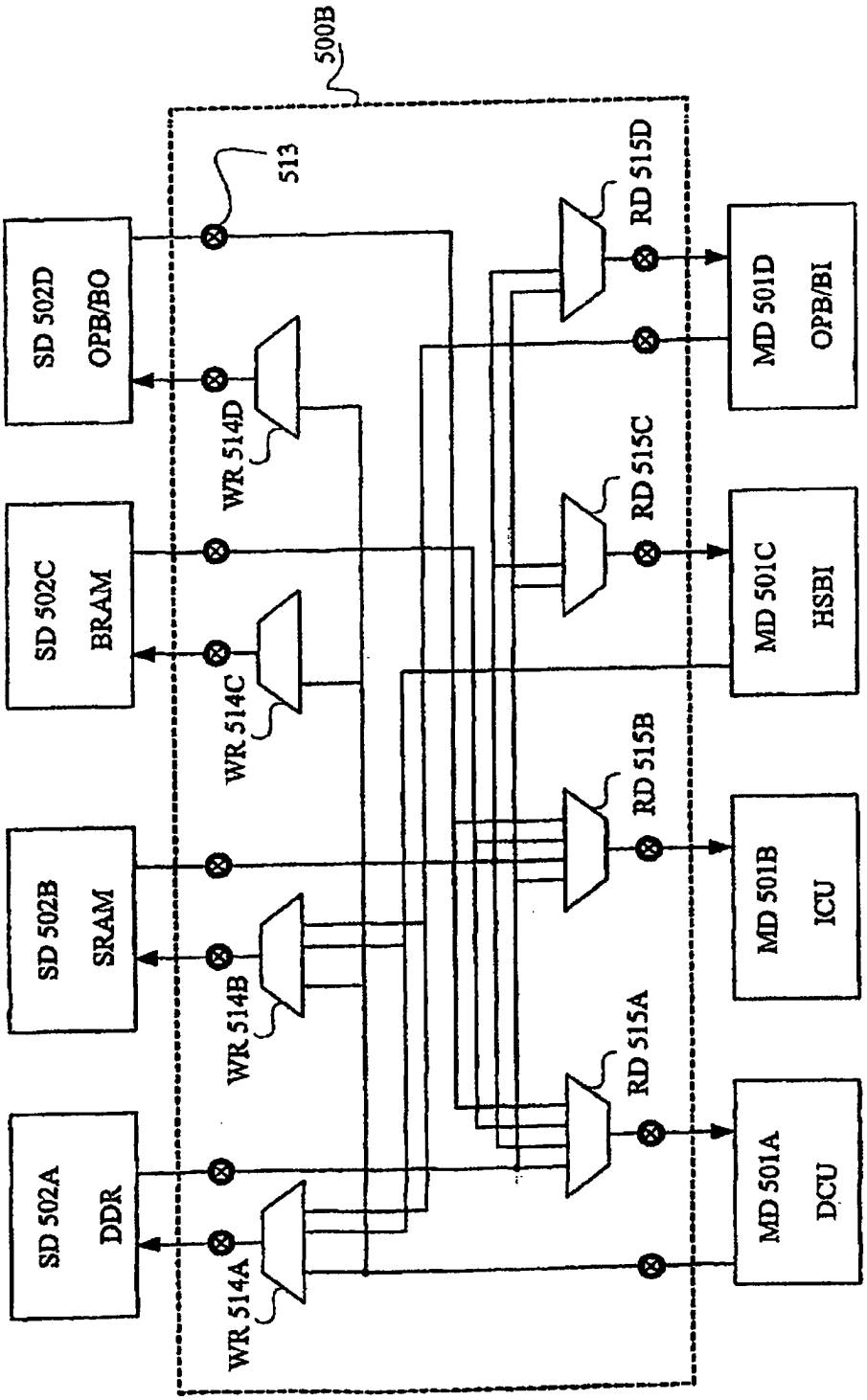
Figur 5B


500B
513
SD 502D
OPB/BO
SD 502C
BRAM
SD 502B
SRAM
SD 502A
DDR
WR 514D
WR 514C
WR 514B
WR 514A
RD 515D
RD 515C
RD 515B
RD 515A
MD 501D
OPB/BI
MD 501C
HSBI
MD 501B
ICU
MD 501A
DCU

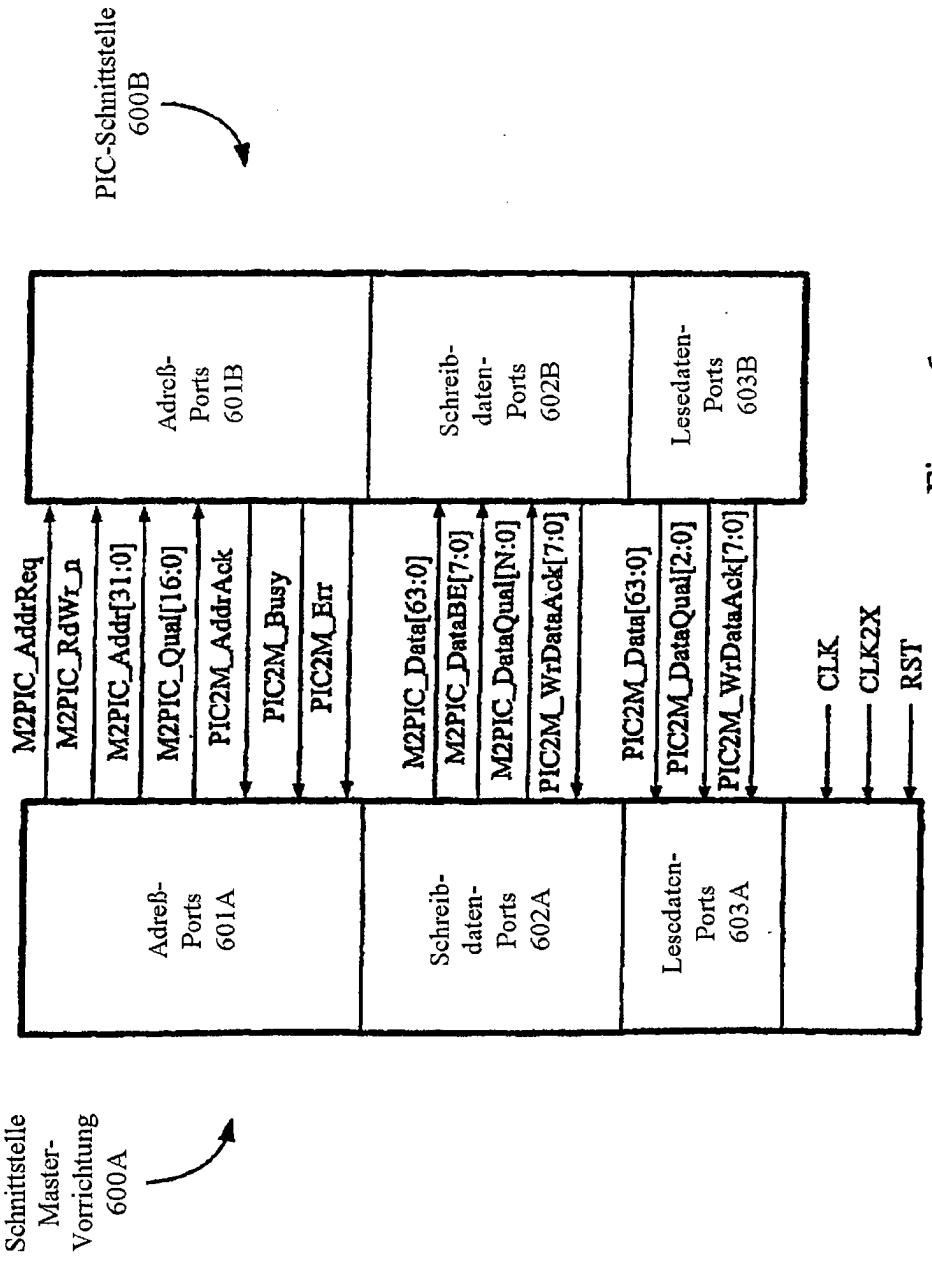
Figur 5C


500A
513
SD 502D OPB/BO
SD 502C BRAM
SD 502B SRAM
SD 502A DDR
Zugriffsarbiter 512D
Zugriffsarbiter 512C
Zugriffsarbiter 512B
Zugriffsarbiter 512A
Adreßdecodierer 511A
Adreßdecodierer 511A
Adreßdecodierer 511B
Adreßdecodierer 511A
MD 501D OPB/BI
MD 501C HSBI
MD 501B ICU
MD 501A DCU

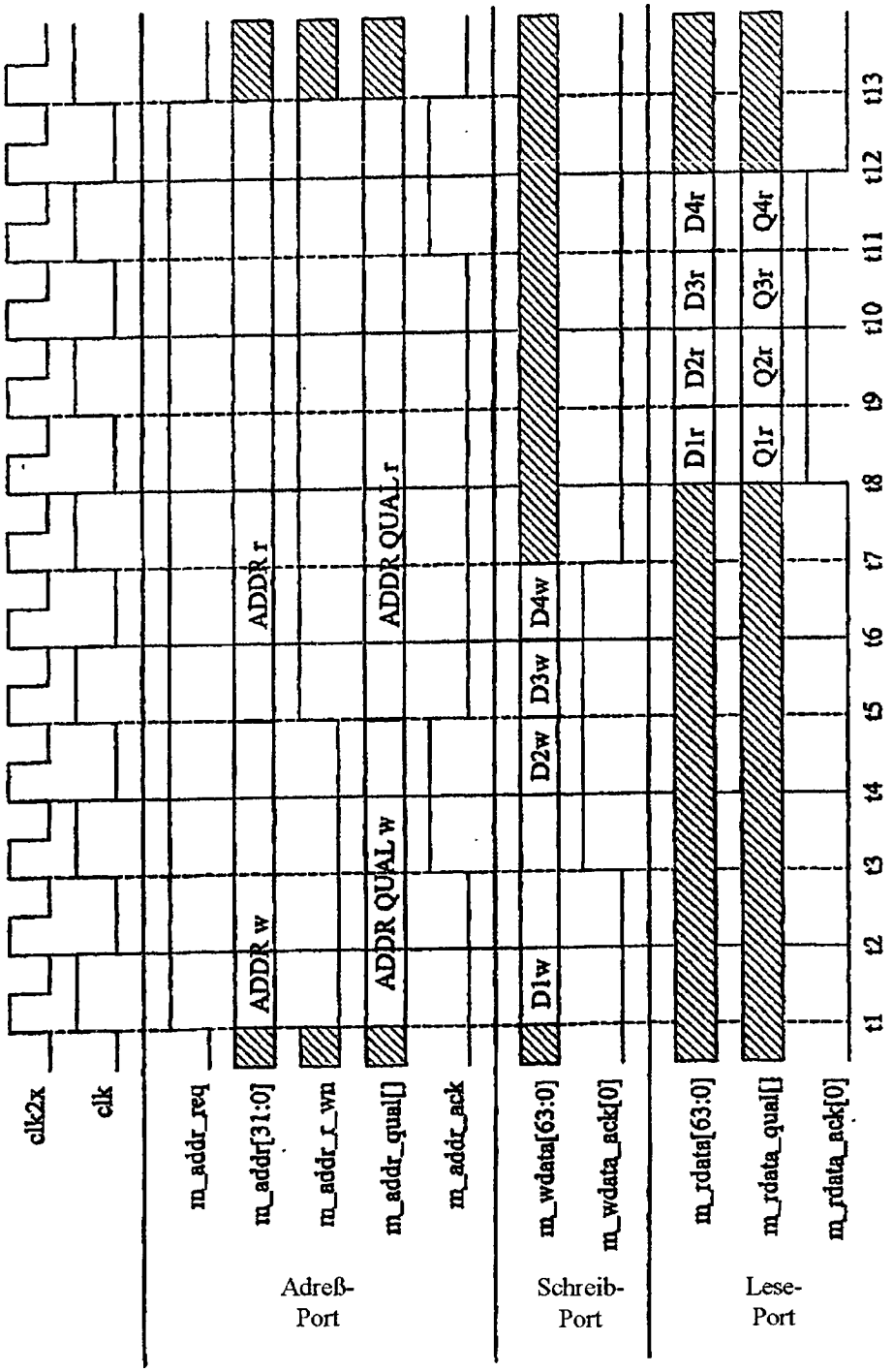
Figur 5D


500B
513
SD 502D OPB/BO
SD 502C BRAM
SD 502B SRAM
SD 502A DDR
WR 514D
WR 514C
WR 514B
WR 514A
RD 515D
RD 515C
RD 515B
RD 515A
MD 501D OPB/BI
MD 501C HSBI
MD 501B ICU
MD 501A DCU

