Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Daniel Sabotta

FPGA basierter Entwurf eines bidirektionalen USB zu ISA Konverters für Kommandos einer CNC-Steuerung

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Daniel Sabotta

FPGA basierter Entwurf eines bidirektionalen USB zu ISA Konverters für Kommandos einer CNC-Steuerung

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Prof. Dr. rer nat Jürgen Reichardt
Zweitgutachter : Prof. Dr. Thomas Lehmann

Abgegeben am 17. Februar 2011

Daniel Sabotta

Thema der Bachelorthesis

FPGA basierter Entwurf eines bidirektionalen USB zu ISA Konverters für Kommandos einer CNC-Steuerung

Stichworte

FPGA, USB, ISA, CNC

Kurzzusammenfassung

Diese Arbeit beschreibt den Entwurf eines USB zu ISA Konverters für spezielle CNC-Kommandos der Sieb & Meyer CNC46 und CNC48. Die CNC46/48 kommunizieren direkt über den ISA-Bus mit dem Computer; dies soll in Zukunft über die USB-Schnittstelle geschehen. Wozu die USB-Umsetzter-Karte, die hier beschrieben wird, sorgen soll. Dabei werden die Daten über USB empfangen und per ISA-Bus an die CNC geschickt. Statusmeldungen von der CNC sollen von dieser Karte ausgelesen und über USB an den Computer geschickt werden.

Daniel Sabotta

Title of the paper

FPGA-based design of a bidirectional converter for USB to ISA commands of a CNC control

Keywords

FPGA, USB, ISA, CNC

Abstract

This paper describes the design of a USB to ISA converter for special CNC commands for a Sieb & Meyer CNC46 and CNC48. The CNC46/48 communicates directly through the ISA bus with the computer, which should happen in the future via the USB interface. What the USB card, which is described here, should do. The data is received via USB and is sent to the CNC via ISA bus. Status messages from the CNC should be read from the card and sent via USB to the computer.

Danksagung

An dieser Stelle möchte ich meinem betreuendem Professor Prof. Prof. Dr. rer nat Jürgen Reichardt und auch meinem Zweitgutachter Prof. Dr. Thomas Lehmann für die Unterstützung bei dem Erstellen dieser Arbeit danken.

Ebenfalls geht mein Dank an das Unternehmen Sieb & Meyer, vor allem an die Abteilung “CNC-Hardware”, bei dem ich für das Erstellen dieser Arbeit viel Unterstüzung bekommen habe.

Aber vorallem möchte ich mich bei meinen Eltern, meiner Schwester und meiner restlichen Familie für die Unterstützung während meiner gesamten Studienzeit bedanken.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung und Hintergrund

Diese Bachelorthesis wurde in dem Unternehmen Sieb & Meyer in Lüneburg erstellt und beschreibt die Umsetzung von Kommandos einer CNC-Steuerung von USB auf den ISA-Bus. In diesem Kapitel werden grundlegende Begriffe erläutert und ein Überblick über die Situation, sowie ein Ausblick auf die entstehende Hardware gegeben.

Sieb & Meyer wurde 1962 in Hamburg durch Reinhard Sieb und Johannes Meyer, mit dem Ziel innovative programmierbare Steuerungen für Leiterplattenbohrmaschinen zu fertigen, gegründet. Mittlerweile beschäftigt das Unternehmen, mit jetzigem Hauptsitz in Lüneburg, ca. 250 Mitarbeiter und bietet eine breite Produktpalette der CNC- und Antriebstechnik an.

1.1. Computerized Numerical Control, CNC

CNC Ist ein Akronym für Computerized Numerical Control (englisch für: „computergesteuert numerische Steuerung“). In der modernen Produktion werden CNC-Maschinen zur automatischen Fertigung von Werkstücken eingesetzt. Eine CNC-Maschine erlaubt es, ohne ständige Betreuung durch Personal Werkstücke mit hoher Präzision und Geschwindigkeit zu fertigen.

Das Werkstück wird zuvor mit einem CAD-Programm entworfen und ein CNC-Programm[1] entwickelt. Das CNC-Programm beschreibt die Arbeitsschritte, die die CNC ausführen muss. Um mit der CNC-Maschine möglichst jeden Punkt eines Werkstückes zu erreichen, werden drei senkrecht zueinander angeordnete Achsen (X, Y, und Z-Achse) benötigt. An einer dieser Achsen ist das Werkzeug, wie zum Beispiel ein Bohrer oder eine Fräse, senkrecht zum Werkstück befestigt.

Die Steuerung beziehungsweise Regelung dieser Achsen übernimmt die CNC. In manchen Fällen ist es auch notwendig, dass das Werkzeug nicht senkrecht, sondern in einem anderen Winkel zum Werkstück, steht. Hierfür bieten einige CNC-Maschinen drei weitere Achsen, mit der das Werkstück gedreht werden kann.

[1] CAD steht für: „computer-aided design“ („computergestützte Konstruktion)“ und ist ein Hilfsmittel zum Erstellen von technischen Zeichnungen.

In einem CNC-Programm sind die Koordinaten, die von der CNC-Maschine angefahren werden sollen, und weitere Parameter, wie zum Beispiel die Art und Dicke eines Bohrers hinterlegt. Die CNC hat die Aufgabe, aus diesen Koordinaten einen Weg für das Werkzeug zu interpolieren. Bei der Regelung einer CNC-Maschine werden, um eine möglichst hohe Genauigkeit zu gewährleisten, die aktuellen Positionen der Achsen kontrolliert und gegebenenfalls korrigiert. Außerdem steuert die CNC auch Komponenten, wie zum Beispiel die Kühlmittelzufuhr für Bohrer und Fräsen. CNC-Maschinen finden überall dort Verwendung, wo eine präzise und automatisierte Produktion gewünscht ist.

1.1.1. CNC-Maschinen für die Leiterplattenfertigung

Die Sieb & Meyer CNCs werden zum Großteil von Maschinenherstellern eingesetzt, die CNC-Maschinen für die Produktion von Leiterplatten herstellen. Sieb & Meyer stellt nicht die komplette CNC-Maschine sondern „nur“ die CNC-Steuerungen und die Antriebe für die Motoren her. In einer momentan typischen Leiterplattenbohrmaschine sind sechs Motorspindeln[2] verbaut. An jede dieser Motorspindeln kann ein Werkzeug befestigt werden. So können bis zu sechs identische Leiterplatten gleichzeitig gebohrt werden. Die Motorspindel mit dem Werkzeug ist senkrecht zu der Leiterplatte montiert. Mit Hilfe zweier Motoren (oft Linearmotoren) kann diese Motorspindel überall über dem Werkstück (der Leiterplatte) positioniert (X- und Y-Achse) werden. Ein weiterer Motor ermöglicht das Herauf- und Hinabfahren (Z-Achse) der Motorspindel.

Die Werkzeuge (Bohrer und Fräsen) sind innerhalb der Maschine angebracht, so dass die Maschine diese selbstständig wechseln kann. Bei einem Werkzeugwechsel wird geprüft, ob die Qualität des Werkzeugs innerhalb von Toleranzen liegt. Hat die Maschine das Werkzeug gewählt, wird der Punkt auf der Leiterplatte angefahren (X- und Y- Achse). Anschließend wird die Motorspindel mit dem Werkzeug heruntergefahren und so das Loch gebohrt. Bei diesem Vorgang treten hohe Beschleunigungen der Achsen auf. Damit die Maschine bei den Bewegungen der einzelnen Achsen einen möglichst festen Stand hat, werden die Gestelle oft aus Granit-Blöcken gefertigt.

Die Motoren der Achsen werden mit Servoverstärkern angetrieben. Über die Sensoren der Motoren, kann ein Servoverstärker die Position des Motors ermitteln. Um die Motorspindel anzutreiben wird ein Frequenzumrichter benötigt, der aus der vorhandenen Versorgungsspannung das benötigte Drehfeld generiert. Die Geschwindigkeit der Motorspindel ist von der Frequenz des Drehfeldes abhängig.

[2] Motorspindeln sind Motoren, die speziell für Werkzeugmaschinen gefertigt werden. Sie besitzen eine Halterung für Bohrer/Fräsen.

1.1.2. CNC46/48

Die CNC46 und die CNC48 unterscheiden sich nur darin, dass die CNC46 einen eingeschränkten Funktionsumfang bietet. Beide CNC-Steuerungen bestehen aus einer CNC-Logik mit integriertem Computer für die Datenein- und ausgabe, sowie ein CNC-Power, was die Versorgungsspannungen für die CNC-Logik und dem Computer liefert. Ein zusätzliches Power liefert die Spannungen für die Antriebe der Motoren. Es sind drei Servoverstärker notwendig, für jede Achse (X-, Y- und Z-Achse) einer, und für die Motorspindel ein Frequenzumrichter. Diese einzelnen Komponenten sind in einem Gehäuse verbaut und kommunizieren über Servolink[3] miteinander. Die CNC-Logik steuert den Ablauf des Programms und übernimmt die Regelung der Position der Achsen. Den Servoverstärkern und dem Frequenzumrichter wird die Solldrehzahl über Servolink übermittelt. Des Weiteren verarbeitet und steuert die CNC-Logik externe Ein- und Ausgabesignale, wie zum Beispiel Verriegelungen von Türen oder das Vorhandensein von Druckluft. Die Servoverstärker regeln die Motordrehzahl und –richtung. Ebenso verarbeiten sie die Motorsensoren und liefern der CNC-Logik die aktuelle Position. Der Frequenzumrichter treibt die Motorspindel an. Der Computer dient lediglich der Benutzerein- und ausgabe von CNC-Programmen, Konfigurationsdateien sowie der Visualisierung des Fortschrittes des Arbeitsvorganges.

1.1.2.1. Speicherverwaltung der CNC46/48

Auf der CNC-CPU-I/O-Karte befindet sich ein 64KByte großer DP-RAM[4] Speicher. Über dieses DP-RAM kommuniziert die CNC mit dem Computer. Auf dieses DP-RAM kann die CPU der CNC direkt und der Computer über den ISA-Bus zugreifen (siehe Abbildung 1.1). Dabei übernimmt ein CPLD (Lattice M4A5-64) die Priorisierung der Zugriffe auf das RAM. Das RAM hat eine Datenwortbreite von 16 Bit. Dazu wurden in Hardware zwei 8-Bit-RAM-Bausteine mit je 32KByte benutzt (Samsung KM68257). Jedes Byte besitzt dabei eine eigene Adresse. Es ist aber möglich, auf zwei Bytes (16 Bit) auf einmal zuzugreifen. Die RAM-Bausteine besitzen einen Chip Select (EN_LL für das Low und EN_H für das Highbyte), einen Output Enable (RE) und einen Write Enable (WE) Eingang. Der Chip ist nur aktiv, wenn das Chip Select Signal gesetzt ist. Die Daten der adressierten Adresse werden mit Output Enable aus dem RAM gelesen und mit Write Enable in das RAM geschrieben. An die Adresseingänge der Bausteine führen die Adressleitungen A15 bis A01. Ob nun das High- oder Lowbyte oder beide ausgelesen beziehungsweise geschrieben werden sollen, wird durch die RAM-Logik im CPLD mit den Output- und Write Enable Leitungen entschieden. Das Highbyte liegt an den geraden und das Lowbyte an den ungeraden Adressen. Für die Kommunikation über den ISA-Bus wird die Chip Select Leitung gesetzt, wenn eine Adresse zwischen 851968d

[3] Servolink ist ein Bus von Sieb & Meyer für die Kommunikation innerhalb einer CNC-Maschine.
[4] (Dual-Port-RAM) - ermöglicht Zugriff auf das RAM von mehreren Quellen aus

(D0000h) und 884735d (D7FFFh) anliegt. Wird über den ISA-Bus eine Leseanforderung gesendet, setzt die RAM-Logik die Output Enable Leitung. Das bedeutet, dass bei einem Lesezugriff immer auf die kompletten 16 Bit zugegriffen wird, somit spielt das niederwertigste Adressbit bei der Leseoperation keine Rolle. Bei einer Schreibanforderung wird eine oder es werden beide Chip Enable Leitungen gesetzt. Das Highbyte wird nur dann geschrieben, wenn eine gerade Adresse (niederwertigstes Bit SA0 = 0) anliegt. Soll das Lowbyte geschrieben werden, muss ein 16-Bit-Zugriff über den ISA-Bus angefordert werden. Dementsprechend muss also eine gerade Adresse anliegen, wenn sowohl das Low- als auch das Highbyte geschrieben werden soll.

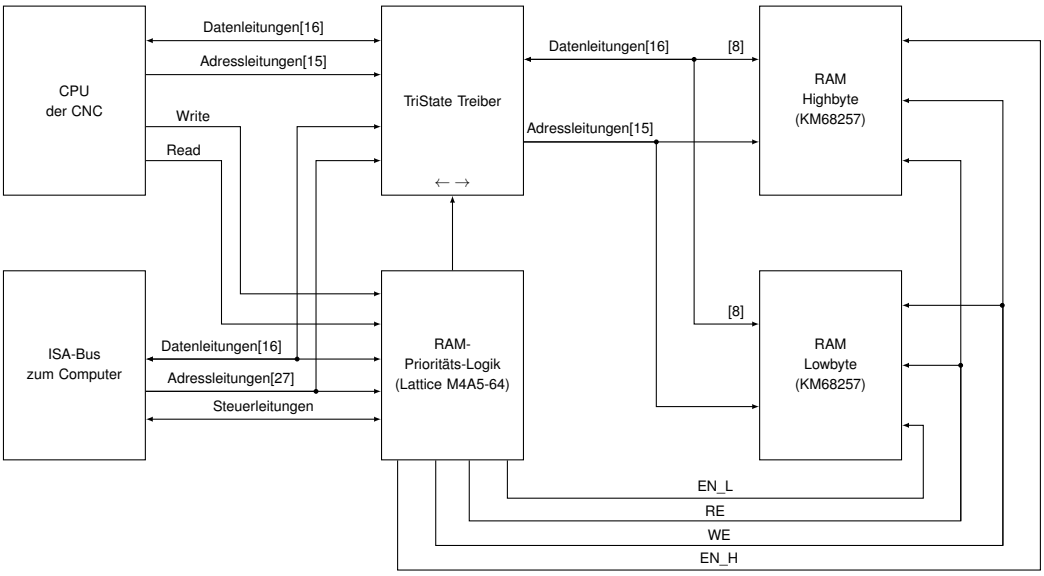


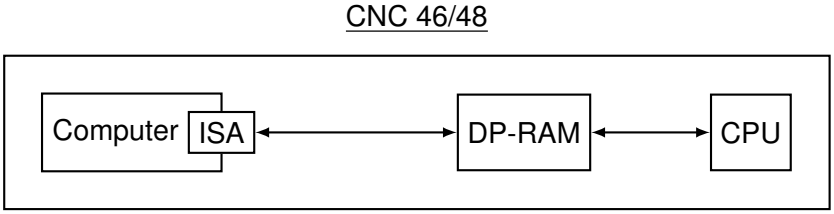
Abbildung 1.1.: Architektur der Speicherverwaltung der CNC46/48. Die CPU der CNC und der Computer teilen sich ein RAM. Die RAM-Prioritäts-Logik steuert den Zugriff auf dieses.