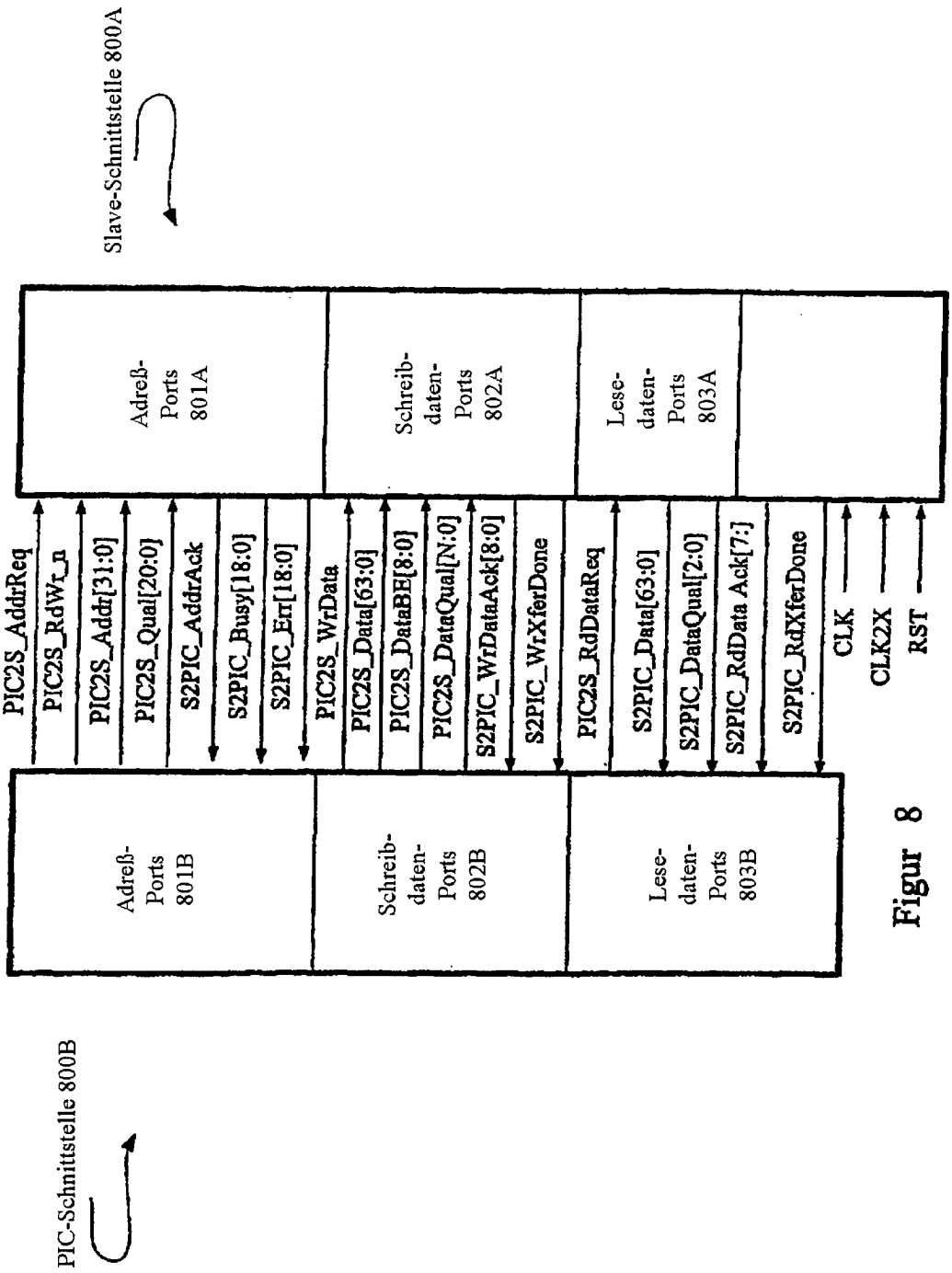
Figur 5E


PIC-Schnittstelle
600B
Adreß-
Ports
601B
Schreib-
daten-
Ports
602B
Lesedaten-
Ports
603B
M2PIC_AddrReq
M2PIC_RdWr_n
M2PIC_Addr[31:0]
M2PIC_Qual[16:0]
PIC2M_AddrAck
PIC2M_Busy
PIC2M_Err
M2PIC_Data[63:0]
M2PIC_DataBE[7:0]
M2PIC_DataQual[N:0]
PIC2M_WrDataAck[7:0]
PIC2M_Data[63:0]
PIC2M_DataQual[2:0]
PIC2M_WrDataAck[7:0]
CLK
CLK2X
RST
Schnittstelle
Master-
Vorrichtung
600A
Adreß-
Ports
601A
Schreib-
daten-
Ports
602A
Lesedaten-
Ports
603A

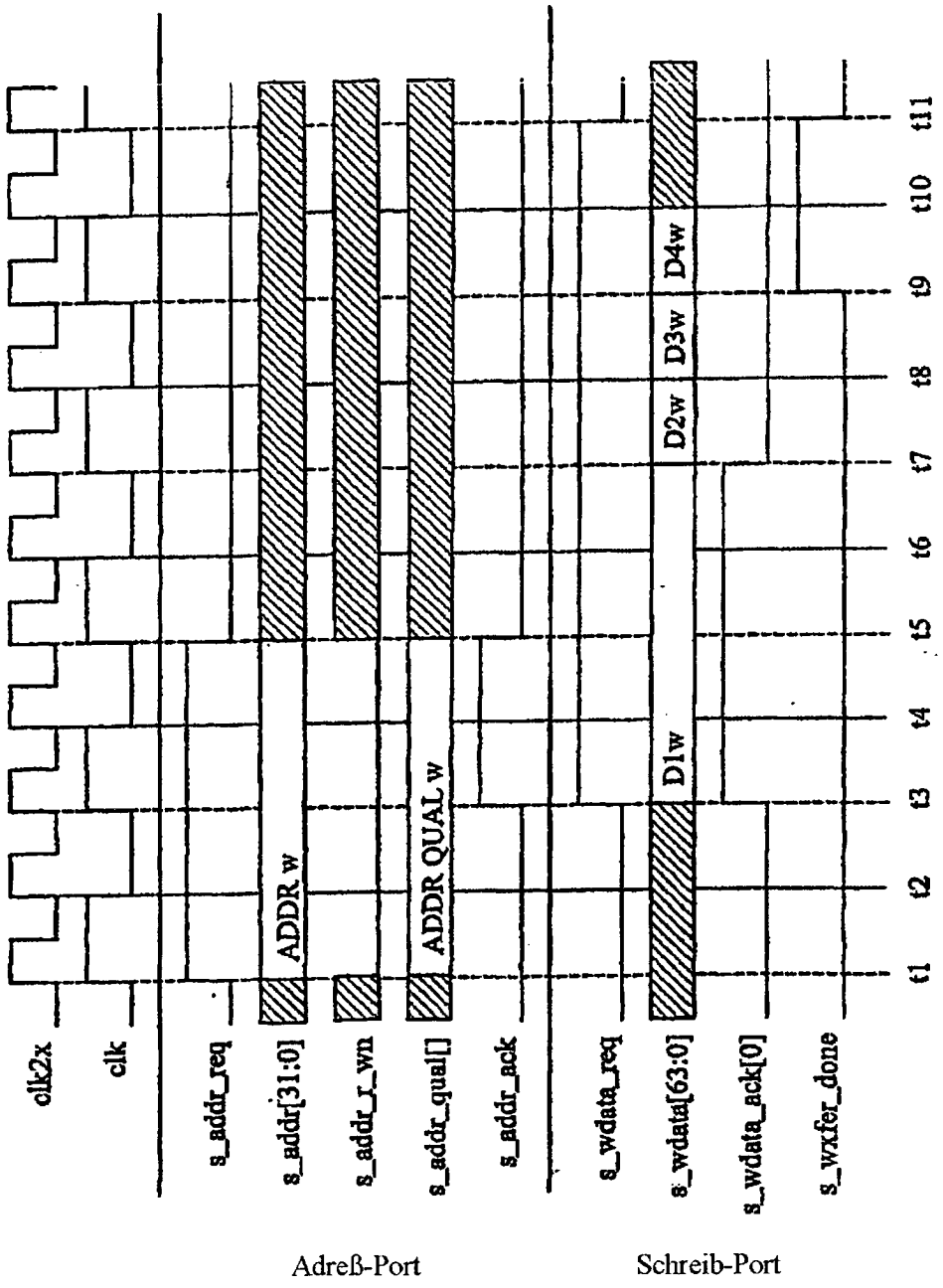
Figur 6


clk2x
clk
m_addr_req
m_addr[31:0]
m_addr_r_wn
m_addr_qual[]
m_addr_ack
m_wdata[63:0]
m_wdata_ack[0]
m_rdata[63:0]
m_rdata_qual[]
m_rdata_ack[0]
ADDR w
ADDR r
ADDR QUAL w
ADDR QUAL r
D1w
D2w
D3w
D4w
D1r
D2r
D3r
D4r
Q1r
Q2r
Q3r
Q4r
t1
t2
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12
t13
Adreß-
Port
Schreib-
Port
Lese-
Port