1.1.2.2. Schwachpunkte der CNC46/48

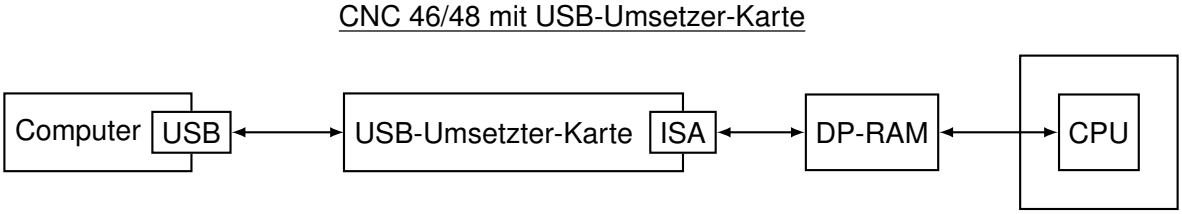
Die CNC46 beziehungsweise CNC48 CNC-Steuerungen haben sich seit 19XX mit einer hohen Ausfallsicherheit bewährt. Mittlerweile kann der in der Steuerung verbaute Computer nicht mehr produziert werden, da unverzichtbare Komponenten nicht mehr hergestellt werden. So ist es nicht möglich, die Steuerung bei Ausfall dieses Computers weiter zu betreiben. Ein weiterer Nachteil der Lösung mit integriertem Computer ist der hohe Preis dieses Gerätes, was auf die relativ niedrige Stückzahl zurückzuführen ist. Da es heutzutage schon möglich ist, einen Computer mit mehr Rechenleistung in einem Supermarkt zu kaufen, wirkt dieses Modell des integrierten Computers als stark veraltet. Um den Betrieb der CNC46/48

weiterhin zu gewährleisten ist es notwendig, den integrierten Computer zu ersetzen. Dies soll in Zukunft mit der USB-Umsetzer-Karte erfolgen.

Dazu soll die USB-Umsetzter-Karte anstelle des Computers in die CNC verbaut werden. Ein externer Computer, mit entsprechender Software, soll dann über USB die Aufgaben des internen Computers übernehmen (siehe Abbildung 1.2).



(a) Ohne USB-Umsetzter-Karte



(b) Mit USB-Umsezter-Karte

Abbildung 1.2.: Vergleich der Architektur der Kommunikation zwischen Computer und CNC-CPU.

1.2. ISA-Bus

1.2.1. Allgemein

Die Industry Standard Architecture (ISA) ist ein Computerbus, der Mitte der 1980er Verwendung in IBM-kompatiblen PCs fand. Er ergänzte die zuvor häufig verwendete XT-Bus-Architektur um weitere 8 auf 16 Bit. Um die Kompatibilität weiterhin zu gewährleisten, wurden die weiteren Anschlüsse so angebracht, dass auch XT-Bus-Geräte (der XT-Bus wird oft auch 8-Bit-ISA-Bus genannt) weiterhin mit dem ISA-Bus arbeiten können. Der Mitte der 1990er entwickelte PCI-Bus löste den ISA-Bus auf den Mainboards der Computer immer mehr ab. Heutzutage besitzt so gut wie kein PC mehr einen ISA-Steckplatz auf dem Mainboard. In der Industrie findet man in manchen Computersystemen allerdings auch heute noch den ISA-Bus, teils aufgrund seiner langen Produktlebensdauer.

Der ISA-Bus ermöglicht DMA (“Direct Memory Access”), Speicherdirektzugriff. Des weiteren ermöglicht der ISA-Bus der Peripherie „Interrupt Requests“ (Englisch für „Unterbrechungsanforderung“) zu senden. Auch Busmastering wird von dem ISA-Bus ermöglicht, dies ist eine Methode, die es der Peripherie kurzzeitig ermöglicht die Funktion des Busmasters zu übernehmen.

1.2.2. Signalbeschreibung

1.2.2.1. Adress- und Datenleitungen

AEN	**Address Enable** line wird aktiv gesetzt, wenn ein DMA-Zugriff stattfindet.
BALE	**Bus Address Latch Enable** („Bus Adressverriegelung“) wird von dem Busmaster gesendet, wenn die Adresse gültig auf dem Bus anliegt. Ebenso wird BALE aktiv, wenn eine Peripherie oder die DMA-Logik die Kontrolle über den Bus übernommen hat.
SA<19:0>	**Address** lines („Adressleitungen“) werden vom Busmaster getrieben. Mit diesen 20 Adressleitungen wird der untere 1MB Speicherbereich adressiert.
LA<23:17>	**Latched Address** lines („verriegelte Adressleitungen“) werden benötigt, um den 16MB Speicherbereich zu adressieren. Sie werden vom Busmaster oder von der DMA-Logik gesetzt.
$\overline{\text{SBHE}}$	**System High Byte Enable** („System Highbyte aktiviert”) wird von dem Busmaster gesetzt, wenn ein 16-Bit-Zugriff stattfinden soll, also die Datenleitungen SD<15:8> benutzt werden sollen.
SD<15:0>	**Data** lines („Datenleitungen“) dienen zur Übertragung der Daten. Die Leitungen 0 – 7 werden für 8-Bit-Zugriffe und die Leitungen 0 – 15 für 16-Bit-Zugriffe verwendet. Wobei SD0 das niederwertigste Bit repräsentiert.

1.2.2.2. Zugriffskontrollleitungen

$\overline{\text{MEMR}}$	**Memory Read** line („Lesender Speicherzugriff“) wird von dem Busmaster oder der DMA-Logik gesetzt, um eine Speicherquelle aufzufordern, die adressierten Daten auf den Bus zu legen.
$\overline{\text{SMEMR}}$	**System Memory Read** line („Lesender Systemspeicherzugriff”) wird vom Busmaster aktiv geschaltet, wenn ein Lesender Speicherzugriff erfolgt und dazu müssen die Adressleitungen LA<23:20> die untersten 1MB adressieren.

$\overline{\text{MEMW}}$	**Memory Write** line („Schreibender Speicherzugriff“) wird vom Busmaster gesetzt, wenn eine Speicherquelle Daten von dem Datenbus übernehmen soll.
$\overline{\text{SMEMW}}$	**System Memory Write** line („Schreibender Systemspeicherzugriff”) wird vom Busmaster aktiv geschaltet, wenn ein Schreibender Speicherzugriff auf die untersten 1MB erfolgt (von LA<23:20> adressiert).
$\overline{\text{IOR}}$	**I/O Read** line („Lesender Ein-/Ausgabezugriff“) wird von dem Busmaster oder der DMA-Logik auf aktiv geschaltet, wenn eine Ein- beziehungsweise Ausgabequelle Daten auf dem Bus zur Verfügung stellen soll.
$\overline{\text{IOW}}$	**I/O Write** line („Schreibender Ein-/Ausgabezugriff“) wird vom Busmaster oder der DMA-Logik gesetzt, wenn eine Ein- beziehungsweise Ausgabequelle Daten von dem Bus übernehmen soll.
$\overline{\text{MEMCS16}}$	**Memory Chip Select 16** line wird von einer Speicherquelle gesetzt, um zu signalisieren, dass es sich um eine 16 Datenbits fähige ISA-Quelle handelt. Wenn dieses Signal gesetzt ist, ist es dem Busmaster erlaubt, kürzere Wartezeiten zwischen den Signalwechseln einzuhalten.
$\overline{\text{IOCS16}}$	**I/O Chip Select 16** line wird von einer Ein- beziehungsweise Ausgabequelle gesetzt, um zu signalisieren, dass sie einen 16 Datenbit-Zugriff unterstützt. Wie bei dem $\overline{\text{MEMCS16}}$ Signal wird es damit dem Busmaster ermöglicht kurze Wartezeiten zwischen den Signalwechseln einzuhalten.
IOCHRDY	**I/O Channel Ready** line wird von den Slaves gesetzt, wenn sie zusätzliche Zeit benötigen, um die Anfrage von dem Busmaster umzusetzen.
$\overline{\text{SRDY}}$	**Synchronous Ready** line wird von einem Slave, auf den zugegriffen wird, gesetzt, um zu signalisieren, dass seine Zugriffszeit kürzer als der Standard ist.

1.2.2.3. Buskontrollleitungen

$\overline{\text{REFRESH}}$	**Memory Refresh** wird von einem, extra dafür vorgesehenen, Refresh-Controller gesetzt, um einen Refresh-Zyklus anzukündigen. In dem Refresh-Zyklus werden die flüchtigen Speicherzellen neu geladen, um Datenverlust zu verhindern.
$\overline{\text{MASTER16}}$	**$\overline{\text{MASTER16}}$** line wird nur von einem alternativen Busmaster gesetzt, der von der DMA-Logik berechtigt wurde, den Busmaster zu übernehmen.
$\overline{\text{IOCHK}}$	**I/O Channel Check** line wird von einer beliebigen Quelle aktiv gesetzt, wenn ein allgemeiner Fehlerfall aufgetreten ist, der keiner speziellen Behandlung bedarf.
RESET	**Reset** line wird von dem Busmaster aktiv gesetzt. Alle Bus-Teilnehmer, die ein Reset-Signal empfangen, müssen unverzüglich alle Ausgangstreiber in den Tri-State-Modus schalten und in den Reset-Zustand zurückkehren.

BCLK	**System Bus Clock** line ist ein Takt, vom Busmaster. Der Takt hat ein Tastverhältnis von 50% +/- 5% bei einer Frequenz zwischen 6MHz und 8MHz (+/- 500ppm)
OSC	**Oscillator** line ist ein weiterer Takt, der vom Busmaster stammt. Dieser Takt hat eine Frequenz von 14,31818MHz (+/- 500ppm) bei einem Tastverhältnis von 45% bis 55%. Der Takt ist zu keinem anderen Signal des ISA-Busses synchron.

1.2.2.4. Intrrupts

IRQx	**Interrupt Request** lines erlauben es des Slaves, den Busmaster um eine Interrupt-Behandlung zu bitten. Es stehen IRQ3 bis IRQ7 und IRQ9 bis IRQ14 zur Verfügung.

1.2.2.5. Direct Memory Access (Speicherdirektzugriff)

DRQx	**Direct Memory Access Request** lines können von Ein- und Ausgabegeräten aktiv gesetzt werden, um einen Speicherdirektzugriff bei der DMA-Logik anzufordern. Es stehen DRQ0 bis DRQ3 und DRQ5 bis DRQ7 zur Verfügung.
$\overline{\text{DACKx}}$	**Direct Memory Access Acknowledge** lines werden von der DMA-Logik geschaltet. Um dem Ein- beziehungsweise Ausgabegerät, das einen Speicherdirektzugriff angefordert hat, zu signalisieren, dass es den Speicherdirektzugriff durchführen kann, wird dieses Signal aktiv geschaltet.
TC	**Terminal Count** line wird von der DMA-Logik gesetzt, wenn ein Speicherdirektzugriff abgeschlossen ist.

1.2.3. Der ISA-Bus in der CNC46/48

Für die Kommunikation der CNC-Logik-Karte mit dem Computer werden nicht alle Signale des ISA-Standards benötigt. Die für die Kommunikation relevanten Signale sind der Tabelle 1.6 zu entnehmen.

Tabelle 1.6.: Von der CNC46/48 benutzten ISA-Signale

Bezeichnung im ISA-Standard	VHDL-Bezeichner	Beschreibung
SD	PC_D	Datenleitungen (16-Bit-Vektor).
SA	PC_A	Adressleitungen (19-Bit-Vektor).
LA	PC_LA	verriegelte Adressleitungen (7-Bit-Vektor) addressieren zusammen mit PC_A-Signalen das RAM der CNC.
IOCHRDY	IO_CHRDY	Über I/O Channel Ready bestätigt die CNC eine erfolgreiche Übertragung.
$\overline{\text{MEMCS16}}$	nPC_MEM16	Mit dem Signal Memory Chip Select 16 bestätigt die CNC einen 16-Bit-Zugriff.
$\overline{\text{SBHE}}$	nPC_SBHE	Mit System High Byte Enable kann von der USB-umsetzter-Karte ein 16-Bit-Zugriff angefordert werden.
$\overline{\text{SMEMR}}$	nPC_SMEM_R	System memory Read signalisiert der CNC einen Lesezugriff.
$\overline{\text{SMEMW}}$	nPC_SMEM_W	System memory Write signalisiert der CNC einen Schreibzugriff.
AEN	PC_AEN	Address Enable signalisiert der CNC, dass die angelegte Adresse gültig ist.

In Abbildung 1.3 ist zu sehen, wie die Signale des ISA-Busses gesetzt werden müssen, um einen 16-Bit-Schreibzugriff durchzuführen.

1. • Adresse an PC_LA und PC_A anlegen.
   • Daten auf die Datenleitungen PC_D legen.
   • nPC_SBHE Null setzen, um einen 16-Bit-Zugriff anzufordern.
   • Adresse durch Setzen des PC_AEN-Signals bestätigen.
2. nPC_MEM16 wird von der CNC, als Bestätigung des 16-Bit-Zugriffes, auf Null gesetzt.
3. Das Schreiben wird durch das Null-Setzten von nPC_SMEM_W eingeleitet.
4. Mit dem IO_CHRDY-Signal bestätigt die CNC, dass sie die Daten übernommen hat.
5. Jetzt kann nPC_SMEM_W wieder Eins gesetzt werden.
6. Daraufhin setzt die CNC IO_CHRDY wieder Null.
7. Nach einer kurzen Wartezeit können die restlichen Signale ebenfalls in den ursprünglichen Zustand zurückkehren.
8. Die CNC bestätigt das Ende des 16-Bit-Zugriffes, indem das Signal nPC_MEM16 wieder Eins gesetzt wird.