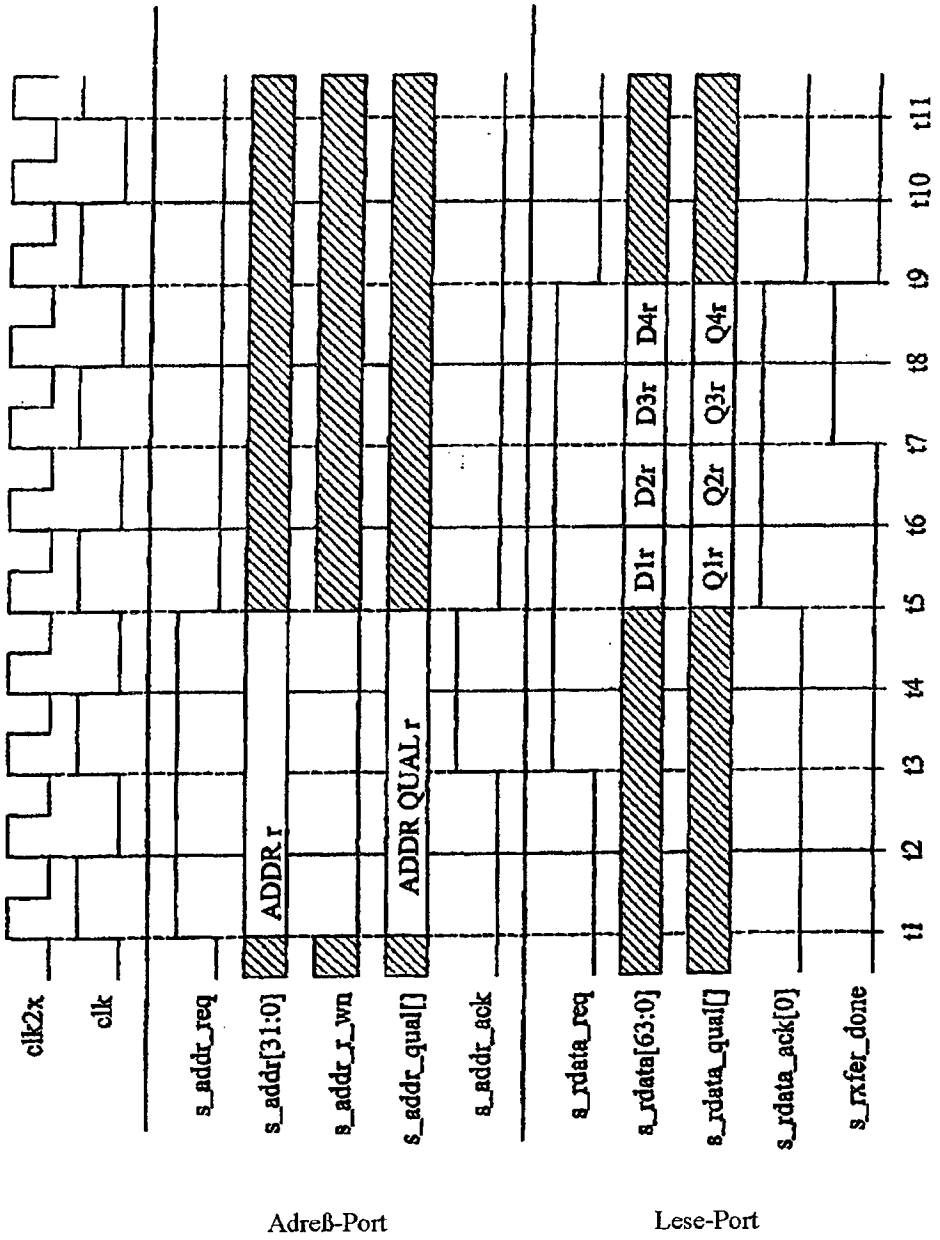
Figur 7


Slave-Schnittstelle 800A
Adreß-Ports 801A
Schreib-daten-Ports 802A
Lese-daten-Ports 803A
PIC2S_AddrReq
PIC2S_RdWr_n
PIC2S_Addr[31:0]
PIC2S_Qual[20:0]
S2PIC_AddrAck
S2PIC_Busy[18:0]
S2PIC_Err[18:0]
PIC2S_WrData
PIC2S_Data[63:0]
PIC2S_DataBE[8:0]
PIC2S_DataQual[N:0]
S2PIC_WrDataAck[8:0]
S2PIC_WrXferDone
PIC2S_RdDataReq
S2PIC_Data[63:0]
S2PIC_DataQual[2:0]
S2PIC_RdData Ack[7:]
S2PIC_RdXferDone
CLK
CLK2X
RST
Adreß-Ports 801B
Schreib-daten-Ports 802B
Lese-daten-Ports 803B
PIC-Schnittstelle 800B

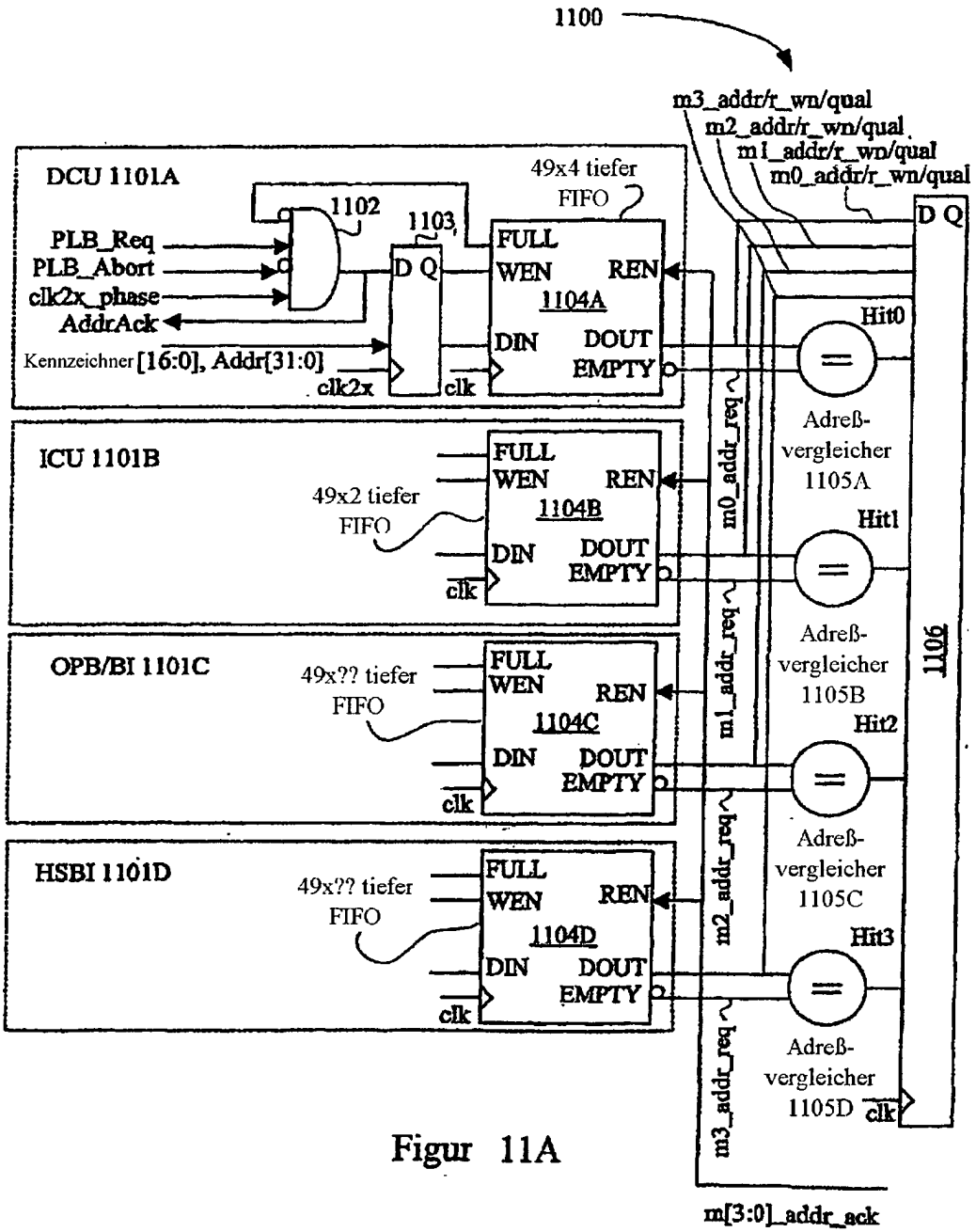
Figur 8


clk2x
clk
s_addr_req
s_addr[31:0]
s_addr_r_wn
s_addr_qual[]
s_addr_ack
s_wdata_req
s_wdata[63:0]
s_wdata_ack[0]
s_wxfer_done
ADDR w
ADDR QUAL w
D1w
D2w
D3w
D4w
t1
t2
t3
t4
t5
t6
t7
t8
t9
t10
t11
Adreß-Port
Schreib-Port

Figur 9


clk2x
clk
s_addr_req
s_addr[31:0]
s_addr_r_wn
s_addr_qual[]
s_addr_ack
s_rdata_req
s_rdata[63:0]
s_rdata_qual[]
s_rdata_ack[0]
s_rxfer_done
ADDR r
ADDR QUAL r
D1r
D2r
D3r
D4r
Q1r
Q2r
Q3r
Q4r
t1
t2
t3
t4
t5
t6
t7
t8
t9
t10
t11
Adreß-Port
Lese-Port