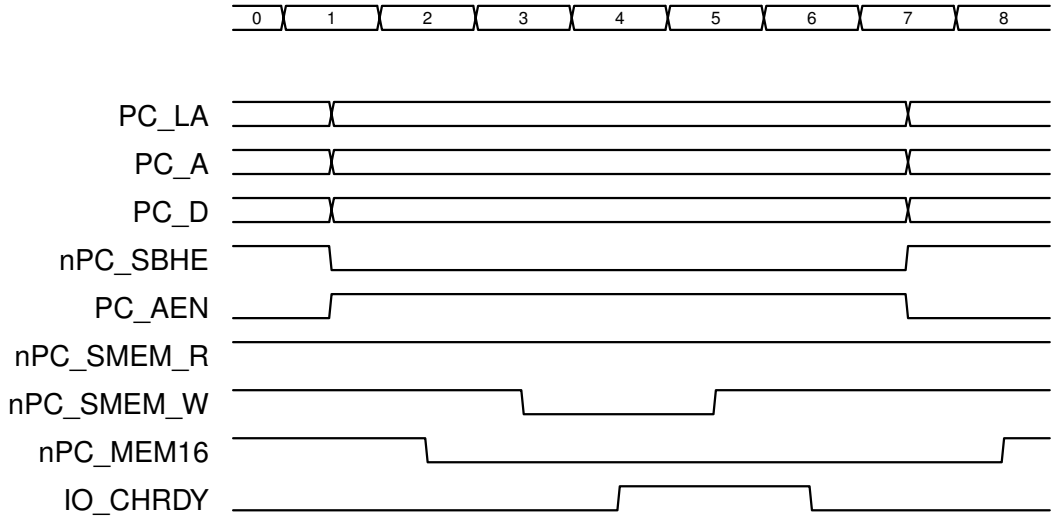


Abbildung 1.3.: Abfolge der Signale für die ISA-Kommunikation mit der CNC (16-Bit-Schreibzugriff).

In welcher Reihenfolge die Signale des ISA-Busses für einen Lesezugriff gesetzt werden müssen, ist in Abbildung 1.4 zu sehen:

1. • Adresse an PC_LA und PC_A anlegen.
   • nPC_SBHE Null setzen, um einen 16-Bit-Zugriff anzufordern.
   • Adresse durch Setzen des PC_AEN-Signals bestätigen.
2. nPC_MEM16 wird von der CNC, als Bestätigung des 16-Bit-Zugriffes, auf Null gesetzt.
3. Durch das Null-Setzen von nPC_SMEM_R wird die CNC aufgefordert die Daten auf den Bus zu legen.
4. Mit dem IO_CHRDY-Signal bestätigt die CNC, dass sie die Daten (PC_D) auf den Bus gelegt hat.
5. Jetzt kann nPC_SMEM_R wieder Eins gesetzt werden.
6. Daraufhin setzt die CNC IO_CHRDY wieder Null und löscht die Daten (PC_D) von dem Bus.
7. Nach einer kurzen Wartezeit können die restlichen Signale ebenfalls in den ursprünglichen Zustand zurückkehren.
8. Die CNC bestätigt das Ende des 16-Bit-Zugriffes, indem das Signal nPC_MEM16 wieder Eins gesetzt wird.

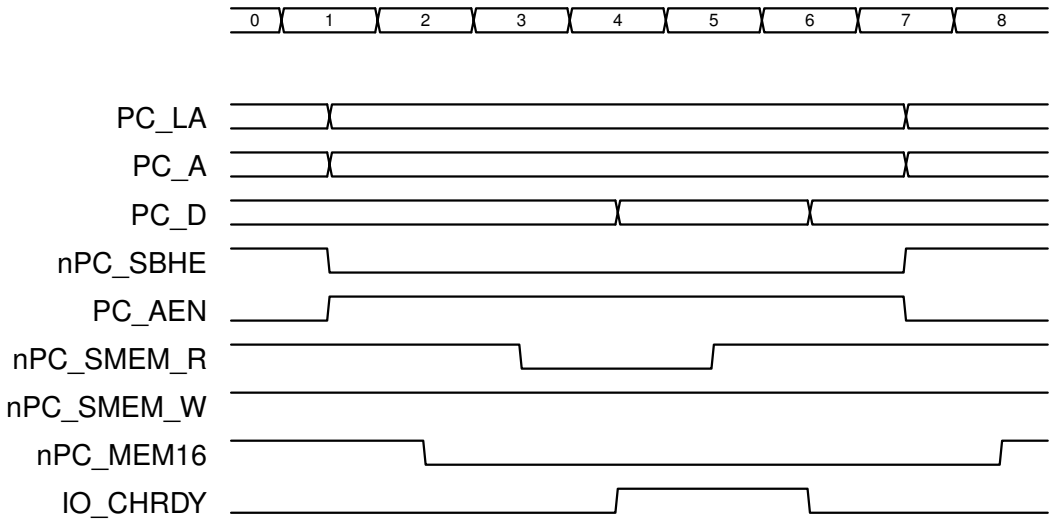


Abbildung 1.4.: Abfolge der Signale für die ISA-Kommunikation mit der CNC (16-Bit-Lesezugriff).

1.3. Universal Serial Bus, USB

USB ist ein bitserieller Bus, bei dem die Übertragung der Daten symmetrisch auf zwei verdrillten Datenleitungen (D+ und D-) erfolgt. Die Erste USB-Version (USB 1.0) wurde von Intel 1996 entwickelt und ermöglichte eine maximale Übertragungsgeschwindigkeit von 1,5Mbit/s („Low-Speed“). Anschließend folgte USB 1.1, die weitestgehend Fehlerkorrekturen der USB 1.0-Spezifikation enthält. Des Weiteren wurde eine weitere Übertragungsvariante eingeführt („Full-Speed“), mit der eine Übertragungsgeschwindigkeit von maximal 12Mbit/s möglich ist. Im Jahr 2000 wurde dann die Spezifikation für USB 2.0 vorgestellt. Diese neue Spezifikation ermöglicht eine maximale Übertragungsrate von 480Mbit/s. 2008 wurde die derzeit neuste Spezifikation veröffentlicht, USB 3.0. Hierbei kommen zwei weitere Datenleitungspaare zum Einsatz; damit sind Übertragungsgeschwindigkeiten von bis zu 5Gbit/s möglich.

USB ist mittlerweile in fast allen Computern verbaut und wird auch von den meisten Betriebssystemen unterstützt. Bei USB darf es nur einen Busmaster geben, den Host. Dieser steuert die komplette Busaktivität. Neben den beiden Datenleitungen besitzt USB noch eine Masseleitung und eine Leitung, auf der eine 5 Volt Spannung übertragen wird, die die USB-Geräte als Versorgungsspannung benutzen können. Diese Versorgungsspannung ist pro Port bis maximal 500 Milliampere belastbar.

Die Datenübertragung zwischen den USB-Geräten erfolgt über sogenannte Endpunkte. Diesen Endpunkten wird eine Adresse und eine Richtung (in der die Daten gesendet werden) zugewiesen. Dabei kann ein USB-Gerät mehrere Endpunkte (aber mindestens einen) besitzen. Beim Anstecken eines USB-Gerätes an den Bus wird über den (immer vorhandenen) Endpunkt 0 die Adresse des Gerätes und der Endpunkte, sowie die Konfiguration übermittelt (Control-Transfer). Der Endpunkt 0 ist der einzige, der bidirektional arbeitet. Soll also eine bidirektionale Datenübertragung erfolgen, müssen neben dem Endpunkt 0 noch mindestens zwei weitere Endpunkte benutzt werden (einer für ein- und einer für ausgehende Daten).

Zur Kommunikation über USB dienen Datenpakete. In diesen Datenpaketen werden Steuerinformationen, Adressierungen und die Daten übertragen. Aus diesen Datenpaketen holt sich das adressierte USB-Gerät die Daten; allerdings wird es an jedes Gerät des Busses geschickt.

Neben dem Contorl-Transfer gibt es drei weitere Transferarten um Daten über USB zu versenden. Den Isochronen-, den Interrupt- und den Bulk-Transfer. Der Bulk-Transfer ist für große Datenmengen optimiert. Da dieser Transferart eine niedrige Priorität zugeordnet ist, eignet er sich nicht für zeitkritische Übertragungen. Tritt bei der Übertragung ein Fehler auf, so wird das fehlerhafte Paket maximal dreimal wiederholt gesendet. Die Fehlererkennung geschieht mit Hilfe einer CRC16-Prüfsumme. Geeignet ist diese Transferart zum Beispiel für USB-Massenspeicher. Der Isochrone-Transfer garantiert eine feste Bandbreite, und kann

somit für zeitkritische Übertragungen eingesetzt werden. Kann der USB-Host die angeforderte Bandbreite allerdings nicht zur Verfügung stellen, so kommt die Verbindung nicht zu Stande. Auch diese Transferart wird mit einer CRC16-Prüfsumme gesichert. Allerdings wird in einem Fehlerfall die Übertragung nicht wiederholt; das Paket wird verworfen. Diese Art wird beispielsweise bei externen Soundkarten verwendet. Der Interrupt-Transfer ist nur für die Übertragung kleiner Datenmengen geeignet. In zuvor festgelegten Zeitintervallen wird der Endpunkt von dem USB-Host abgefragt (Polling). Anwendung findet diese Transferart zum Beispiel in Computer-Mäusen und -Tastaturen.

1.4. Field Programmable Gate Array, FPGA

Vor dem Zeitalter der programmierbaren Logik mussten Hardwareentwickler auf Standardbauteile, wie zum Beispiel Und- und Oder-Gatter, oder auf teure ASIC Technologien zurückgreifen. Heutzutage können FPGAs eingesetzt werden, die den Hardwareaufwand (im Vergleich zu Standardbauteilen) stark verringern.

FPGAs bestehen zum Großteil aus Wahrheitstabellen (Look-Up-Tables, kurz: LUTs) und Ein-Bit-Registern (D-Flipflops). Die Wahrheitstabellen besitzen oft zwischen vier (zum Beispiel Virtex-II-FPGAs) und sechs (zum Beispiel Virtex-5-FPGAs) Eingängen. So lassen sich alle erdenklichen Vier-Bit- beziehungsweise Sechs-Bit-Verknüpfungen mit einer Wahrheitstabelle realisieren. Häufig ist der Wahrheitstabelle ein D-Flipflop nachgeschaltet.

Kombinationen aus mehreren Wahrheitstabellen und mehreren Flipflops werden Slice (Stück) genannt (oft zwei Wahrheitstabellen und zwei Flipflops). Um komplexe Schaltungen zu realisieren, können die Wahrheitstabellen und Flipflops über programmierbare Schaltmatrizen miteinander verbunden werden.

Die Schaltmatrizen und Wahrheitstabellen basieren meist auf der SRAM-Technologie (Static random-access memory), die über ein flüchtiges Speicherverhalten verfügen. Dies führt dazu, dass die Konfiguration des FPGAs nach Abschalten der Betriebsspannung nicht mehr vorhanden ist. In der Regel werden deshalb externe nichtflüchtige Speicherbausteine an das FPGA angeschlossen und bei dem Anschalten der Betriebsspannung die Konfiguration neu geladen. Einige FPGAs besitzen auch einen internen, nichtflüchtigen Speicher, so dass keine externe Beschaltung notwendig ist (siehe [Lattice Semiconductor Corp. (2009)]).

Die Funktionsweise der FPGA lässt sich mit einer Hardwarebeschreibungssprache, wie zum Beispiel VHDL oder Verilog, beschreiben. Ein Synthesetool erstellt aus der Hardwarebeschreibungssprache unter Nutzung der im FPGA vorhandenen Ressourcen eine Netzliste. In der Netzliste sind die Verbindungen zwischen den Komponenten des FPGAs beschrieben. Eingesetzt werden FPGAs unter anderem als digitale Filter, für schnelle Fouriertransformationen und zur Lösung mathematischer Aufgaben. Der Aufgabenbereich ist sehr vielfältig,

weshalb eine Fülle von FPGA-Bausteinen, mit den unterschiedlichsten Spezialisierungen, existieren.

1.5. Sieb & Meyer CNC-Kommandos

Die CNC kommuniziert mit einem Computer, um CNC-Programme, Konfigurationen und Statusmeldungen zu übertragen. Dazu wurden ursprünglich von dem Computer nacheinander 8- beziehungsweise 16-Bit große Datenblöcke über den ISA-Bus in das DP-RAM der CNC geschrieben, bis alle Informationen übertragen wurden. Um die Kommunikation über USB zu ermöglichen werden diese Daten in sogenannte USB-Commands beziehungsweise USB-Results verpackt.

Das Senden eines Kommandos geht grundsätzlich von dem Computer aus – er sendet ein USB-Command. Auf dieses USB-Command muss die CNC mit einer passenden Nachricht (USB-Result) antworten. Dabei sind die Kommandos in drei Arten unterteilt, die aus mehreren 16-Bit Blöcken bestehen: Der Leseblock, der Schreibblock und das Sendekommando.

Der Leseblock dient dazu Daten direkt aus dem RAM auszulesen und der Sendeblock, um Daten direkt in das RAM zu schreiben. Beide Nachrichtentypen werden im normalen CNC-Betrieb nicht gebraucht. Die eigentliche Kommunikation erfolgt mit dem Sendekommando, wobei diverse Daten an bestimmte Adressen in das RAM geschrieben werden.

USB-Command
Bei allen CNC-Nachrichten wird als erstes der Modus übertragen, dieser identifiziert den Typ der Nachricht. Dabei steht eine eins für ein Sendekommando, eine zwei für ein Lese- und eine drei für einen Schreibblock. Die folgenden 16 Bit übertragen die Blocknummer, welche mit jeder Nachricht, die der PC sendet, erhöht wird. Damit identifiziert die Blocknummer jedes Kommando eindeutig.

USB-Result
Als USB-Results werden die Bestätigungen der CNC auf die USB-Commands bezeichnet. Diese beginnen mit der Blocknummer, damit der PC dem USB-Result ein USB-Command zuweisen kann. Anschließend folgen weitere Daten, die vom Nachrichtentyp abhängen.

1.5.1. Der Leseblock

In dem USB-Command eines Leseblocks folgt der Blocknummer die Adresse im RAM, an der mit dem Lesen angefangen werden soll. Anschließend wird die Länge übertragen, die angibt, wie viele Bytes gelesen werden sollen (siehe Tabelle 1.8).

Das USB-Result enthält neben der Blocknummer nur noch die Daten, die in dem angeforderten Adressbereich stehen (siehe Tabelle 1.9).

Tabelle 1.8.: Leseblock USB-Command (wird von dem Computer gesendet)

Offset in 2 Bytes	Beschreibung
0	Modus = 0x0002
1	Blocknummer
2	Adresse
3	Länge (Größe der Zieldaten)

Tabelle 1.9.: Leseblock USB-Result (wird von der USB-Umsetzer-Karte als Antwort gesendet)

Offset in 2 Bytes	Beschreibung
0	Blocknummer
1...	Zieldaten

1.5.2. Der Schreibeblock

Nach der Blocknummer wird, im USB-Command, die Adresse, an die die Daten in das RAM geschrieben werden sollen, übertragen. Danach kommt die Anzahl der zu schreibenden Bytes, gefolgt von den zu schreibenden Daten (die Quelldaten), dessen Anzahl sich nach der zuvor übertragenden Anzahl richtet (siehe Tabelle 1.10).

Die Antwort (USB-Result) auf einen Schreibeblock enthält lediglich nur die Blocknummer (siehe Tabelle 1.11).

1.5.3. Das Sendekommando

Das Sendekommando ist die komplexeste und auch am meisten genutzte Form von CNC-Kommandos. In dem USB-Command folgt hier, nach der Blocknummer, die Achse, für die das Kommando gelten soll und anschließend das eigentliche Kommando. Dann folgt die Anzahl der zu schreibenden und der zu lesenden Bytes, und die Quell- und die Ziellänge. Anschließend werden zwei weitere Parameter für diese Achse übertragen, der „Time Out“-

Tabelle 1.10.: Schreibeblock USB-Command (wird von dem PC gesendet)

Offset in 2 Bytes	Beschreibung
0	Modus = 0x0003
1	Blocknummer
2	Adresse
3	Länge (Größe der Quelldaten)
4...	Quelldaten

Tabelle 1.11.: Schreibeblock USB-Result (wird von der USB-Umsetzer-Karte als Antwort gesendet)

Offset in 2 Bytes	Beschreibung
0	Blocknummer

und der „Retry“-Parameter. Die Adresse, an die die Quelldaten geschrieben beziehungsweise von der die Zieldaten gelesen werden, ist abhängig von einem Wert, der von der CNC in das DP-RAM geschrieben wird, dem sogenannten NextJob0, der angibt, welches Kommando als nächstes abgearbeitet werden soll. Deshalb wird hier im Gegensatzt zu dem Lese- beziehungsweise Schreibeblock keine Adressen übermittelt. Der Aufbau des Sendekommandos ist in Tabelle 1.12 abgebildet.

Die Antwort auf das Sendekommando (USB-Result) beinhaltet neben der Blocknummer, das Resultat und den Status der Achse, sowie die Daten, die ausgelesen werden sollten (siehe Tabelle 1.13).

1.6. USB-Umsetzer-Karte, die neue Hardware

Die USB-Umsetzer-Karte soll die in Kapitel 1.1.2.2 beschriebenen Nachteile der CNC beseitigen. So soll es mit dieser neuen Karte möglich sein, einen handelsüblichen Computer über USB an die CNC anzuschließen. Weiterhin soll die CNC mit dem Programm, das vorher auf dem in der CNC verbauten Computer lief, zu bedienen sein. Dazu wird der in der CNC verbaute Computer entfernt, und stattdessen die USB-Umsetzer-Karte auf den ISA-Bus gesteckt. Dabei werden folgende Anforderungen an die USB-Umsetzer-Karte gestellt:

1. Die USB-Umsetzer-Karte soll es einem angeschlossenen Computer ermöglichen, mittels der Software „CNC 48.00“ beziehungsweise „CNC 46.00“ die CNC zu bedienen.

Tabelle 1.12.: Sendekommando USB-Command (wird von dem Computer gesendet)

Offset in 2 Bytes	Beschreibung
0	Modus = 0x0001
1	Blocknummer
2	Achse
3	Kommando
4	Quelllänge (Größe der Quelldaten)
5	Ziellänge (Größe der Zieldaten)
6	Time Out
7	Retry
8...	Quelldaten

Tabelle 1.13.: Sendekommando USB-Result (wird von der USB-Umsetzer-Karte als Antwort gesendet)

Offset in 2 Bytes	Beschreibung
0	Blocknummer
1	Resultat
2	Status
3...	Zieldaten

2. Die USB-Umsetzer-Karte soll die von dem Computer versendeten USB-Kommandos verarbeiten und die enthaltenen Informationen über den ISA-Bus in das RAM der CNC schreiben.

3. Die USB-Umsetzer-Karte soll anstatt des Computers in der CNC verbaut werden.

2. Architektur der USB-Umsetzer-Karte

Die Aufgaben der USB-Umsetzter-Karte lassen sich grob in drei Kategorien unterteilen: Die USB-Kommunikation auf der Computerseite, die ISA-Kommunikation auf der Seite der CNC und die Verarbeitung der CNC-USB-Kommandos dazwischen.

Für die USB-Kommunikation wird ein USB-Controller (siehe Kapitel 2.1.1) eingesetzt, dieser ermöglicht eine relativ einfache Realisierung der USB-Kommunikation.

Es wurde bei Sieb & Meyer schon in der Vergangenheit eine CNC entworfen, die über USB mit einem Computer kommuniziert, die CNC46AS. In den Abbildungen 2.1 und 2.2 sind die Unterschiede der CNC46AS zu der CNC46/48 zu sehen (die Signale werden in den Tabellen 2.1, 2.2 und 2.3 erläutert). Die Kommunikation über USB mit dem Computer sind bei beiden CNCs, ebenso wie die Verarbeitung der CNC-Kommandos, identisch. Somit können die Zustandsautomaten, die diese Aufgaben übernehmen, für die USB-Umsetzter-Karte übernommen werden. Der Unterschied zwischen der CNC46AS und der CNC46 beziehungsweise der CNC48 besteht darin, dass diese nicht über ein Shared Memory in dem FPGA sondern über den ISA-Bus Daten austauschen. Die CNC46AS kann die Daten mit Hilfe der Steuerleitungen WE_Low, WE_High und RE Daten (Data) aus dem Shared Memory des FPGAs auslesen beziehungsweise beschreiben. Von welcher Adresse gelesen beziehungsweise auf welche Adresse geschrieben wird, wird mit Adress bestimmt. Die USB-Umsetzer-Karte der CNC46/48 hingegen schickt die Daten über den ISA-Bus an die CPU der CNC. Das bedeutet, es muss nur noch die Kommunikation zwischen FPGA und CNC-CPU mittels des ISA-Busses realisiert werden.

Da der ISA-Bus für andere Spannungen als die Hardware auf der USB-Umsetzter-Karte ausgelegt ist, werden zwischen FPGA und ISA-Bus Bustreiber benötigt. Das Blockschaltbild der Kommunikation der USB-Umsetzer-Karte für die CNC46/48 ist in Abbildung 2.2 zu sehen.

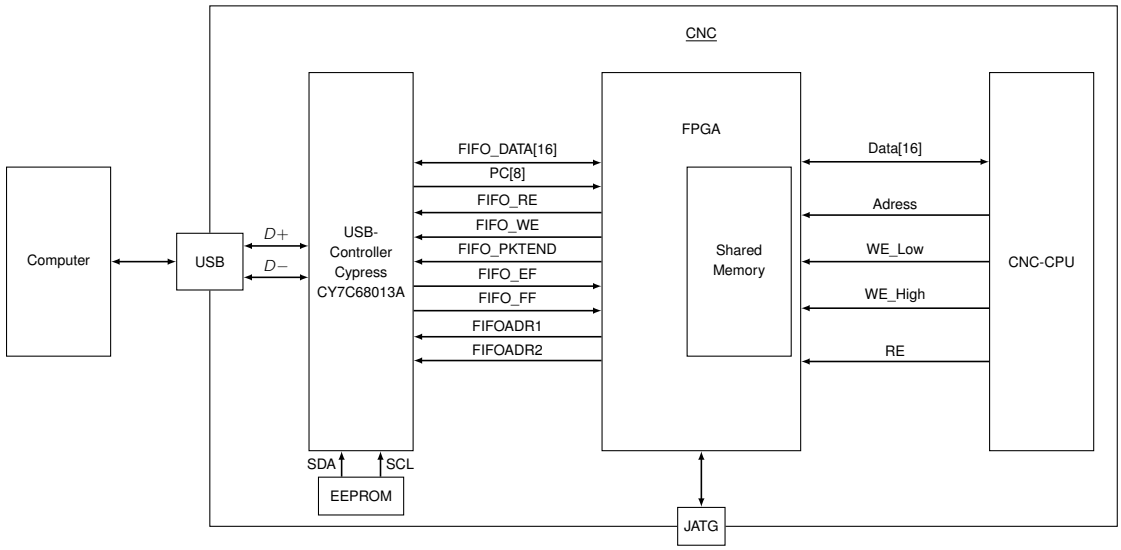


Abbildung 2.1.: Blockschaltbild der Kommunikation der CNC46AS

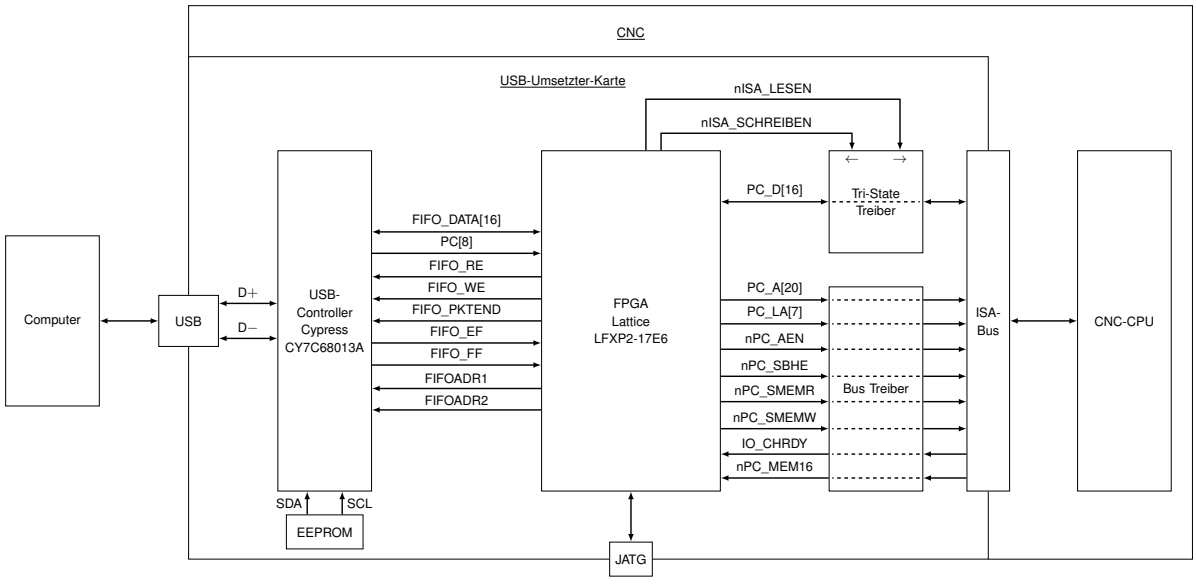


Abbildung 2.2.: Blockschaltbild der Kommunikation der USB-Umsetzter Karte für die CNC46/48

Tabelle 2.1.: Signale für die Kommunikation zwischen CNC-CPU und Shared Memory im Blockschaltbild aus Abbildung 2.1

Signal	Beschreibung
Data[16]	16 Bit breite Datenleitungen (zum Übertragen der Daten zwischen CNC-CPU und dem Shared Memory)
Adress[15]	15 Bit breite Adresse (zum Adressieren des Shared Memorys)
WE_Low	Steuerleitung, die das Schreiben des Lowbytes, der angelegten Adresse, einleitet
WE_High	Steuerleitung, die das Schreiben des Highbytes, der angelegten Adresse, einleitet
RE	Steuerleitung, die das Lesen des Highbytes und des Lowbytes, der angelegten Adresse, einleitet

Tabelle 2.2.: Signale für die Kommunikation zwischen CNC-CPU und FPGA im Blockschaltbild aus Abbildung 2.2

Signal	Beschreibung
nISA_LESEN	16 Bit breite Datenleitungen (zum Übertragen der Daten zwischen CNC-CPU und dem Shared Memory)
nISA_SCHREIBEN	15 Bit breite Adresse (zum Adressieren des Shared Memorys)
PC_D	Datenleitungen (16-Bit-Vektor).
PC_A	Adressleitungen (19-Bit-Vektor).
PC_LA	verriegelte Adressleitungen (7-Bit-Vektor) adressieren zusammen mit PC_A-Signalen das RAM der CNC.
IO_CHRDY	Über I/O Channel Ready bestätigt die CNC eine erfolgreiche Übertragung.
nPC_MEM16	Mit dem Signal Memory Chip Select 16 bestätigt die CNC einen 16-Bit-Zugriff.
nPC_SBHE	Mit System High Byte Enable kann von der USB-Umsetzer-Karte ein 16-Bit-Zugriff angefordert werden.
nPC_SMEM_R	System memory Read signalisiert der CNC einen Lesezugriff.
nPC_SMEM_W	System memory Write signalisiert der CNC einen Schreibzugriff.
PC_AEN	Address Enable signalisiert der CNC, dass die angelegte Adresse gültig ist.

Tabelle 2.3.: Signale für die Kommunikation über USB zwischen Computer und FPGA in den Blockschaltbildern aus Abbildung 2.2 und 2.1

Signal	Beschreibung
FIFO_DATA[16]	16 Bit breite Datenleitungen (zum Übertragen der Daten)
FIFO_RE	Steuersignal zum Auslesen des FIFOs des USB-Controllers
FIFO_WE	Steuersignal zum Beschreiben des FIFOs des USB-Controllers
FIFO_PKTEND	Steuersignal, dass dem USB-Controller das Ende eines Paketes signalisiert
FIFO_EF	Mit diesem Signal meldet der USB-Controller, dass sein Eingangs-FIFO leer ist (FIFO Empty flag)
FIFO_FF	Mit diesem Signal meldet der USB-Controller, dass sein Ausgangs-FIFO voll ist (FIFO Full flag)
PC[8]	Ein Ausgangsport des USB-Controllers, zum Übertragen von Reset-Signalen

2.1. Bauteile der USB-Umsetzer-Karte

2.1.1. USB-Controller

Die USB-Kommunikation wird mit dem USB-Controller CY7C68013A [Cypress Semiconductor Corporation (2005)] des Herstellers Cypress realisiert. Dieser Controller übernimmt die komplette USB Kommunikation. Er stellt lediglich die reinen Nutzdaten der restlichen Hardware zur Verfügung (siehe weiter unten). Das Anmelden an den Computer wird von dem USB-Controller autonom durchgeführt. Auch die USB-Header samt Checksummen werden von dem USB-Controller erstellt. Die Hersteller- und die Produkt-ID werden über den I^2C-Bus[1] (mit den Leitungen SCL und SDA) aus einem externen Speicher (EEPROM) ausgelesen. Es ist auch möglich die Firmware des Controllers über ein so angeschlossenes EEPROM auszulesen. Hier wird die Firmware allerdings per USB übertragen, was das nachträgliche Ändern der Firmware erleichtert.

Für die USB-Umsetzter-Karte wird dieser Kontroller im Bulk-Transfer betrieben, da die Datenübertragung nicht zeitkritisch ist.

Der CY7C68013A ist ein USB2.0 fähiger Mikrocontroller mit einem 8051 Kern. Der Controller wurde so konfiguriert, dass zwei First In – First Out (FIFO) Speicher zur Verfügung stehen, auf die über einen 16 Bit breiten bidirektionalen Datenbus zugegriffen werden kann.