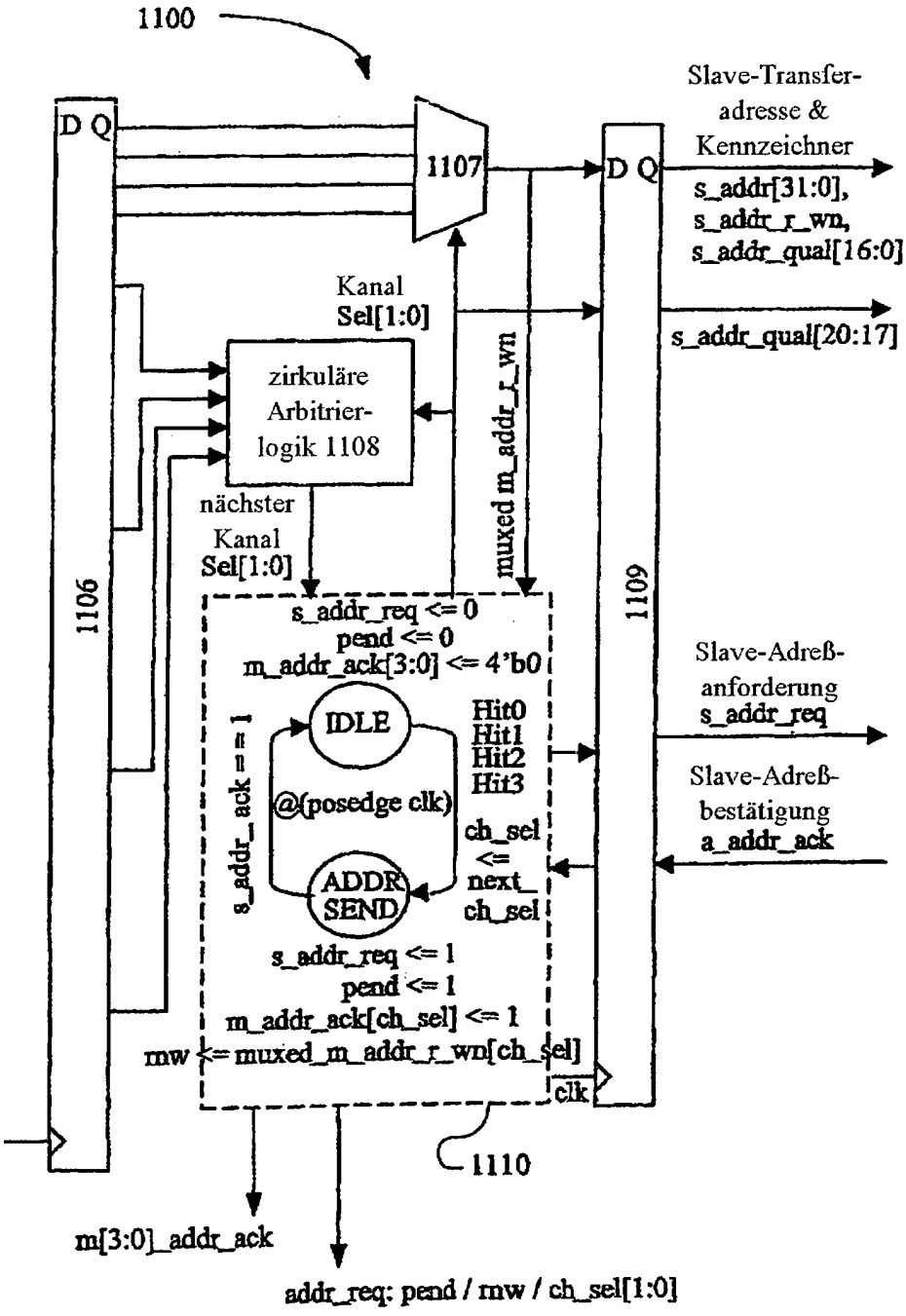
Figur 10



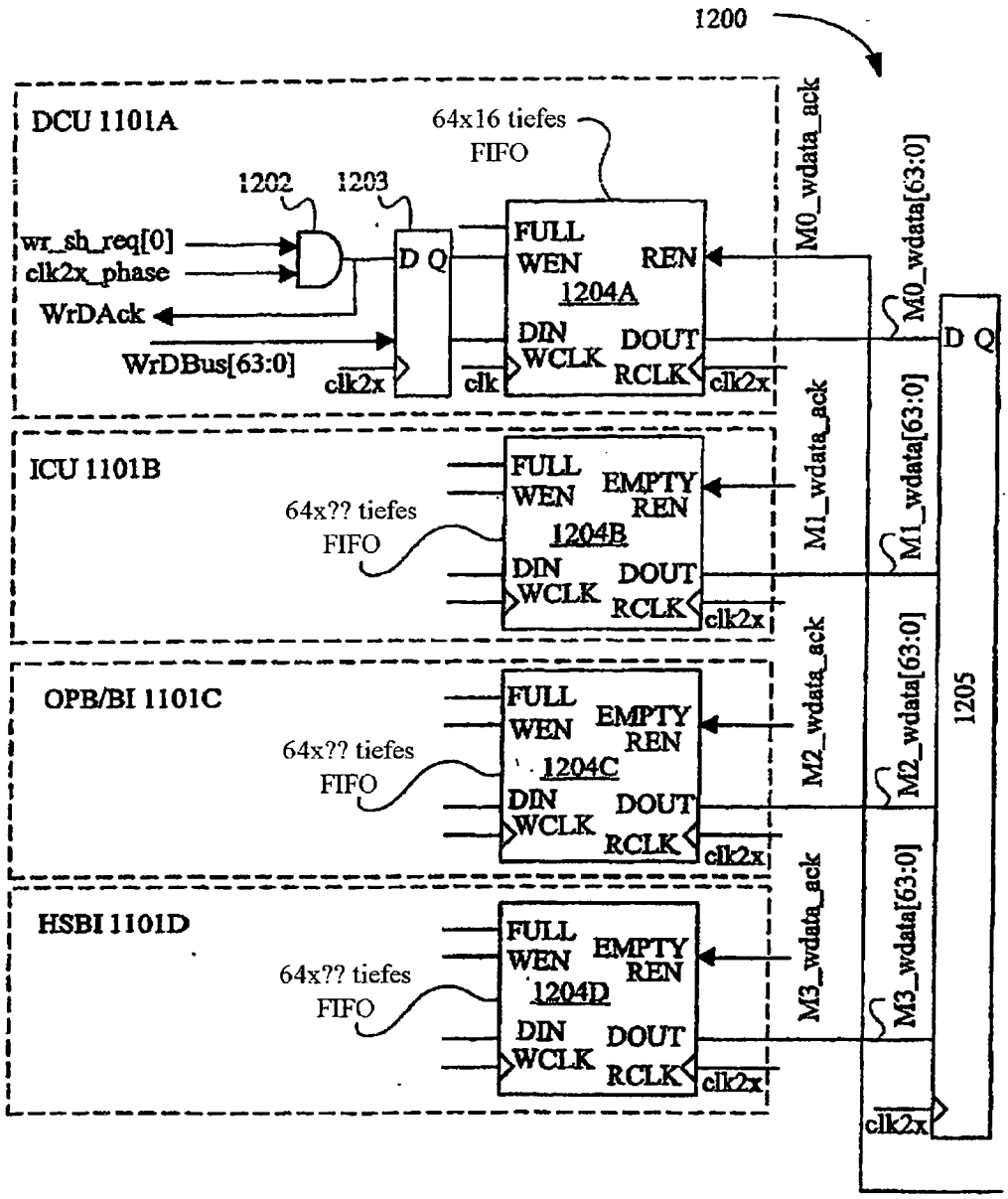
Figur 11A

Figur 11A	Figur 11B

Legende


1100
D Q
1107
D Q
Slave-Transfer-adresse & Kennzeichner
s_addr[31:0],
s_addr_r_wn,
s_addr_qual[16:0]
Kanal Sel[1:0]
s_addr_qual[20:17]
zirkuläre Arbitrier-logik 1108
nächster Kanal Sel[1:0]
muxed m_addr_r_wn
1106
1109
s_addr_req <= 0
pend <= 0
m_addr_ack[3:0] <= 4'b0
IDLE
s_addr_ack == 1
@(posedge clk)
ADDR SEND
Hit0
Hit1
Hit2
Hit3
ch_sel <= next_ch_sel
s_addr_req <= 1
pend <= 1
m_addr_ack[ch_sel] <= 1
mw <= muxed_m_addr_r_wn[ch_sel]
clk
Slave-Adreß-anforderung
s_addr_req
Slave-Adreß-bestätigung
a_addr_ack
1110
m[3:0]_addr_ack
addr_req: pend / mw / ch_sel[1:0]

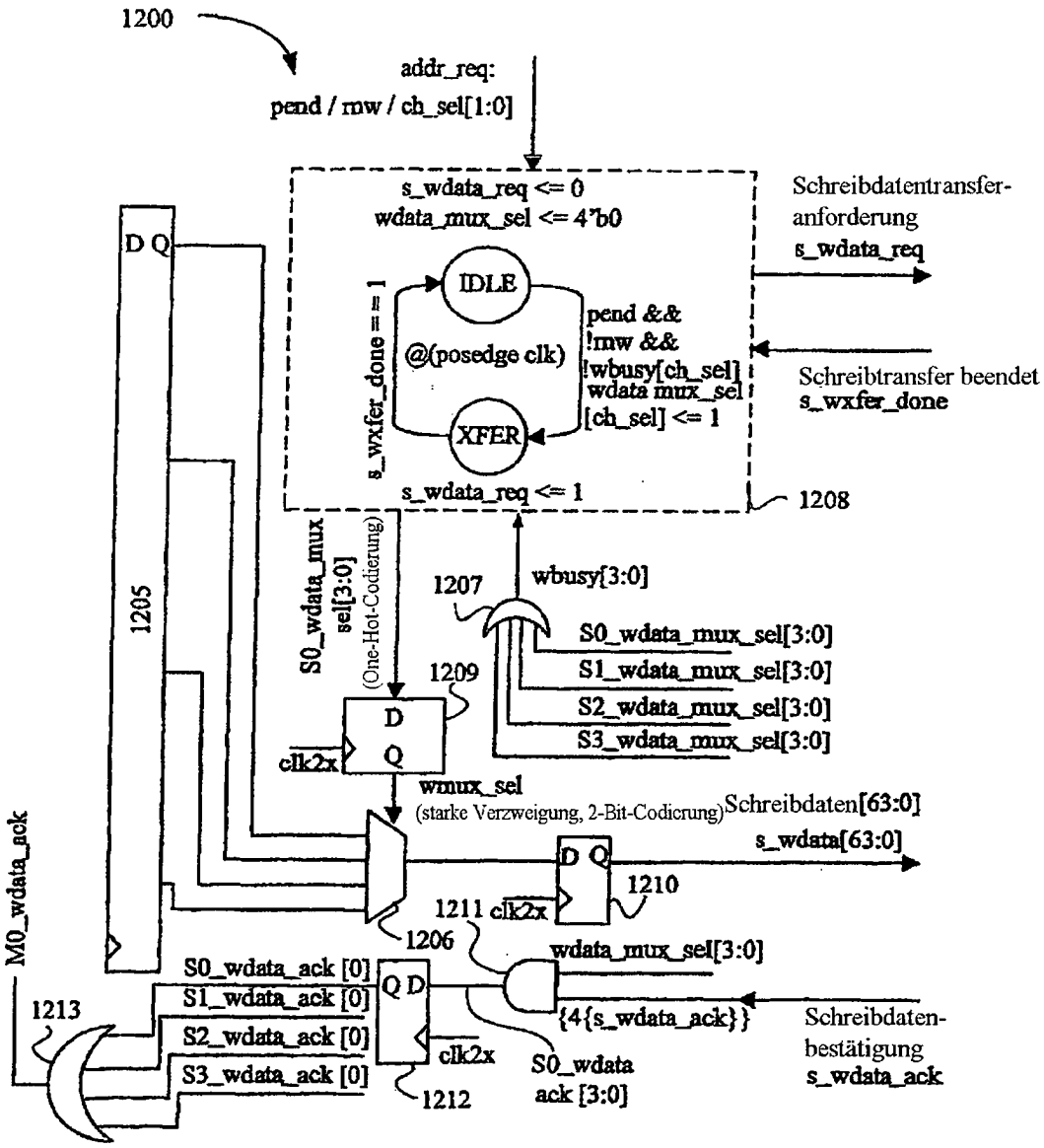
Figur 11B



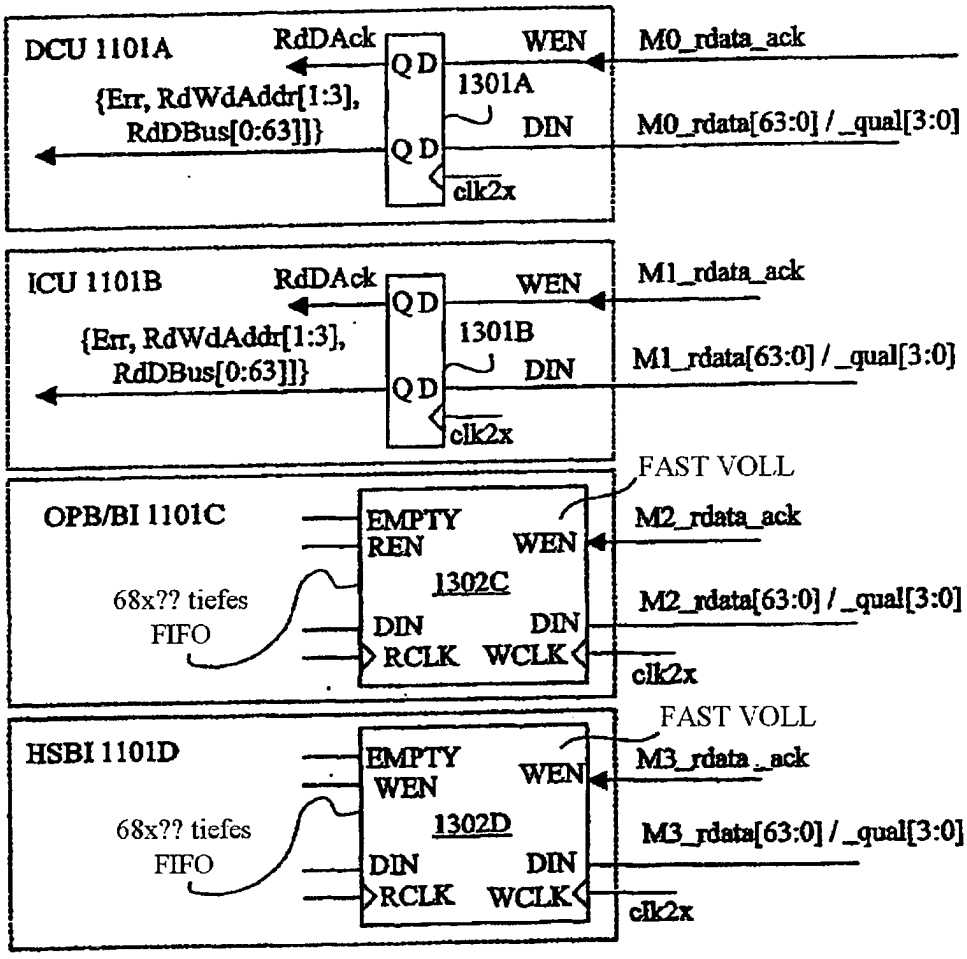
Figur 12A

Figur 12A	Figur 12B

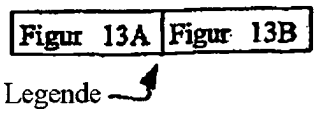
Legende

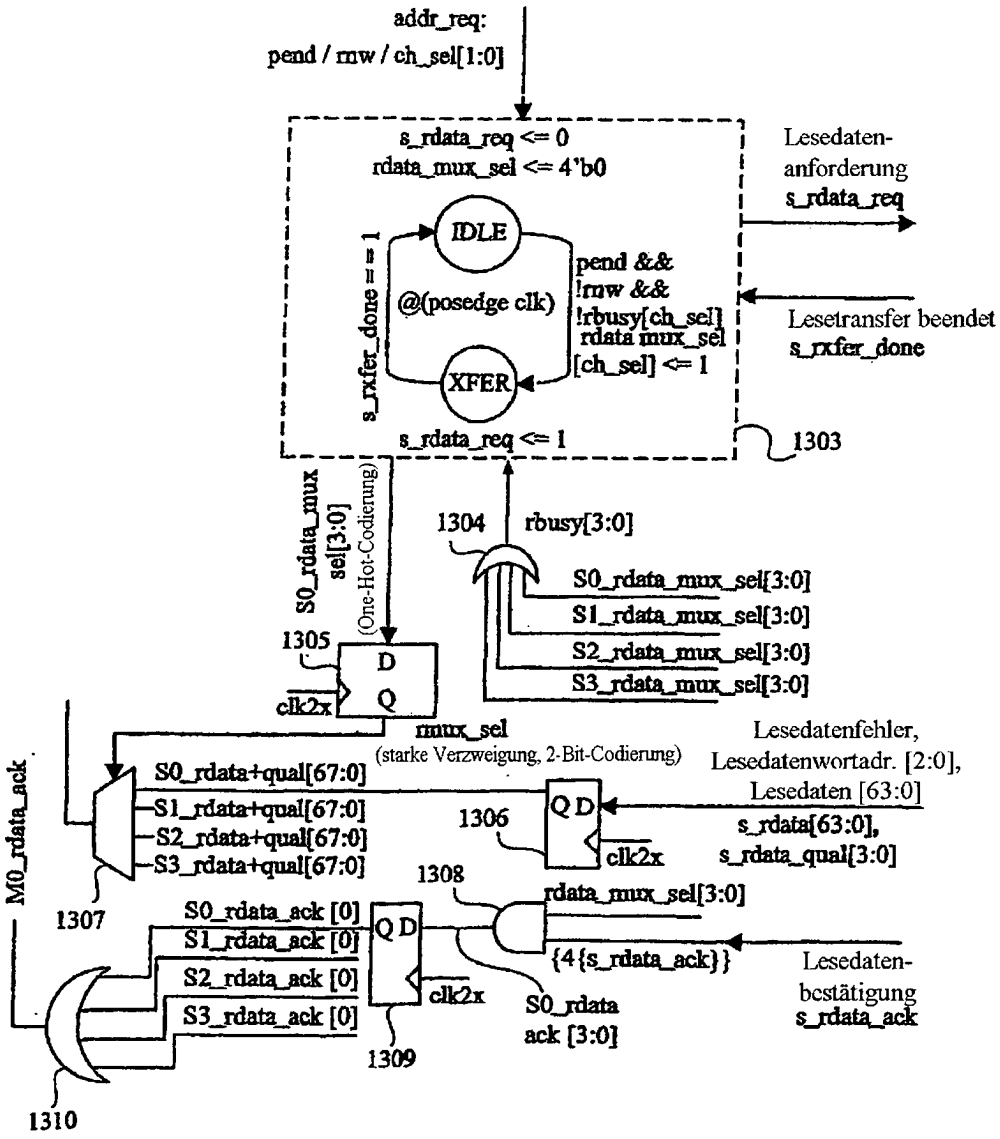

1200
addr_req:
pend / mw / ch_sel[1:0]
s_wdata_req <= 0
wdata_mux_sel <= 4'b0
IDLE
@(posedge clk)
XFER
pend &&
!mw &&
!wbusy[ch_sel]
wdata mux_sel
[ch_sel] <= 1
s_wxfer_done == 1
s_wdata_req <= 1
Schreibdatentransfer-
anforderung
s_wdata_req
Schreibtransfer beendet
s_wxfer_done
1208
D Q
1205
S0_wdata_mux
sel[3:0]
(One-Hot-Codierung)
1207
wbusy[3:0]
S0_wdata_mux_sel[3:0]
S1_wdata_mux_sel[3:0]
S2_wdata_mux_sel[3:0]
S3_wdata_mux_sel[3:0]
1209
D
Q
clk2x
wmux_sel
(starke Verzweigung, 2-Bit-Codierung)
Schreibdaten[63:0]
s_wdata[63:0]
D Q
1210
1211 clk2x
1206
wdata_mux_sel[3:0]
M0_wdata_ack
S0_wdata_ack [0]
S1_wdata_ack [0]
S2_wdata_ack [0]
S3_wdata_ack [0]
1213
Q D
clk2x
1212
S0_wdata
ack [3:0]
{4{s_wdata_ack}}
Schreibdaten-
bestätigung
s_wdata_ack

Figur 12B

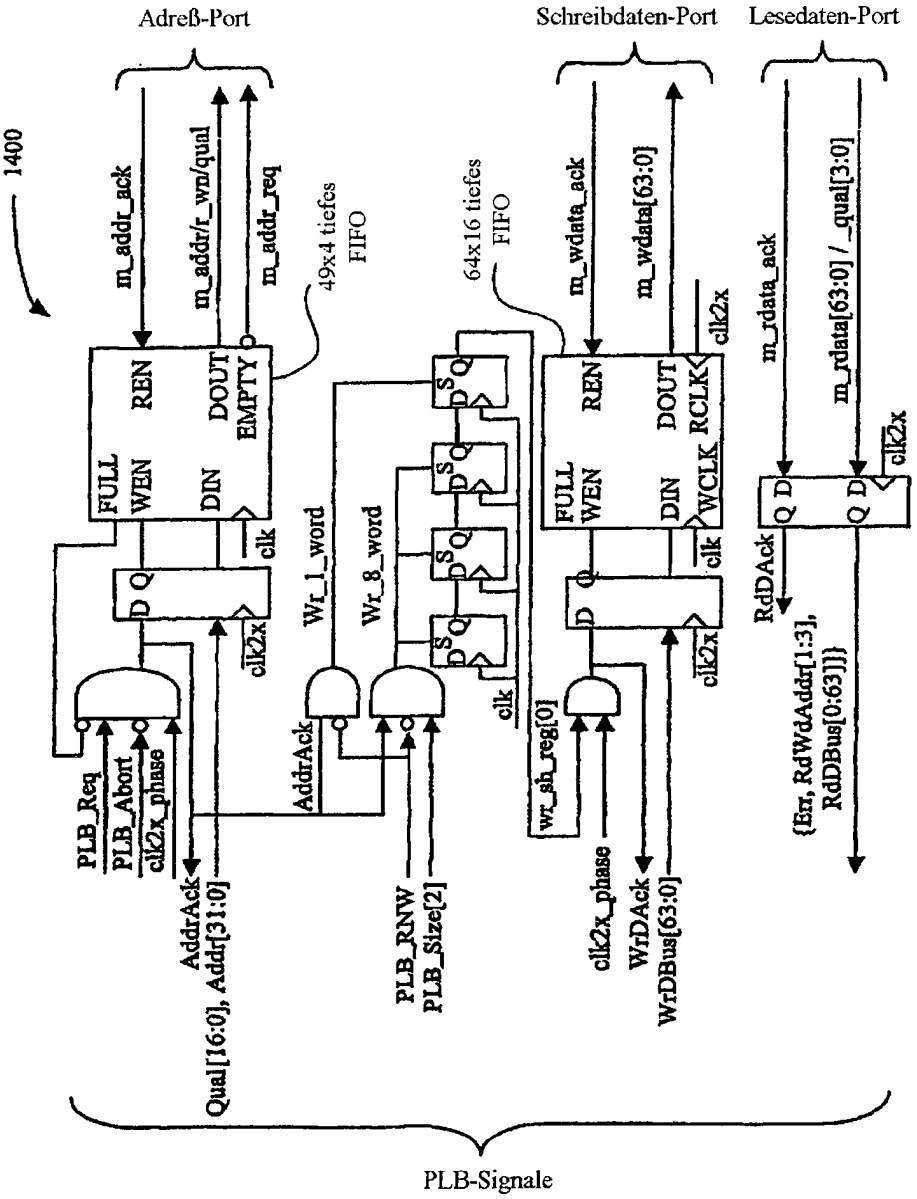

DCU 1101A
RdDAck
{Err, RdWdAddr[1:3], RdDBus[0:63]]}
Q D
Q D
WEN
1301A
DIN
clk2x
M0_rdata_ack
M0_rdata[63:0] / _qual[3:0]
ICU 1101B
RdDAck
{Err, RdWdAddr[1:3], RdDBus[0:63]]}
Q D
Q D
WEN
1301B
DIN
clk2x
M1_rdata_ack
M1_rdata[63:0] / _qual[3:0]
OPB/BI 1101C
68x?? tiefes FIFO
EMPTY
REN
WEN
1302C
DIN
RCLK
DIN
WCLK
FAST VOLL
M2_rdata_ack
M2_rdata[63:0] / _qual[3:0]
clk2x
HSBI 1101D
68x?? tiefes FIFO
EMPTY
WEN
WEN
1302D
DIN
RCLK
DIN
WCLK
FAST VOLL
M3_rdata _ack
M3_rdata[63:0] / _qual[3:0]
clk2x