- Die FIFOs sind über zwei Adressleitungen (FIFOADR1, FIFOADR2) adressierbar.
  - Der FIFO an der Adresse „00“ dient als Speicher für die eingehenden (vom PC kommenden) Daten und
  - der FIFO mit der Adresse „01“ als Speicher für die ausgehenden Daten.
- SLOE (Slave Output Enable) schaltet die Datenleitungen des USB-Controllers für ausgehende Daten frei.
- Über die Steuerleitungen SLRD (Slave Read) wird der FIFO mit der Adresse „00“ ausgelesen.
- Mit der Steuerleitung SLWR (Slave Write) werden die Daten auf dem Datenbus in das FIFO mit der Adresse „01“ übernommen.
- Der USB-Conroller sammelt im FIFO mit der Adresse „01“ Daten, um sie dann in USB-Paketen an den PC zu schicken.
- Um dem USB-Controller zu signalisieren, dass für das USB-Paket keine weiteren Daten folgen, dient das PKTEND (Packet End) Signal.

[1] I^2C ist ein serieller Datenbus

- Mit den Flags A und B des USB-Controllers wird dem FPGA der Zustand des adressierten FIFO signalisiert.
  - FLAGA ist aktiv, wenn der FIFO für ausgehende Daten voll ist (FIFO_FULL_FLAG) und
  - FLAGB ist aktiv, wenn der FIFO für eingehende Daten leer ist (FIFO_EMPTY_FLAG).

Der Controller wurde direkt mit der USB-Buchse der USB-Umsetzter-Karte verbunden. Die Steuerleitungen (SLRD, SLWR, SLOE, FLAGA, FLAGB), die Adressleitungen (ADR0, ADR1) und die Datenleitungen der FIFOs gehen an das FPGA. Die Spannungsversorgung des Controllers beträgt 3,3 Volt.

2.1.2. Lattice LFXP2 -17E, FPGA

Die FPGAs der Lattice XP2 [Lattice Semiconductor Corp. (2009)] Familie besitzen integrierte Flash-Speicher, auf denen die Konfiguration des FPGAs gespeichert wird. Beim Starten des FPGAs wird die Konfiguration aus dem Flash-Speicher in die SRAM Konfigurationsspeicherzellen überspielt. Das bietet den großen Vorteil, dass keine externe Logik und Speicher für das Einspielen der Konfiguration vonnöten ist.

In diesem Lattice Baustein wird der Zusammenschluss von vier Slices (Slice 0 bis Slice 3) Programmable Functional Unit (PFU) genannt. Wobei die Slices 0 bis 2 jeweils zwei Look-Up-Tables mit jeweils vier Eingängen und zwei D-Flipflops enthalten; Slice 3 enthält nur zwei Look-Up-Tables. Die Programmable Fuctional Units können logische und arithmetische Funktionen sowie RAMs und ROMs bilden. Die Informationen dieser RAMs und ROMs kann in das integrierte Flash-RAM geschrieben werden und geht somit bei Ausschalten der Versorgungsspannung des FPGAs nicht verloren. RAM-Bausteine stehen auch über das Embedded Block RAM (EBR) zur Verfügung; dieses kann auch als DDR-Speicher[2] benutzt werden.

Die Spannung, mit der die Interne Logik des FPGAs arbeitet, beträgt 1,2 Volt. Zusätzlich benötigt das FPGA noch 3,3 Volt für V_{CCAUX}, V_{CCPLL}, V_{CCJ} und V_{CCIO0} bis V_{CCIO7}. Dabei dient V_{CCAUX} als interne Referenz für die Eingangsbuffer, V_{CCPLL} als Versorgungsspannung für interne PLLs (Phase Locked Loops) und V_{CCJ} als Spannungsversorgung für die JTAG Schnittstelle. Die Spannungen V_{CCIO0} bis V_{CCIO7} sind Versorgungsspannungen für die I/O-Banks (Ein- und Ausgabebänke).

Auf der USB-Umsetzter-Karte wird der LFXP2-17E hauptsächlich eingesetzt, da er keine externen Speicherbausteine für die Konfiguration benötigt. In Kombination mit dem Sicherheitsbit wird es sehr schwer, die Funktionsweise des FPGAs auszulesen. Die Erfahrungen,

[2] DDR-Ram ermöglicht eine Datenübertragung bei steigender als auch bei fallender Taktflanke

die in diesem Unternehmen in der Benutzung der Lattice XP2 Familie gesammelt wurden, sind ein weiterer Grund, der für dieses FPGA spricht.

2.1.3. Bustreiber

Damit die logischen Zustände sicher auf dem Bus liegen, werden Bustreiber eingesetzt. Diese sind in der Lage, eine höhere Spannung und einen höheren Strom als das FPGA zu treiben. Des Weiteren bieten sie die Möglichkeit, die Busausgänge hochohmig zu schalten (Tri-State), so ist es möglich, von Leitungen zu lesen (Ausgang hochohmig) sowie auch darauf zu schreiben.

Auf der USB-Umsetzter-Karte werden sechs 74HCT541 und zwei 74LVC541 Tri-State Bustreiber verwendet. Die Bauteile 74HCT541 (siehe [Philips Semiconductors (1990)] und [Philips Semiconductors (1997)]) (High-speed CMOS) sind kompatibel zu TTL-Pegeln und können mit einer Betriebsspannung von $4,5$ bis $5,5$ Volt betrieben werden. An die 74HCT541 Treiber werden alle Ausgänge des FPGAs angeschlossen (alle Daten- und Adressleitungen, sowie die Steuerleitungen). Die Bustreiber der Steuersignale und Adressleitungen sind immer aktiv. Das ist bei den Datenleitungen nicht der Fall. Da diese bidirektional sind, werden sie über das Signal $\overline{\text{ISA-SCHREIBEN}}$ aktiv geschaltet ($\overline{\text{ISA-SCHREIBEN}}$ = 0: Bustreiber aktiv; $\overline{\text{ISA-SCHREIBEN}}$ = 1: Tri-State).

Zusätzlich gehen die Datenleitungen an die 74LVC541 Treiber, um auch das Lesen zu ermöglichen. Die 74LVC541 können mit 3,3 Volt betrieben werden und können somit direkt von der USB-Umsetzter-Karte versorgt werden. Die Treiber besitzen ebenfalls einen Enable-Eingang, mit dem die Ausgänge hochohmig geschaltet werden können ($\overline{\text{ISA-LESEN}}$). Ist $\overline{\text{ISA-LESEN}}$ = 0 befinden sich die Treiber in aktivem Zustand, ist $\overline{\text{ISA-LESEN}}$ = 1 sind die Ausgänge des Treibers hochohmig. Es darf also immer nur einer der beiden Treiber für die Datenleitungen zu einem Zeitpunkt aktiv sein.

2.1.4. Sonstige Bauteile der USB-Umsetzer-Karte

Die 5 Volt Versorgungsspannung der USB-Umsetzer-Karte kommt über den ISA-Bus von der CNC. Aus diesen 5 Volt wird mit Hilfe eines FAN1086M33 eine 3,3 Volt Spannung erzeugt. Der FAN1086M33 ist ein Spannungsregler, der auch bei niedrigen Spannungsdifferenzen zwischen Ein- und Ausgangsspannung funktionsfähig ist (er benötigt eine Eingangsspannung zwischen $4,8$ und 7 Volt).

Die 1,2 Volt für die Kernspannung des FPGAs wird mit einem TPS79301 erzeugt. Das ist ein einstellbarer Spannungswandler, der ebenfalls in der Lage ist, mit einer niedrigen Spannungsdifferenz zwischen Ein- und Ausgang zu funktionieren.

Ein Reset-Baustein (LM809-2.93) steuert die Reset-Eingänge des USB-Controllers und des FPGAs. Steigt die Versorgungsspannung beim Einschalten über 2,93 Volt deaktiviert der Reset-Baustein nach 240 Millisekunden den Reset. Der Systemtakt wird von einem Quarzoszillator generiert, der mit 24MHz schwingt. Der Chip, aus dem der USB-Controller die Produkt- und Hersteller-ID ausliest, ist ein X24C04-3. Der X24C04-3 ist ein serielles 4096 Bit EEPROM (electrically erasable programmable read-only memory).

2.2. Funktionen des FPGAs

Das FPGA übernimmt die zentrale Rolle der USB-Umsetzter-Karte. Es soll das FIFO des USB-Controllers auslesen und beschreiben, die eingehenden CNC-USB-Kommandos verarbeiten und auch die Kommunikation mit der CNC-Karte über den ISA-Bus durchführen. Diese Aufgaben können ebenfalls in drei Bereiche aufgeteilt werden, so dass daraus drei Zustandsautomaten entstehen. Wobei der Zustandsautomat für die Verarbeitung der USB-Kommandos die Steuerung der anderen Zustandsautomaten übernimmt. Die Zustandsautomaten für die USB-Kommunikation und der Verarbeitung der CNC-USB-Kommandos wurden schon in anderen Projekten des Unternehmens verwendet und mussten somit nur an wenigen Stellen angepasst werden. Abbildung 2.3 zeigt das Zusammenspiel der Zustandsautomaten der bereits vorhandenen CNC46AS. Die Zustandsautomaten für die USB-Kommunikation (USB_COMMUNICATION) und für die Kommandobearbeitung (COMMAND_HANDLER) können mit aus der CNC46AS übernommen werden (in Abbildung rot makiert).

Das FPGA benötigt auf der einen Seite Pins, um mit dem USB-Controller zu kommunizieren und auf der anderen Seite Pins für den ISA-Bus. Anstelle des Zustandsautomaten für die DPRAM-Kommunikation (DPRAM_COMMUNICATION) tritt nun ein Zustandsautomat, der die Kommunikation über den ISA-Bus steuert (ISA_COMMUNICATION). Das Zusammenspiel der Zustandsautomaten der USB-Umsetzter-Karte ist in Abbildung 2.4 zu sehen.

2.2.1. Kommandoverarbeitung

Die ankommenden CNC-USB-Kommandos werden in einem Zustandsautomat (COMMAND_HANDLER) verarbeitet. Dieser Zustandsautomat stellt die zentrale Komponente der USB-Umsetzter-Karte dar. Die Verbindung zu dem Computer (über USB) und zu der CNC (über den ISA-Bus) übernehmen zwei weitere Zustandsautomaten, der USB-Zustandsautomat und der ISA-Zustandsautomat. Anhand der Informationen, die über USB gesendet werden (die CNC-Kommandos), erkennt der Zustandsautomat welche Daten an welche Adresse im RAM der CNC geschrieben, beziehungsweise von welcher Adresse die

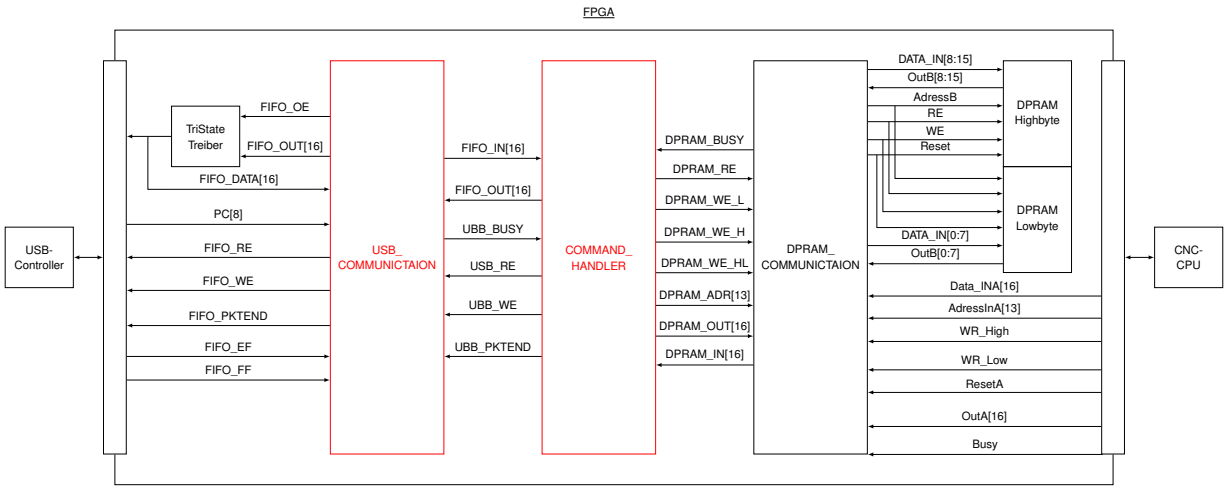


Abbildung 2.3.: Blockschaltbild der Zustandsmaschinen der CNC46AS

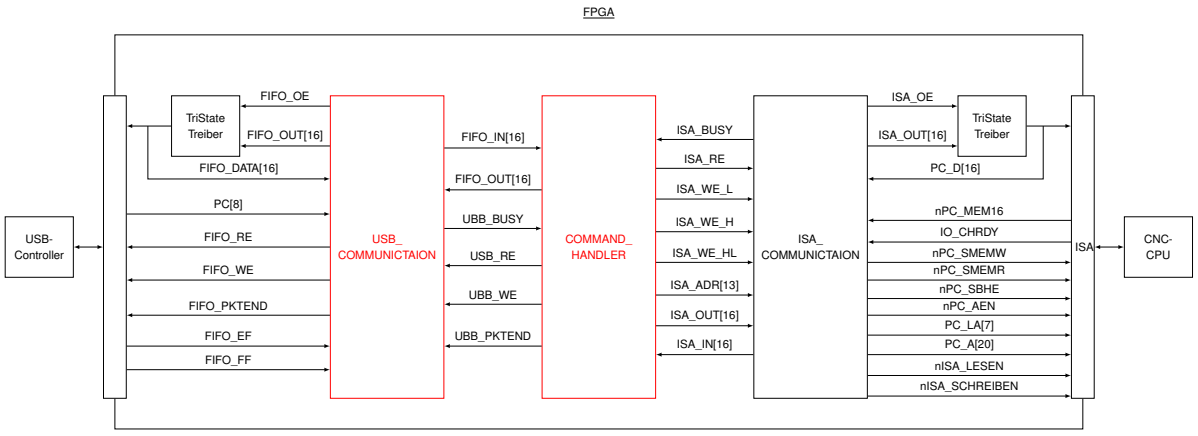


Abbildung 2.4.: Blockschaltbild der Zustandsmaschinen der USB-Umsetzer-Karte

Daten ausgelesen werden sollen. Über Steuersignale wird den beiden anderen Zustandsautomaten mitgeteilt, wann gelesen beziehungsweise geschrieben werden soll. Die Adressen für die Kommunikation über den ISA-Bus sind in diesem Zustandsautomaten mit einer Länge von 13 Bit gespeichert und adressieren jeweils 16 Bit breite Daten. Welches der Bytes angesprochen werden soll, wird mit Steuerleitungen entschieden.