Figur 13A

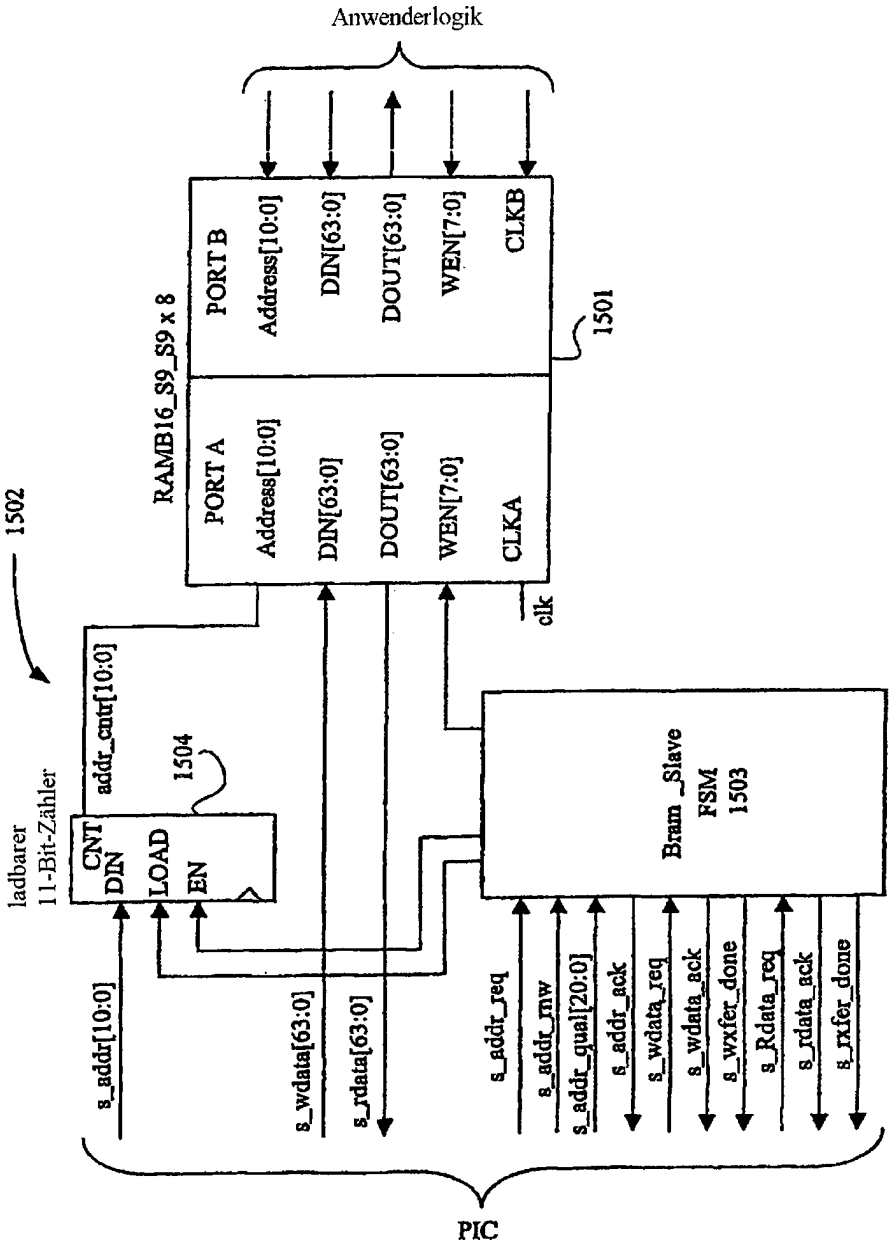

Figur 13A
Figur 13B
Legende


addr_req:
pend / rnw / ch_sel[1:0]
s_rdata_req <= 0
rdata_mux_sel <= 4'b0
IDLE
@(posedge clk)
XFER
s_rxfer_done == 1
pend &&
!rnw &&
!rbusy[ch_sel]
rdata mux_sel
[ch_sel] <= 1
s_rdata_req <= 1
1303
Lesedaten-
anforderung
s_rdata_req
Lesetransfer beendet
s_rxfer_done
S0_rdata_mux
sel[3:0]
(One-Hot-Codierung)
1304
rbusy[3:0]
S0_rdata_mux_sel[3:0]
S1_rdata_mux_sel[3:0]
S2_rdata_mux_sel[3:0]
S3_rdata_mux_sel[3:0]
1305
D
Q
clk2x
rmux_sel
(starke Verzweigung, 2-Bit-Codierung)
Lesedatenfehler,
Lesedatenwortadr. [2:0],
Lesedaten [63:0]
S0_rdata+qual[67:0]
S1_rdata+qual[67:0]
S2_rdata+qual[67:0]
S3_rdata+qual[67:0]
1306
Q D
clk2x
s_rdata[63:0],
s_rdata_qual[3:0]
M0_rdata_ack
1307
1308
rdata_mux_sel[3:0]
S0_rdata_ack [0]
S1_rdata_ack [0]
S2_rdata_ack [0]
S3_rdata_ack [0]
Q D
clk2x
{4{s_rdata_ack}}
Lesedaten-
bestätigung
s_rdata_ack
S0_rdata
ack [3:0]
1309
1310

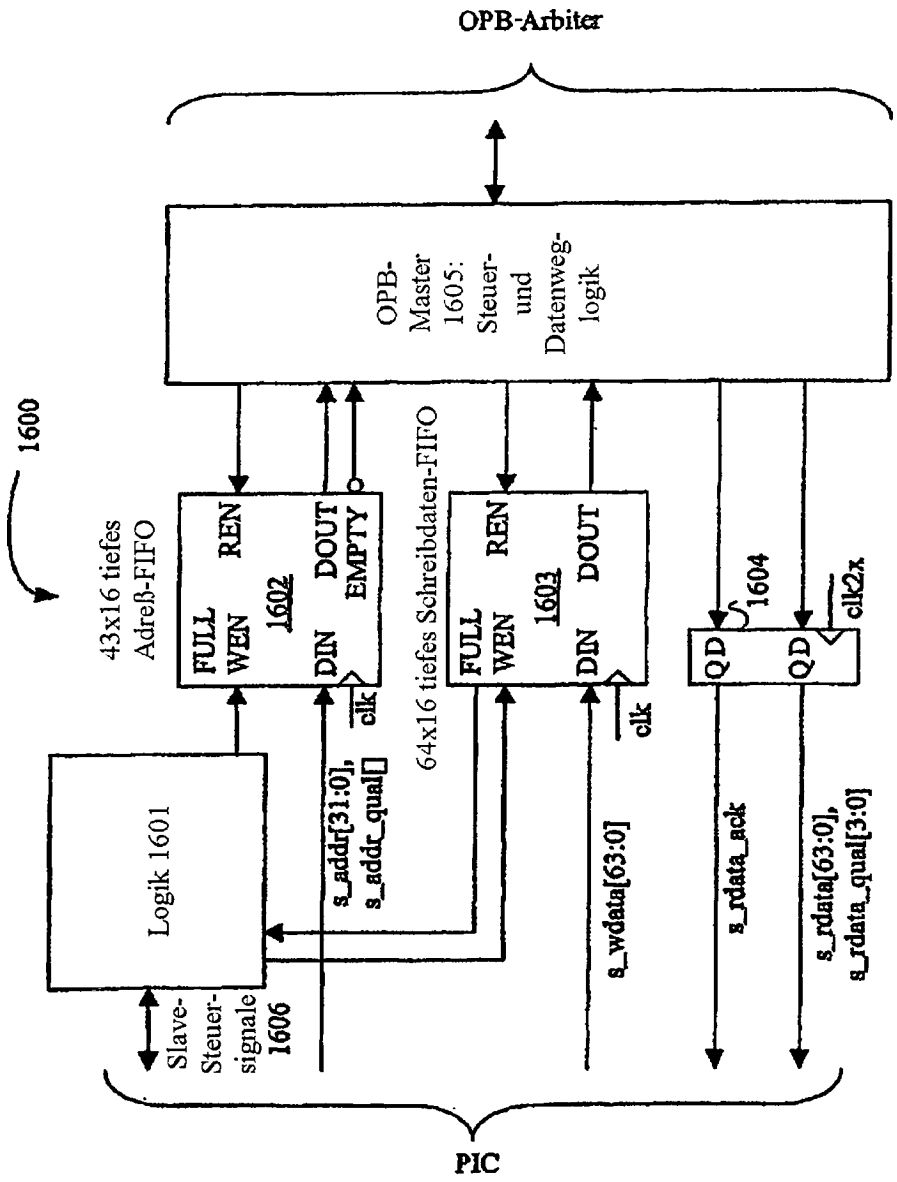
Figur 13B


1400
Adreß-Port
Schreibdaten-Port
Lesedaten-Port
m_addr_ack
m_addr/r_wn/qual
m_addr_req
49x4 tiefes FIFO
64x16 tiefes FIFO
m_wdata_ack
m_wdata[63:0]
m_rdata_ack
m_rdata[63:0] / _qual[3:0]
FULL
WEN
REN
DOUT
EMPTY
DIN
clk
clk2x
RCLK
WCLK
Wr_1_word
Wr_8_word
AddrAck
RdDAck
{Err, RdWdAddr[1:3], RdDBus[0:63]}
PLB_Req
PLB_Abort
clk2x_phase
Qual[16:0], Addr[31:0]
PLB_RNW
PLB_Size[2]
wr_sh_reg[0]
WrDAck
WrDBus[63:0]
PLB-Signale