2.2.2. USB_COMMUNICATION-Zustandsautomat

Der Zustandsautomat steuert den USB-Controller über die Adress- und Steuerleitungen. Mit den drei Signalen USB_RE, USB_WE und USB_PKTEND kann dem Zustandsautomaten signalisiert werden, dass ein 16-Bit-Block aus dem FIFO des USB-Controllers ausgelesen (USB_RE), ein 16-Bit-Block in das FIFO geschrieben (USB_WE) oder dem USB-Controller das Ende eines Paketes (USB_PKTEND) signalisiert werden soll. Ob der USB-Zustandsautomat gerade beschäftigt ist, wird mit dem Signal USB_BUSY angezeigt.

2.2.3. ISA_COMMUNICATION-Zustandsautomat

Der ISA-Zustandsautomat steuert die Kommunikation mit der CNC über den ISA-Bus. Hier werden die dazu benötigten Steuerleitungen und die Daten- und Adressleitungen geschaltet. Der Aufbau des RAMs der CNC macht es notwendig, drei verschiedene Varianten des Schreibzugriffs zu realisieren (siehe Kapitel 1.1.2.1); Schreiben des Lowbytes (ISA_WE_L), Schreiben des Highbytes (ISA_WE_H) und das Schreiben des High- und des Lowbytes (ISA_WE_LH). Das Lesen erfolgt immer in dem 16-Bit Modus (ISA_RE). Bei dem Schreiben des Highbytes wird ein 8-Bit-Schreibzugriff durchgeführt. Bei dem Schreiben des Lowbytes und auch bei dem Schreiben von 16 Bit (Low- und Highbyte) wird hingegen ein 16-Bit-Schreibzugriff durchgeführt.

Da der Zustandsautomat der Kommandoverarbeitung mit einer Adresse von 13 Bit Breite arbeitet, muss diese in dem ISA-Zustandsautomaten ergänzt werden. Die obersten sechs Adressbits sowie die Latched-Adress-Bits, werden mit der Adresse der CNC-Steuerung belegt. Die Adressbits 14 bis 19 enthalten die dem ISA-Zustandsautomaten übergebene Adresse. Bei einem Lesezugriff, beim Schreiben des Highbytes oder einem 16-Bit-Schreibzugriff wird das niederwertigste Bit mit einer Null ergänzt. Bei dem Schreiben des Lowbytes wird das niederwertigste Bit allerdings eins gesetzt. Ist ein Lese- oder Schreibzugriff aktiv, wird dies durch das Signal ISA_BUSY angezeigt. Die angesprochene Adresse muss dem ISA-Zustandsautomaten genauso wie zu schreibende Daten übergeben werden.

3. Realisierung

3.1. Realisierung der Zustandsautomaten

Die Zustandsautomaten wurden mit Hilfe der Entwicklungsumgebung „Lattice Diamond 1.0“[1] in VHDL modelliert. Synthetisiert wurde der VHDL-Quelltext mit Synplify von Synplicity, was Bestandteil der Entwicklungsumgebung ist. Die Zustandsautomaten wurden alle in jeweils einem Prozess abgebildet. Da diese Prozesse mit dem Takt CLK getaktet sind, sind alle Ausgänge ebenfalls getaktet. Deshalb handelt es sich bei den Zustandsautomaten um Moore-Automaten. Die Prozesse der Automaten sind, wie in Quelltext 3.1 zu sehen, aufgebaut.

Quelltext 3.1: Modell eines Zustandsautomaten

```
1   FSM: process(CLK, nRESET)
    begin
3       if (nRESET = '0') then
            -- alle Ausgaenge zuruecksetzen
5           -- in Ruhezustand zurueckkehren
        elsif (CLK = '1' and CLK'event)
7           case ZUSTAND is
                when Z1 =>
9                   -- zustand Z1
                    ZUSTAND <= Z2; -- Folgezustand setzen
11                  AUSGANG <= '1'; -- Ausgaenge setzen
                when Z2 =>
13                  -- Zustand Z2
                    -- ...
15              -- ...
            end case;
17      end if;
    end process;
```

[1] http://www.latticesemi.com/products/designsoftware/diamond/

3.1.1. Kommando-Zustandsautomat (COMMAND_HANDLER)

Nach einem Reset geht der Zustandsautomat in den IDLE-Zustand. Dem IDLE-Zustand folgt immer der USB_MODE_0-Zustand, in dem eine Leseanfrage an den USB-Zustandsautomaten gesendet wird. Jetzt wartet der Zustandsautomat so lange, bis der USB-Zustandsautomat nicht mehr beschäftigt ist. Anschließend werden die über USB gelesenen Daten ausgewertet. Diese Daten geben den Typ des Kommandos an. Je nachdem, wie der Inhalt dieser Daten ist, wird der Folgezustand gewählt. Bei einer 1 für ein Sendekommando, bei einer 2 für einen Leseblock, und bei einer 3 für einen Schreibeblock. Enthalten die Daten einen anderen Wert, so kehrt der Automat, ohne Fehlermeldung, in den IDLE-Zustand zurück. Ist ein Fehler aufgetreten, so bemerkt das der Computer, da keine Antwort auf das CNC-Kommando gesendet wurde und resettet den Zustandsautomaten.

3.1.1.1. Leseblock (IN)

Zunächst muss hier die Blocknummer aus dem FIFO des USB-Controllers gelesen werden. Hierzu wird eine Leseanforderung mittels USB_RE an den USB-Zustandsautomaten gesendet. Im nächsten Zustand wird solange gewartet, bis die USB-Kommunikation abgeschlossen ist und anschließend die Daten kopiert. Dieses Auslesen des FIFOs wird häufig angewandt und im folgenden Text nicht weiter erläutert.

Anschließend wird die Adresse aus dem USB-Controller ausgelesen, von der die Daten gelesen werden sollen, danach wie viele Bytes zu lesen sind.

Nun wird die Antwort auf das USB-CNC-Kommando vorbereitet. Als erstes wird die Blocknummer (siehe Kapitel 1.5) an den USB-Controller übertragen.

Sollen noch weitere Daten gelesen werden, so wird die Adresse an den ISA-Zustandsautomaten übergeben und die Daten ausgelesen.

Wurden alle Daten ausgelesen, wird das USB_PKTEND-Signal gesetzt und auf den USB-Controller gewartet. Der Zustandsautomat kehrt nun in den IDLE-Zustand zurück.

3.1.1.2. Schreibeblock (OUT)

Dieser Teil des Zustandsautomaten ähnelt sehr stark dem des Leseblocks. Der Unterschied zum Verarbeiten des Leseblocks liegt darin, dass die Daten aus dem USB-Controller ausgelesen werden und dann über den ISA-Bus an die CNC geschickt werden.

Beim Senden der Daten über den ISA-Bus ist allerdings zu berücksichtigen, ob das High-, das Lowbyte oder Beide geschrieben werden sollen. Dazu wird beim Auslesen der Zieladresse geprüft, ob es sich um eine ungerade Adresse handelt. Wenn dies der Fall ist, so wird bei dem ersten Senden über den ISA-Bus nur das Lowbyte geschrieben.

Ist am Ende eines Schreibblockes nur noch ein Byte zu schreiben, dann wird nur das Highbyte geschrieben.

3.1.1.3. Sendekommando (CMD)

Wird ein Sendekommando übertragen, folgen dem Modus einige Daten (siehe Kapitel 1.5.3). Diese Daten müssen erst einmal eingelesen werden bevor es möglich ist, das Kommando über den ISA-Bus zu senden. Die Blocknummer, Achse, Kommando, Quell- und Ziellänge, Time Out und Retry Daten werden in internen Signalen zwischengespeichert, um sie später an die CNC zu senden. Das Auslesen erfolgt jedes Mal nach dem gleichen Schema. Die Leseanfrage (USB_RE) wird gesetzt. Hiernach wird solange gewartet, bis der USB-Zustandsautomat fertig ist und die Daten in das entsprechende Signal kopiert.

Dann wird der Inhalt der JOB0-Speicherstelle der CNC ausgelesen und in einem Signal gespeichert. Der Inhalt des JOB0-Speichers gibt Information darüber, wo die Daten des Sendekommandos hingeschrieben werden sollen. Anschließend werden die Daten des Sendekommandos über den ISA-Bus an die entsprechenden Stellen im Speicher geschrieben.

Jetzt werden die Quelldaten, falls diese in die CNC geschrieben werden sollen, mit Hilfe des ISA-Zustandsautomaten, über den ISA-Bus übertragen. Danach, oder wenn keine Daten zu schreiben sind, wird der CREG-Speicher der CNC ausgelesen und solange gewartet, bis der Inhalt einen bestimmten Wert erreicht hat.

Ist dies geschehen, werden die Daten Timeout, SJob und CREG0 über den ISA-Bus gesendet. Der Zustandsautomat wartet anschließend bis ein spezieller Wert über den ISA-Bus aus der CNC an der Adresse des Status ausgelesen wird. Anschließend wird der Status der CNC ausgelesen und der Header des USB-Results geschrieben. Dann werden die Quelldaten, wie beim Leseblock, aus der CNC ausgelesen und an den USB-Controller gesendet. Zum Schluss wird der Inhalt des REG0-Registers der CNC abgefragt und solange gewartet, bis dieser den Wert Null enthält. Dann wird der Status der CNC ebenfalls Null gesetzt und die Nummer des Buffers in das REG0-Register kopiert. Anschließend kehrt der Zustandsautomat in den IDLE-Zustand zurück.

3.1.2. USB-Zustandsautomat (USB_FSM)

Nach einem Reset geht dieser Zustandsautomat in den IDLE-Zustand. In diesem Zustand werden die Eingangssignale USB_WE, USB_RE und USB_PKTEND abgefragt. Des Weiteren wird hier das USB_BUSY-Signal Null gesetzt und alle Steuerleitungen zu dem USB-Controller gelöscht.

Wurde das Auslesen des Fifios des USB-Cotroller durch Setzen des USB_RE-Signals angefordert, so wird das USB_BUSY -Signal gesetzt und solange gewartet bis der Fifo-Speicher mit dem Flag FIFIO_EF (Fifo Empty Flag) signalisiert, dass der Fifio nicht leer ist. Sind Daten in dem Fifio-Speicher vorhanden (FIFO_EF = 1) so wird das nSLRD-Steuersignal des USB-Controllers gesetzt und somit das Auslesen des Fifos angefordert. Nach einem Wartezustand werden die Daten von den Fifo-Ausgängen in das USB_IN-Signal kopiert. In dem nächsten Zustand wird nSLRD wieder gelöscht. Es folgt ein weiterer Wartezustand bevor der Zustandsautomat wieder in den IDLE-Zustand zurückkehrt.

Wurde das Beschreiben des Fifos mit dem Signal USB_WE angefordert, so wird das USB-BUSY-Signal gesetzt und gewartet, bis der USB-Controller mit dem FIFO_FF-Signal (Fifio Full Flag) signalisiert, dass der Fifo nicht voll ist. Dann wird die Steuerleitung nSLWR des USB-Controllers gesetzt. Außerdem wird mit dem Signal FOFO_OE der Tri-State-Treiber des FPGAS zu dem USB-Controller freigegeben. Anschließend folgen zwei Wartezustände. Jetzt wird die Steuerleitung nSLWR gelöscht und der Tri-State-Treiber hochohmig geschaltet. Nach einem weiteren Wartezustand folgt nun wieder der IDLE-Zustand.

Soll dem USB-Controller das Ende eines Packetes signalisiert werden, was dem Zustandsautomaten mit dem Signal USB_PKTEND signalisiert wird, so wird das USB_BUSY-Signal, sowie die nPKTEND-Steuerleitung des USB-Controllers gesetzt. Es folgen drei Wartezustände. Danach kehrt der Zustandsautomat wieder in den IDLE-Zustand zurück.

3.1.3. ISA-Zustandsautomat (ISA_COMMUNICATION)

In Tabelle 3.1 sind die für den ISA-Bus notwendigen Zeiten aufgelistet. Die Zeiten sind der SpezifikationAmpro Computers (1998) zu entnehmen

Das ASM-Diagramm[2] ist in Anhang A zu sehen. In Tabelle 3.2 sind die Zustände kurz erläutert.

[2] Algorithmic-State-Machine-Diragramm ist eine Methode um Zustandsautomaten zu beschreiben

Tabelle 3.1.: Erforderliche Zeiten für den ISA-Bus

Nr.	Beschreibung	Zeit in ns	
		8-Bit-Zugriff	16-Bit-Zugriff
1	Command deasserted (mindeste Zeit zwischen zwei Speicherzugriffen)	min. 170	min. 108
2	PC_A, nPC_SBHE setup to nPC_SMEMx (Zeit, die mindestens PC_A und nPC_SBHE vor nPC_SMEMR bzw. nPC_SMEMW anliegen muss)	min. 102	min. 39
3	PC_A, nPC_SBHE hold (solange muss PC_A und nPC_SBHE nach dem nPC_SMEMR bzw. nPC_SMEMW noch anliegen)	min. 53	min. 53
4	PC_LA setup to nPC_MEMx asserted (solange muss PC_A vor nPC_SMEMR bzw. nPC_SMEMW anliegen)	min. 83	min. 120
5	PC_MEM16 valid from PC_LA (Zeit zwischen dem Anlegen von PC_LA und der Antwort der Peripherie durch _PCMEM16)		min. 102
6	PC_AEN asserted to nPC_MEMx active (Zeit, die PC_AEN vor nPC_SMEMR bzw. nPC_SMEMW anliegen muss)	min. 11	min. 11
7	PC_AEN hold to nPC_MEMx inactive (Zeit, die PC_AEN nach nPC_SMEMR bzw. nPC_SMEMW anliegen muss)	min. 11	min. 11
8	IO_CHRDY valid from command asserted (Zeit, die zwischen dem Anlegen von nPC_SMEMR bzw. nPC_SMEMW und IO_CHRDY liegen kann)	min. 60	min. 70

Tabelle 3.2.: Erläuterung der Zustände des ISA_COMMUNICATION-Automaten

Zustand	Erläuterung
IDLE	Hier verweilt der Zustandsautomat solange, bis er zum Schreiben auf oder Lesen von dem ISA-Bus aufgefordert wird.
RE_SET_ADR	Die dem ISA_COMMUNICATION-Automaten übergebene Adresse (ISA_ADR) wird mit einer Null ergänzt und auf den ISA-Bus gelegt (PC_A und PC_LA). Desweiteren werden die Signale nPC_SBHE, nPC_AEN und ISA_LESEN auf Null gesetzt.
RE_WAIT_0 bis RE_WAIT_3	Warten damit die CNC die Adresse übernehmen kann.
RE_CHECK_M16	Es wird geprüft, ob die CNC einen 16-Bit-Lesezugriff zulässt (nPC_MEM16 muss Null sein). Ist dies nicht der Fall folgt der IDLE-Zustand.
RE_WAIT_4 bis RE_WAIT_6	Warten bis eine Leseanforderung gesendet werden darf.
RE_SET_SMEMR	Leseanforderung senden (nPC_SMEMR Null setzen).
RE_WAIT_7 bis RE_WAIT_11	Auf Antwort der CNC warten.
RE_WAIT_CHRDY	Es wird solange in diesem Zustand gewartet, bis die CNC das Signal IO_CHRDY Eins setzt.
RE_READ	Die auf dem ISA-Bus anliegenden Daten einlesen (ISA_IN = PC_D).
RE_CLR_SMEMR	Leseanforderung löschen (nPC_SMEMR Eins setzen).
RE_WAIT_12 bis RE_WAIT_14	Warten bis nächster Zugriff über den ISA-Bus stattfinden darf. Anschließend folgt der IDLE-Zustand.
WR_H_SET_ADR	Die dem ISA_COMMUNICATION-Automaten übergebene Adresse (ISA_ADR) wird mit einer Eins ergänzt und auf den ISA-Bus gelegt (PC_A und PC_LA). Desweiteren werden die Signale nPC_AEN und ISA_SCHREIBEN auf Null und ISA_OE Eins gesetzt.
WR_H_WAIT_0 bis WR_H_WAIT_8	Warten damit die CNC die Adresse übernehmen kann.
WR_H_SET_SMEMW	Schreibanforderung senden (nPC_SMEMW Null setzen).
WR_H_WAIT_9 bis WR_H_WAIT_11	Auf Antwort der CNC warten.
WR_H_WAIT_CHRDY	Es wird solange in diesem Zustand gewartet, bis die CNC das Signal IO_CHRDY Eins setzt.