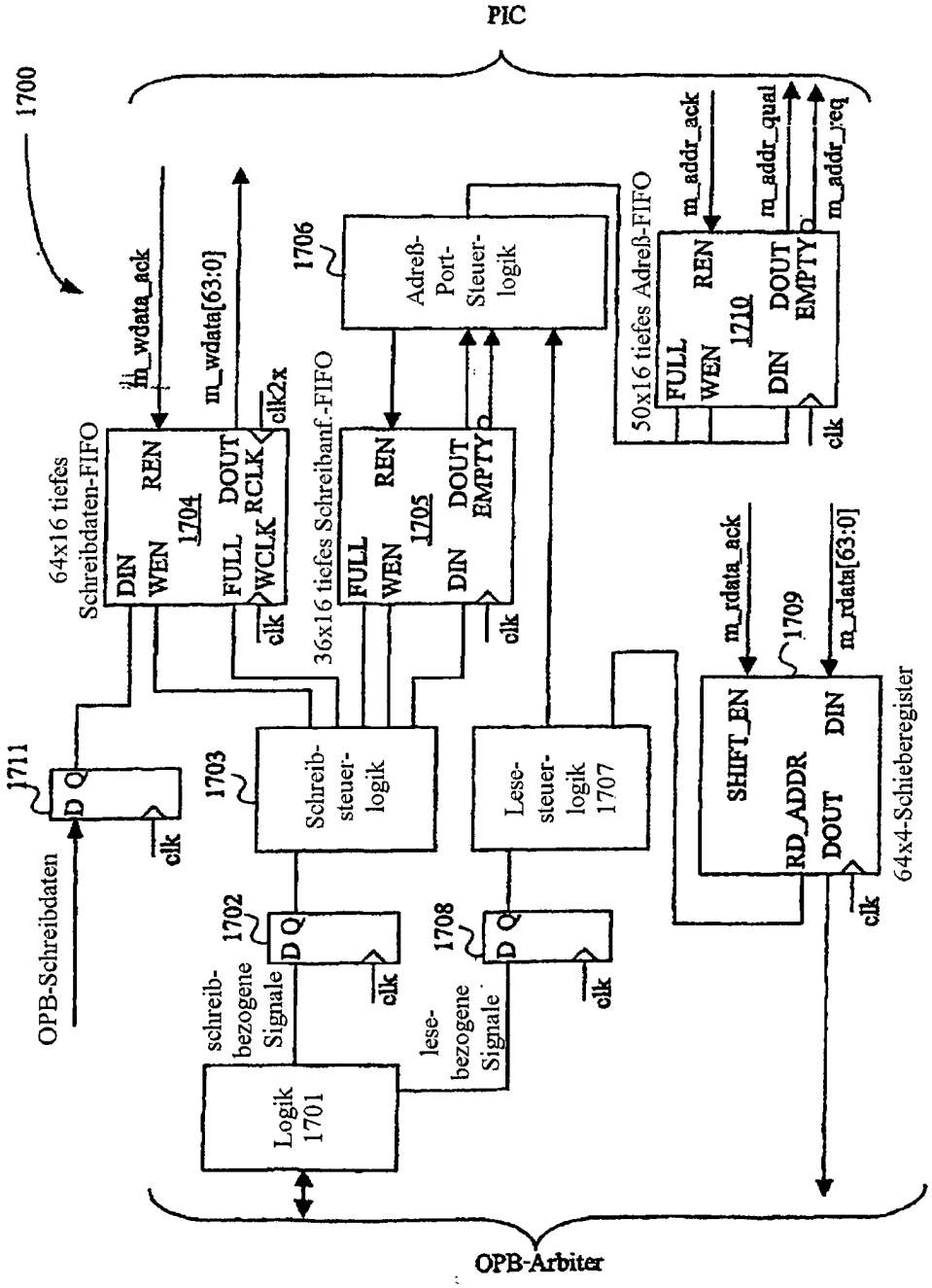
Figur 14


Anwenderlogik
RAMB16_S9_S9 x 8
PORT B
Address[10:0]
DIN[63:0]
DOUT[63:0]
WEN[7:0]
CLKB
1501
PORT A
Address[10:0]
DIN[63:0]
DOUT[63:0]
WEN[7:0]
CLKA
clk
1502
addr_cntr[10:0]
1504
ladbarer
11-Bit-Zähler
CNT
DIN
LOAD
EN
Bram _Slave
FSM
1503
s_addr[10:0]
s_wdata[63:0]
s_rdata[63:0]
s_addr_req
s_addr_rnw
s_addr_qual[20:0]
s_addr_ack
s_wdata_req
s_wdata_ack
s_wxfer_done
s_Rdata_req
s_rdata_ack
s_rxfer_done
PIC

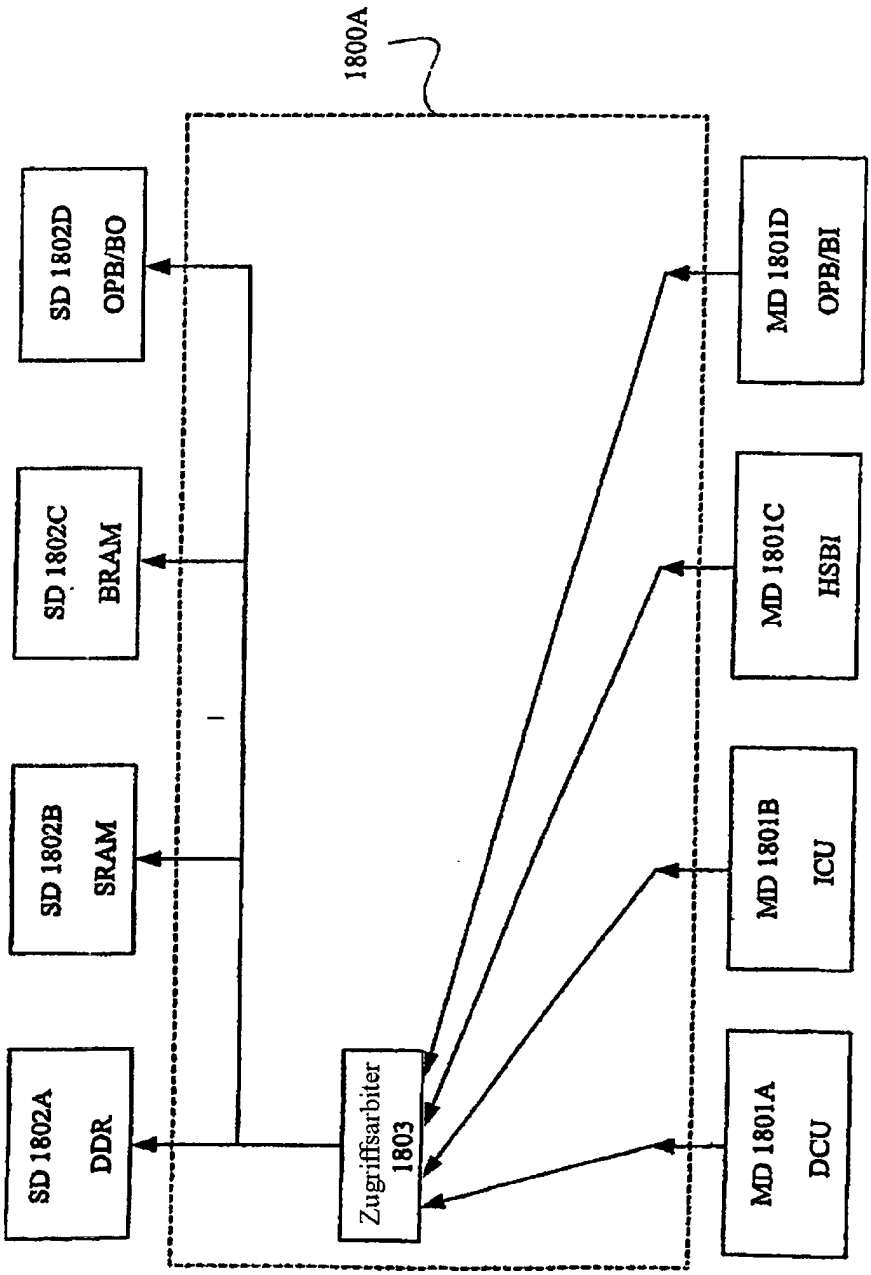
Figur 15


OPB-Arbiter
OPB-Master 1605: Steuer- und Datenweg-logik
1600
43x16 tiefes Adreß-FIFO
64x16 tiefes Schreibdaten-FIFO
FULL
WEN
REN
DOUT
EMPTY
DIN
1602
1603
1604
QD
clk
clk2x
Logik 1601
Slave-Steuer-signale 1606
s_addr[31:0],
s_addr_qual[]
s_wdata[63:0]
s_rdata_ack
s_rdata[63:0],
s_rdata_qual[3:0]
PIC