Tabelle 3.2.: Erläuterung der Zustände des ISA_COMMUNICATION-Automaten Fortsetzung

Zustand	Erläuterung
WR_H_WAIT_12 bis WR_H_WAIT_14	Schreibanforderung löschen (nPC_SMEMW Eins setzen) und warten, bis die Adresse geändert werden darf.
WR_H_WAIT_15	Adresse Null und Signale in den IDLE-Zustand setzen (nISA_SCHREIBEN und nPC_AEN Eins setzen).
WR_H_WAIT_16 und WR_H_WAIT_17	Warten bis nächster Zugriff über den ISA-Bus stattfinden darf. Anschließend folgt der WAIT_0-Zustand.
WR_L_HL_SET_ADR	Die dem ISA_COMMUNICATION-Automaten übergebene Adresse (ISA_ADR) wird bei einer Schreibanforderung des Lowbytes mit einer Eins, bei einer 16-Bit-Schreibanforderung mit einer Null ergänzt. Anschließend wird die Adresse auf den ISA-Bus gelegt (PC_A und PC_LA). Des Weiteren werden die Signale nPC_SBHE, nPC_AEN und ISA_SCHREIBEN auf Null und ISA_OE Eins gesetzt .
WR_L_HL_WAIT_0 bis WR_L_HL_WAIT_3	Warten damit die CNC die Adresse übernehmen kann.
WR_L_HL_CHECK_M16	Es wird geprüft, ob die CNC einen 16-Bit-Lesezugriff zulässt (nPC_MEM16 muss Null sein). Ist dies nicht der Fall, folgt der IDLE-Zustand.
WR_L_HL_WAIT_4 bis WR_L_HL_WAIT_5	Warten bis eine Leseanforderung gesendet werden darf.
WR_L_HL_SET_SMEMW	Schreibanforderung senden (nPC_SMEMW Null setzen).
WR_L_HL_WAIT_6 bis WR_L_HL_WAIT_9	Auf Antwort der CNC warten.
WR_L_HL_WAIT_CHRDY	Es wird solange in diesem Zustand gewartet, bis die CNC das Signal IO_CHRDY Eins setzt.
WR_L_HL_WAIT_10 bis WR_L_HL_WAIT_12	Schreibanforderung löschen (nPC_SMEMW Eins setzen) und warten, bis die Adresse geändert werden darf.
WR_L_HL_WAIT_13	Adresse Null und Signale in den IDLE-Zustand setzen (nISA_SCHREIBEN und nPC_AEN Eins setzen). Anschließend folgt der WAIT_0-Zustand.
WAIT_0 und WAIT_1	Warten bis nächster Zugriff über ISA-Bus stattfinden darf. Anschließend folgt der IDLE-Zustand.

Nach einem Reset geht der Zustandsautomat in den IDLE-Zustand. Hier verweilt der Auto-

mat bis über eine der Steuerleitungen ein Lese- oder Schreibzugriff angefordert wird. Wurde eine Steuerleitung gesetzt, wird die Adresse auf den ISA-Bus gelegt. Bei einem Lesezugriff und bei dem Schreiben des Low- und/oder Highbytes wird mit der Steuerleitung nPC_SBHE ein 16-Bit-Zugriff angefordert. Des Weiteren wird mit dem Signal nPC_AEN die angelegte Adresse als gültig gekennzeichnet.

Bei einem Lesezugriff wird das nISA_LESEN-Signal gesetzt um, den Tri-State-Treibern zu signalisieren, dass Daten von dem ISA-Bus in das FPGA gehen. Bei den Schreibzugriffen wird hingegen das nISA_SCHREIBEN-Signal und das ISA_OE-Signal gesetzt. Wurde der Lesezugriff mit ISA_RE eingeleitet ist der Folgezustand RE_SET_ADR. Bei einer Schreibanforderung des Highbytes ist der Folgezustand WR_H_SET_ADR. Soll das Lowbyte oder das Low- und Highbyte geschrieben werden, ist der Folgezustand WR_L_HL_SET_ADR.

3.1.3.1. Lesezugriff

Dem RE_SET_ADR Zustand folgen vier Wartezustände (RE_WAIT_0 bis RE_WAIT_3). Diese Wartezustände sind notwendig, damit die CNC-Steuerung die angelegte Adresse übernehmen kann. In der ISA-Spezifikation ist festgelegt, dass die Peripherie nach maximal 102 Nanosekunden nach dem Anlegen der Latched-Adress mit dem Signal MEMCS16 die Fähigkeit einen 16-Bit-Zugriff zu ermöglichen bestätigt. Das entspricht bei einem Takt von $48MHz$ fünf ($\frac{1}{48}MHZ = 20,8333ns\frac{102ns}{20,8333n}s = 4,896 \rightarrow 5$) Wartezyklen. Da der Zustand RE_SET_ADR auch zwischen dem Setzen der Adresse und dem Abfragen des MEMCS16-Signals liegt, sind nur vier weitere Wartezustände vonnöten. Anschließend wird im Zustand RE_CHECK_M16 das Signal nPC_MEM16 abgefragt. Ist diese Null, so ist die Peripherie in der Lage einen 16-Bit-Zugriff durchzuführen und es folgt der Zustand RE_WAIT_4. Ist das Signal allerdings Eins, so kann kein 16-Bit-Zugriff erfolgen und der Zustandsautomat kehrt in den IDLE-Zustand zurück; dies dürfte mit der CNC-Steuerung allerdings nicht geschehen, da diese das Signal immer auf Null setzt, wenn diese adressiert wird. Diese Abfrage wurde, obwohl für die Kommunikation mit der CNC-Steuerung unnötig, für spätere, eventuelle Änderungen eingebaut.

Nach weiteren drei Wartezuständen (RE_WAIT_4 bis RE_WAIT_6) wird in dem Zustand RE_SET_SMEMR das Signal nPC_SMEMR gesetzt. Durch die Wartetakte wird gewährleistet, dass zwischen dem Setzen Adressen und dem Setzen des nPC_SMEMR mindestens 102 Nanosekunden verstrichen sind. Jetzt folgen weitere fünf Wartezustände. Diese Wartetakte sind notwendig, um der CNC Zeit zu geben, auf die Leseanforderung zu reagieren und die Daten auf den Datenbus zu legen. Liegen die Daten auf dem Bus, so signalisiert die CNC dies mit dem IO_CHRDY-Signal. Laut den ISA-Spezifikationen muss zwischen dem Setzen des nPC_SMEMR-Signals und dem Abfragen von IO_CHRDY mindestens eine Zeit von 70 Nanosekunden liegen.

In dem nun folgenden Zustand (RE_WAIT_CHRDY) bleibt der Zustandsautomat solange, bis das IO_CHRDY Singnal gesetzt wurde. Anschließend werden in dem Zustand RE_READ die Daten von dem Bus gelesen. Nachdem das nPC_SMEMR Signal wieder gelöscht wurde, folgen drei weitere Wartetakte die garantieren, dass zwischen dem nächsten Aufruf eine Pause von mindestens 108 Nanosekunden liegt. Dann kehrt der Zustandsautomat wieder in den IDLE-Zustand zurück.

3.1.3.2. Schreibzugriff auf das Highbyte (8-Bit-Schreibzugriff)

Dem Setzen der Adresse folgen neun Wartezustände. Anschließend wird in dem Zustand WR_H_SET_SMEMW das Signal nPC_SMEMW gesetzt. Dann folgen weitere drei Wartetakte bevor des IO_CHRDY-Sigal abgefragt wird. Danach wird das PC_SMEMW Signal wieder gelöscht und zwei Takte gewartet, bevor die Adressleitungen Null gesetzt und die Steuersignale nISA_SCHRIBEN ud nPC_AEN wieder gelöscht werden. Nach weiteren drei Wartezuständen kehrt der Zustandsautomat in den IDLE-Zustand zurück.

3.1.3.3. Schreibzugriff auf das Lowbyte und das High- und Lowbyte (16-Bit-Schreibzugriff)

Sowohl das Schreiben des Lowbytes, als auch das Schreiben des High- und Lowbytes sind für den ISA-Bus 16-Bit-Schreibvorgänge und unterscheiden sich lediglich in der Adresse. Der Vorgang ähnelt dem Schreibzugriff auf das Highbyte. Es werden jedoch nicht so viele Wartezustände benötigt, dafür muss allerdigs wieder das nPC_MEM16 Signal abgefragt werden (siehe Kapitel 3.1.3.1).

3.2. Verifikation des ISA-Zustandsautomaten

Um die Funktion des ISA-Zustandsautomaten zu überprüfen, wurde zunächst eine Simulation des Automaten durchgeführt. Hierzu wurde eine Testbench erstellt, die den Speicher der CNC nachbildet und die Steuersignale des ISA-Zustandsautomaten so anregt, dass alle Funktionen getestet werden. Nach erfolgreicher Simulation wurde das FPGA der USB-Umsetzter-Karte bespielt und die Signale des ISA-Busses mit einem Oszilloskop nachgemessen.

3.2.1. Simulation

In der Simulation soll die Funktionsweise des ISA-Zustandsautomaten überprüft werden. Hierbei soll die Reihenfolge und der zeitliche Ablauf der Signale des ISA-Busses überprüft werden. Zusätzlich soll getestet werden, ob die Daten an der korrekten Adresse in das RAM der CNC geschrieben werden. Die für die Kommunikation mit dem ISA-Bus notwendigen Zeiten sind in Tabelle 3.1 aufgelistet (Die Zeiten sind der ISA-Timing-Spezifikation in Anhang D zu entnehmen). Einige Zeiten sind bei einem 8-Bit-Zugriff anders, als bei einem 16-Bit-Zugriff. Die Simulation wurde mit dem Programm Active HDL 8.2[3] der Firma Altec durchgeführt. Dies ist eine Simulationssoftware für das Betriebssystem Windows und operiert mit vielen FPGA Herstellern, wie XILINX, ACTEL und natürlich auch Lattice.

3.2.1.1. Testbench

Die Testbench soll das Verhalten des RAMs und dessen Ansteuerung auf der CNC-Karte simulieren. Dazu wird ein RAM simuliert, dessen Werte in eine Textdatei geschrieben werden können. Der Quelltext der Testbench ist in Anhang C.1 zu sehen. Die Testbench besteht im wesentlichen aus zwei Teilen. Zum einen die Simulation des RAMs der CNC, zum anderen die Anregung der Signale des ISA-Zustandsautomat (ISA_COMMUNICATION).

Chipselect (nRAM_CS) und Output Enable (nRAM_OE) gelten, wie auch in der CNC, für das Low- und das Highbyte des RAMs. Ob das Low- oder das Highbyte geschrieben werden soll, wird mit den Signalen nRAM_WH (für das Highbyte) und nRAM_WL (für das Lowbyte) festgelegt.

Die Signale, die von der CNC über den ISA-Bus gesendet werden, sind IO_CHRDY und nPC_MEM16. Das nPC_MEM16-Signal wird sofort nach Anlegen der Adresse und des PC_AEN-Signals gesetzt. IO_CHRDY wird gesetzt, wenn zusätlich noch nPC_SMEMR oder nPC_SMEMW von dem Busmaster gesetzt wurde. Sind diese Signale nicht gesetzt, werden sie von der Testbench in den Tri-State-Zustand geschaltet.

In dem Prozess RAM_SIMULATION (Zeile 326 des Quelltextes in Anhang C.1) wird nun das High- und Lowbyte unabhängig voneinander gelesen oder beschrieben. Der Prozess enthält das Signal nRAM_CS in seiner Sensitivitätsliste und wird damit bei jedem Zustandswechsel dieses Signals aufgerufen.

Um den Inhalt des RAMs nach der Simulation in eine Textdatei zu schreiben, enthält die Testbench noch eine Funktion namens WRITE_RAM_INTO_FILE, welche die Adresse und die dazugehörigen Bits in die Datei schreibt.

[3] http://www.aldec.com/activehdl/

Tabelle 3.3.: Zeitmessungen aus der Simulation für einen 16-Bit-Lesezugriff

Nr. aus Tabelle 3.1	Erwartete Zeiten in ns	Simulierte Zeiten in ns
1	min. 108	124,992
2	min. 39	208,32
3	min. 53	83,328
4	min. 120	208,32
5	min. 102	104,16
6	min. 11	208,32
7	min. 11	104,16
8	min. 70	104,16

Ein Prozess (CLK_SIM) erzeugt den 48MHz Takt für den ISA-Zustandsautomaten und einer (nRST_SIM) erzeugt einen Reset-Impuls. Ein weiterer Prozess (STIMULI) regt die Eingangssignale des ISA-Zustandsautomaten an. Als erstes wird ein 16-Bit-Schreibzugriff an der Adresse 0 durchgeführt. Hierzu wird zunächst die Adresse an das ISA_ADRESS Signal gelegt, der ISA_DATA_OUT_BUFFER mit einem Wert von 0102_h beschrieben und das ISA_WE_HL-Signal gesetzt. Anschließend wird solange gewartet, bis das Signal ISA_BUSY wieder auf '0' geht. Nach dem 16-Bit-Schreibzugriff wird nun noch ein Schreibzugriff auf ein Highbyte (Adresse: 1 Daten: FF_h) und ein Schreibzugriff auf das Lowbyte (Adresse: 1 Daten: 88_h) durchgeführt. Nach einem Lesezugriff an der Adresse 1_h wird nun der Inhalt des RAMs in die Textdatei geschrieben.