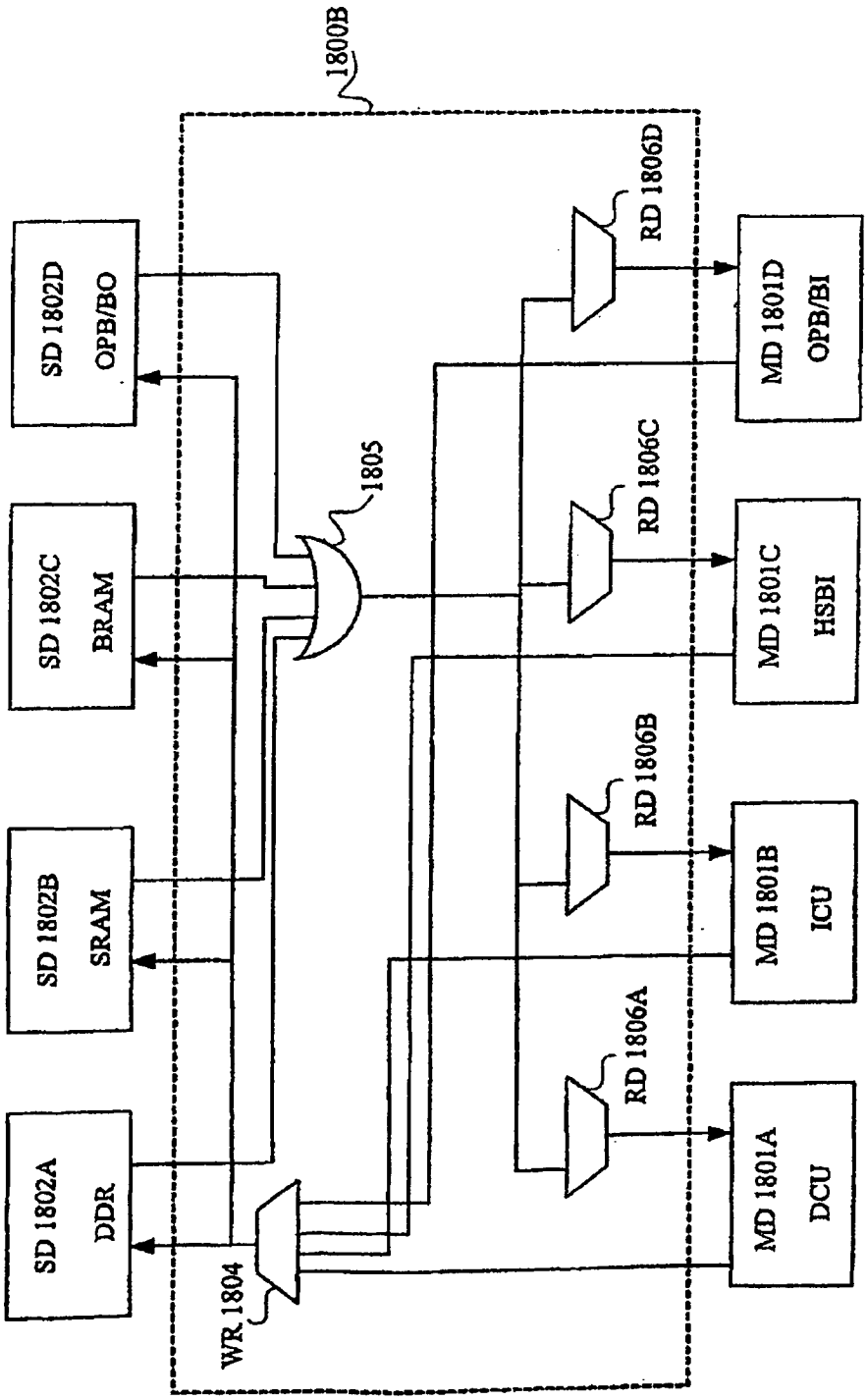
Figur 16


PIC
1700
m_addr_ack
m_addr_qual
m_addr_req
1706
Adreß-
Port-
Steuer-
logik
50x16 tiefes Adreß-FIFO
FULL
WEN
REN
1710
DIN
DOUT
EMPTY
clk
in_wdata_ack
m_wdata[63:0]
clk2x
64x16 tiefes
Schreibdaten-FIFO
DIN
WEN
REN
1704
FULL
DOUT
WCLK
RCLK
clk
36x16 tiefes Schreibanf.-FIFO
FULL
WEN
REN
1705
DIN
DOUT
EMPTY
clk
m_rdata_ack
1709
m_rdata[63:0]
SHIFT_EN
RD_ADDR
DIN
DOUT
clk
64x4-Schieberegister
1711
D
Q
clk
1703
Schreib-
steuer-
logik
Lese-
steuer-
logik
1707
OPB-Schreibdaten
1702
1708
schreib-
bezogene
Signale
lese-
bezogene
Signale
Logik
1701
OPB-Arbiter

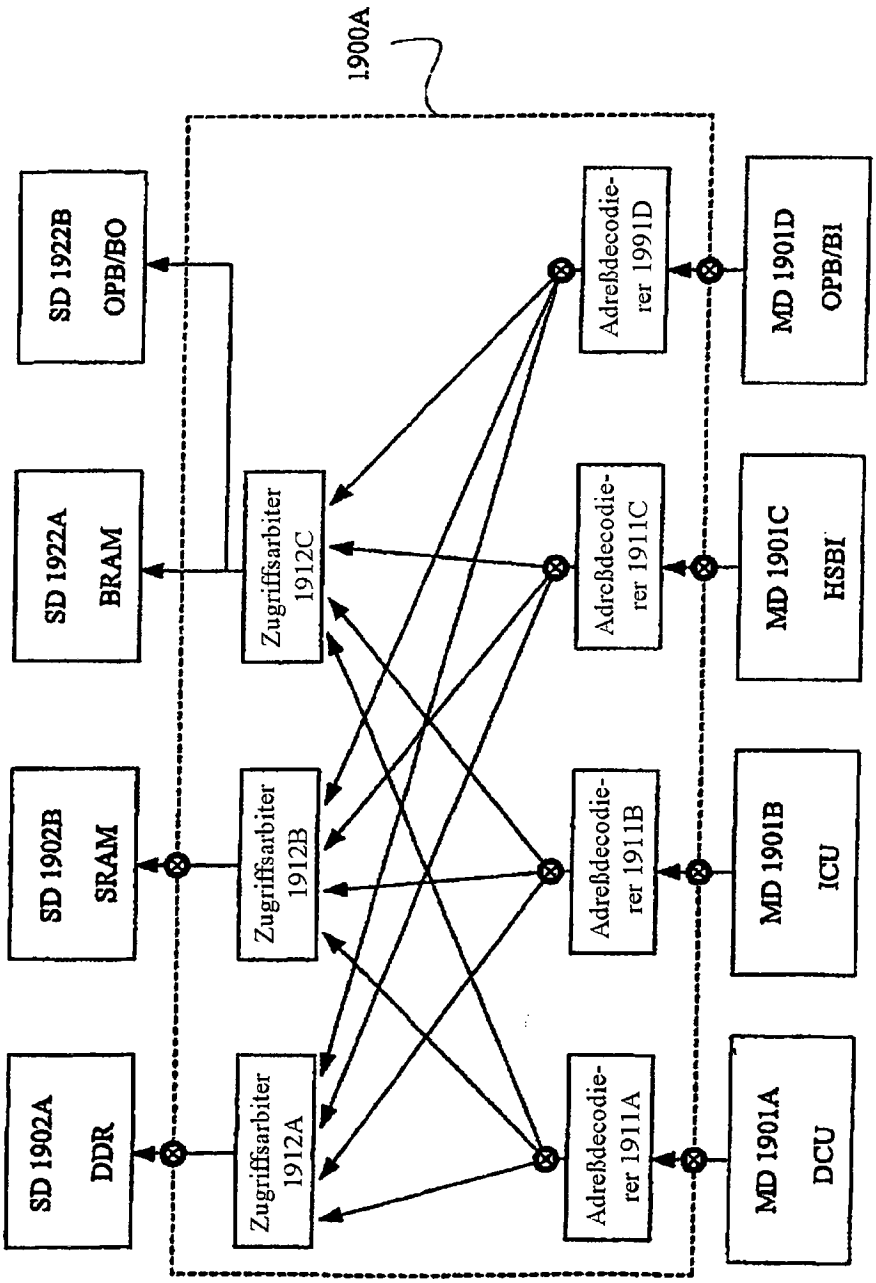
Figur 17


1800A
SD 1802D
OPB/BO
SD 1802C
BRAM
SD 1802B
SRAM
SD 1802A
DDR
Zugriffsarbiter
1803
MD 1801D
OPB/BI
MD 1801C
HSBI
MD 1801B
ICU
MD 1801A
DCU

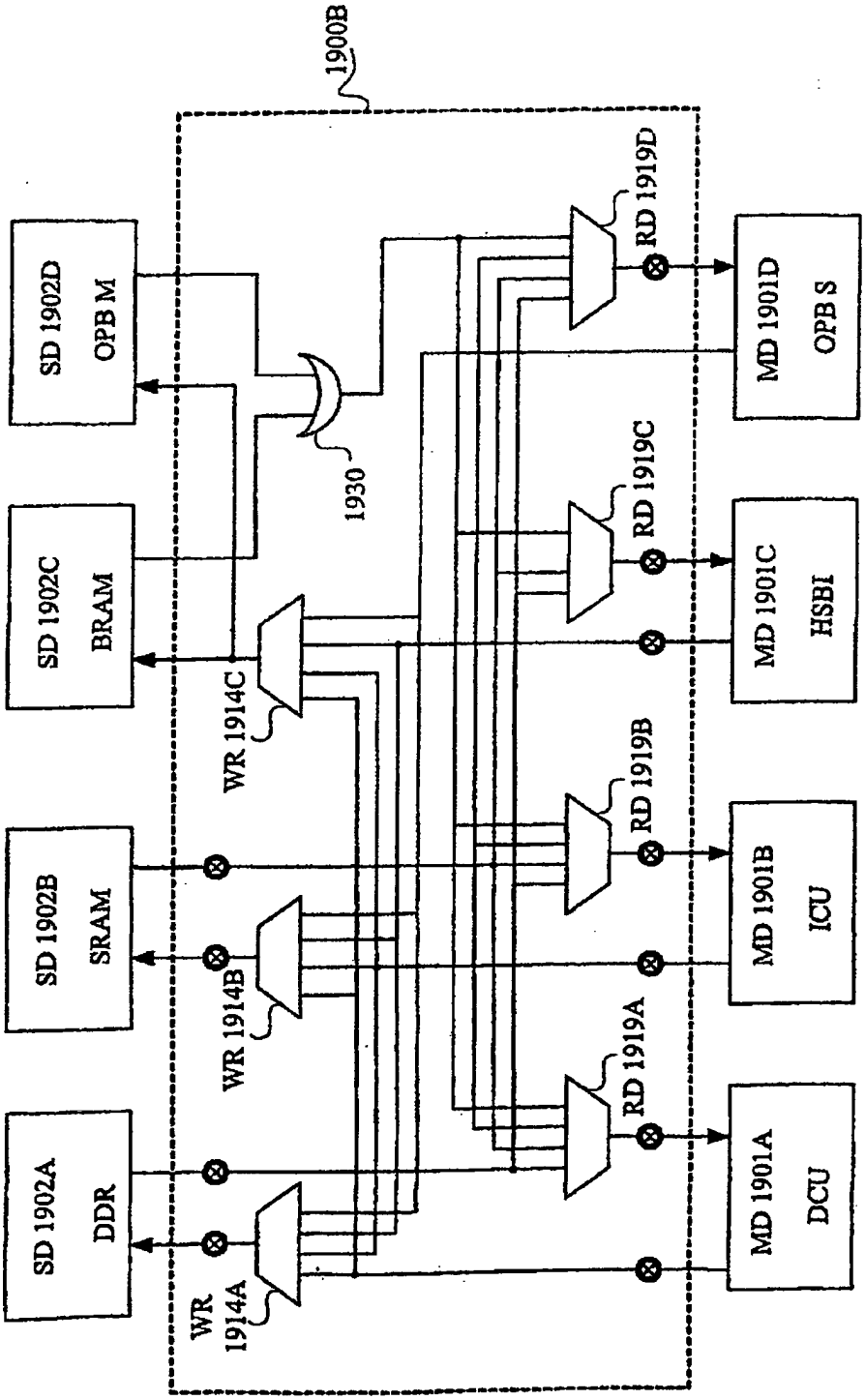
Figur 18A


1800B
SD 1802D OPB/BO
SD 1802C BRAM
SD 1802B SRAM
SD 1802A DDR
1805
WR 1804
RD 1806D
RD 1806C
RD 1806B
RD 1806A
MD 1801D OPB/BI
MD 1801C HSBI
MD 1801B ICU
MD 1801A DCU

Figur 18B


1900A
SD 1922B OPB/BO
SD 1922A BRAM
SD 1902B SRAM
SD 1902A DDR
Zugriffsarbiter 1912C
Zugriffsarbiter 1912B
Zugriffsarbiter 1912A
Adreßdecodierer 1991D
Adreßdecodierer 1911C
Adreßdecodierer 1911B
Adreßdecodierer 1911A
MD 1901D OPB/BI
MD 1901C HSBI
MD 1901B ICU
MD 1901A DCU

Figur 19A


1900B
SD 1902D OPB M
SD 1902C BRAM
SD 1902B SRAM
SD 1902A DDR
1930
WR 1914C
WR 1914B
WR 1914A
RD 1919D
RD 1919C
RD 1919B
RD 1919A
MD 1901D OPB S
MD 1901C HSBI
MD 1901B ICU
MD 1901A DCU

Figur 19B