3.2.1.2. Funktionale Simulation

Mit Hilfe der Simulationssoftware können die Zeiten zwischen zwei Ereignissen bestimmt werden. Die Ergebnisse der Simulation sind in den Tabellen 3.3 bis 3.5 und in den Abbildungen 3.1 bis 3.3 zu sehen.

Wie an den simulierten Zeiten zu erkennen ist, erfüllt der ISA-Zustandsautomat die ISA-Spezifikationen.

3.2.2. Verifizieren der Hardware

In Hardware können leider nicht alle Zeitvorgaben getestet werden, da nicht festgestellt werden kann, wann der Zustandsautomat in Zustände geht, in denen Signale abgefragt werden. Die Signale wurden direkt an dem ISA-Bus mit einem „Tektronix TPS 2024“ Oszilloskop

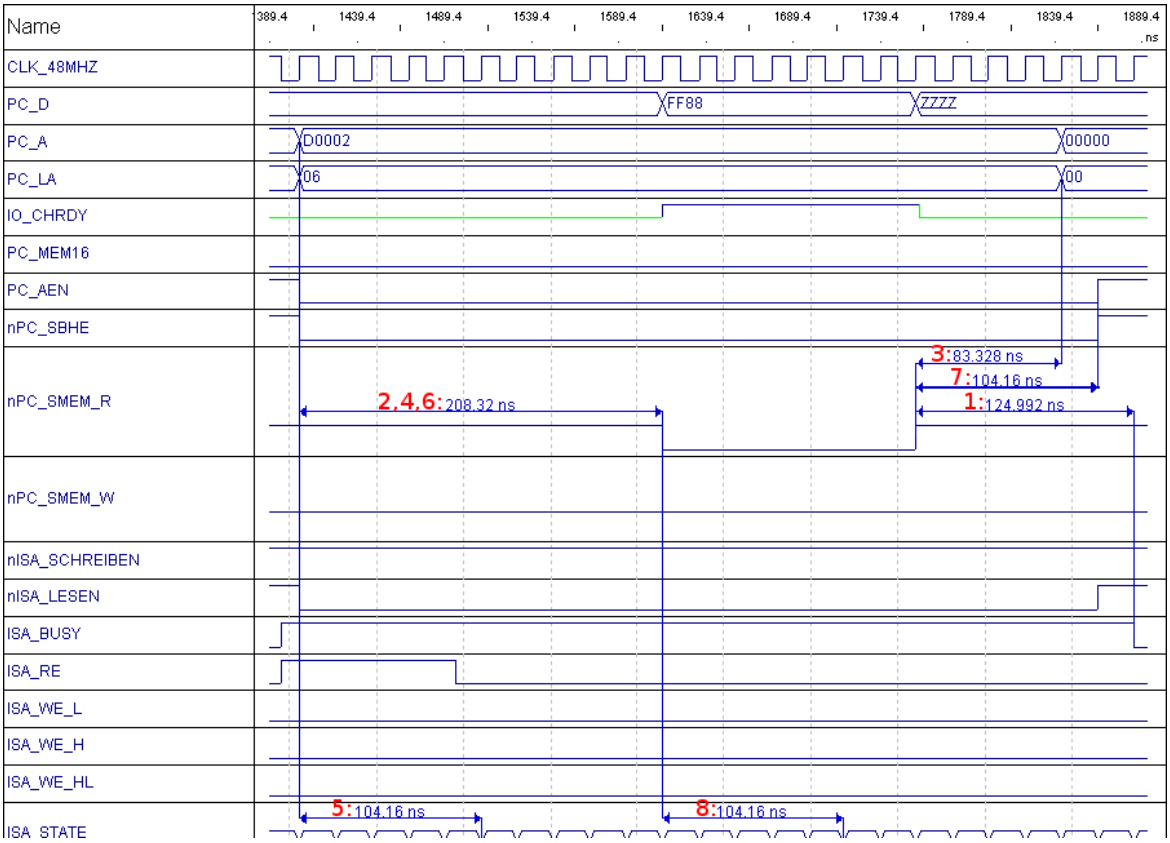


Abbildung 3.1.: Zeitmessung aus der Simulation für einen 16-Bit-Lesezugriff. Ein Vergleich zwischen den simulierten und erwarteten Zeiten ist in Tabelle 3.3 zu sehen. Die roten Zahlen entsprechen dabei der Nummerierung aus der Tabelle.

Tabelle 3.4.: Zeitmessungen aus der Simulation für einen 16-Bit-Schreibzugriff

Nr. aus Tabelle 3.1	Erwartete Zeiten in ns	Simulierte Zeiten in ns
1	min. 108	124,992
2	min. 39	187,488
3	min. 53	83,328
4	min. 120	187,488
5	min. 102	104,16
6	min. 11	187,488
7	min. 11	83,328
8	min. 70	83,328

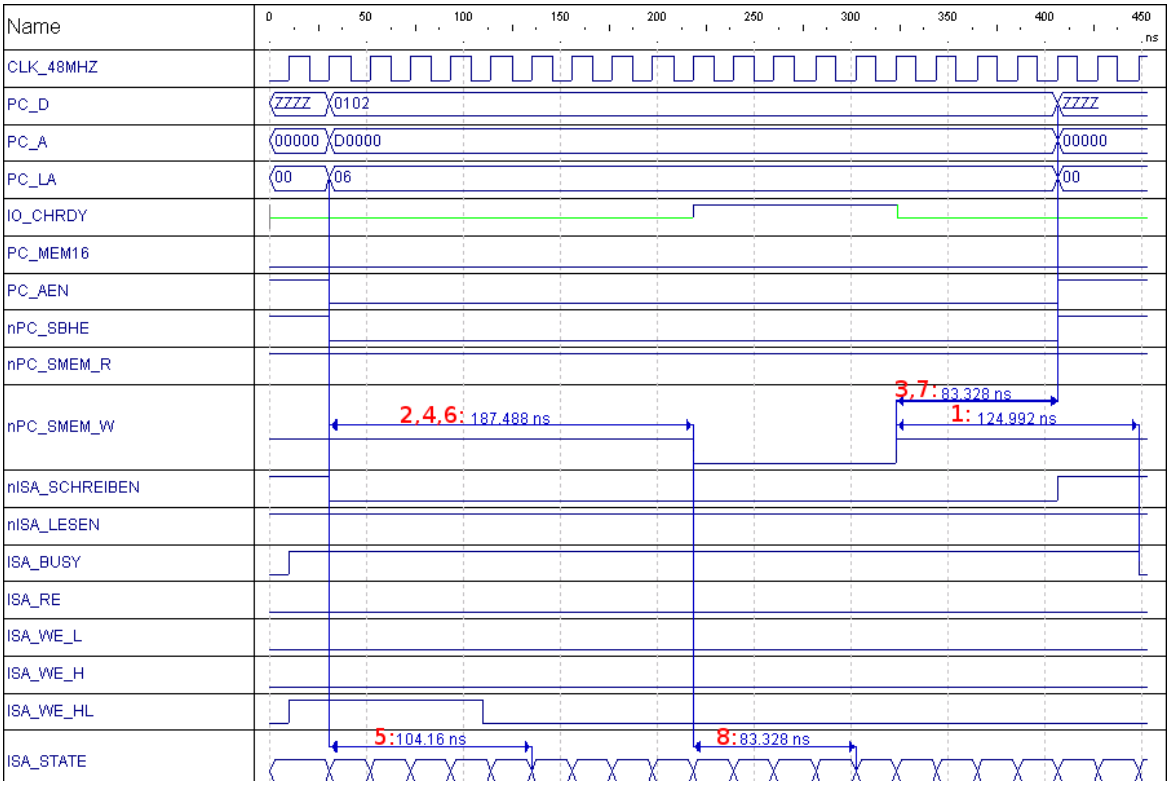


Abbildung 3.2.: Zeitmessung aus der Simulation für einen 16-Bit-Schreibzugriff. Ein Vergleich zwischen den simulierten und erwarteten Zeiten ist in Tabelle 3.4 zu sehen. Die roten Zahlen entsprechen dabei der Nummerierung aus der Tabelle.

Tabelle 3.5.: Zeitmessungen aus der Simulation für einen 8-Bit-Schreibzugriff

Nr. aus Tabelle 3.1	Erwartete Zeiten in ns	Simulierte Zeiten in ns
1	min. 170	166,656
2	min. 102	208,32
3	min. 53	62,496
4	min. 83	208,32
6	min. 11	208,32
7	min. 11	62,496
8	min. 60	83,328

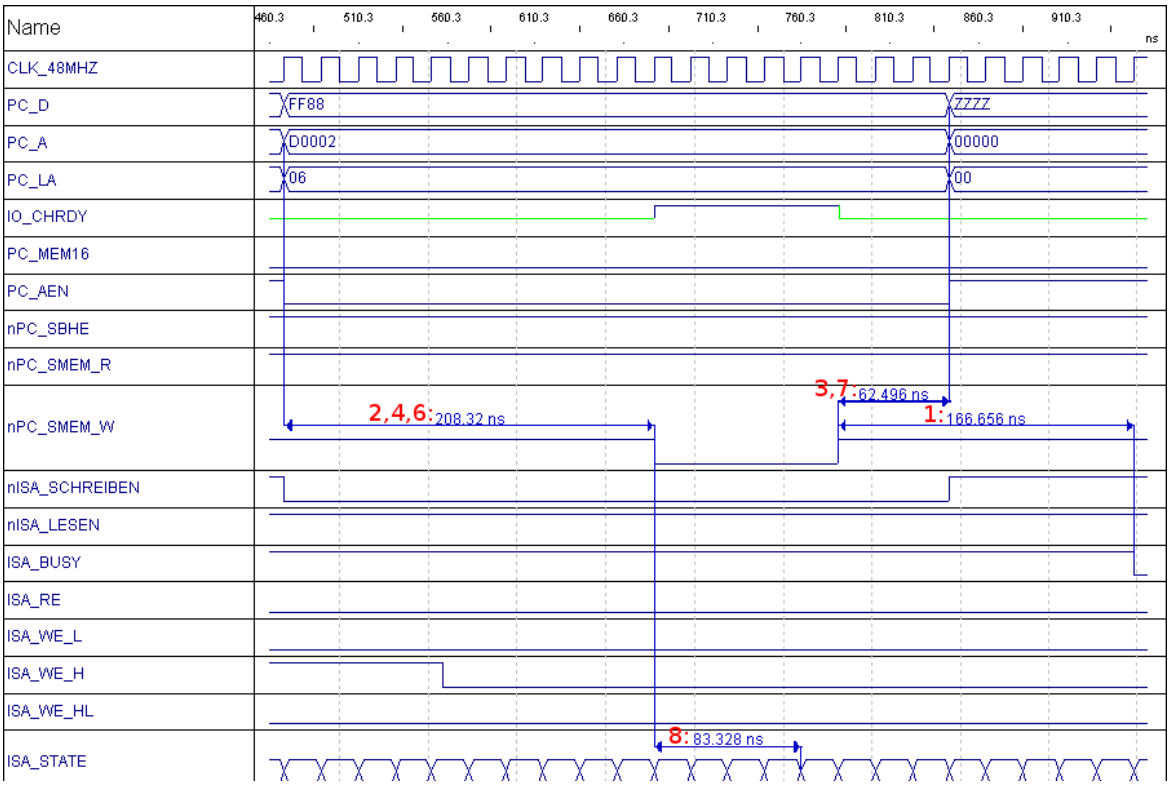


Abbildung 3.3.: Zeitmessung aus der Simulation für einen 8-Bit-Schreibzugriff. Ein Vergleich zwischen den simulierten und erwarteten Zeiten ist in Tabelle 3.5 zu sehen. Die roten Zahlen entsprechen dabei der Nummerierung aus der Tabelle.

gemessen. Die Ergebnisse der Messung sind in den Tabellen in Kapitel 3.2.3 und in den Abbildungen 3.4 bis 3.11 zu sehen.

3.2.2.1. Lesezugriff

In Abbildung 3.4 ist zu sehen, dass die Signale nPC_AEN (CH2), nPC_SBHE (CH3) und nISA_LESEN (CH4) 208ns vor nPC_SMEMR (CH1) anliegen. Dass diese Signale noch 104ns nach dem Löschen von nPC_SMEMR anliegen, ist in Abbildung 3.5 zu sehen.

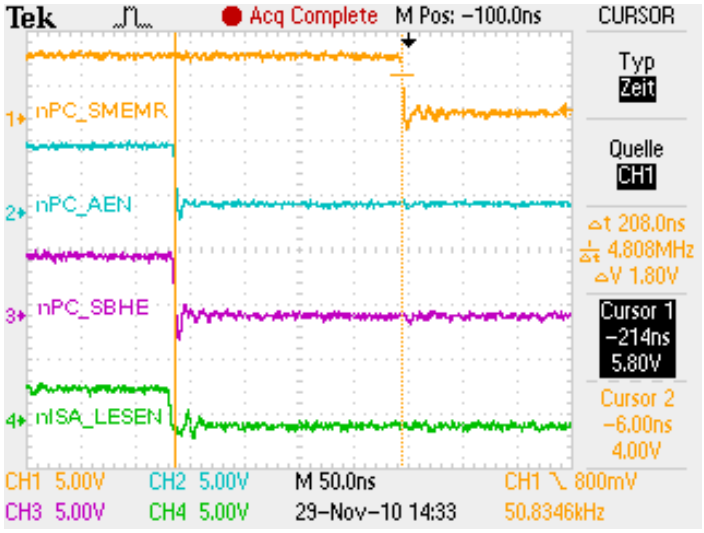


Abbildung 3.4.: Zeitmessung des Lesezugriffes mit Oszilloskop an ISA-Bus (nPC_SMEMR setzen)

In Abbildung 3.6 ist die Antwort der CNC über den ISA-Bus während eines Lesezugriffes zu sehen. Es ist zu erkennen, dass die CNC sofort nach dem Setzen des Signals nPC_AEN (CH3) mit dem Signal IOCHRDY (CH4) signalisiert, dass die Übertragung erfolgreich ist.

3.2.2.2. 16-Bit-Schreibzugriff

In Abbildung 3.7 ist zu sehen, dass 168ns vor dem Setzen von nPC_SMEMW (CH1) schon die Signale nPC_AEN (CH2), nPC_SBHE (CH3) und nISA_SCHREIBEN (CH4) gesetzt wurden. Die Abbildung 3.8 zeigt, dass diese Signale 84ns nach dem Löschen von nPC_SMEMW ebenfalls gelöscht werden.

Abbildung 3.9 zeigt die Antwort der CNC auf einen 16-Bit-Schreibzugriff. Es ist zu sehen, dass die CNC auf eine Anfrage auf einen 16-Bit-Zugriff (nPC_SBHE; CH1) sofort mit dem

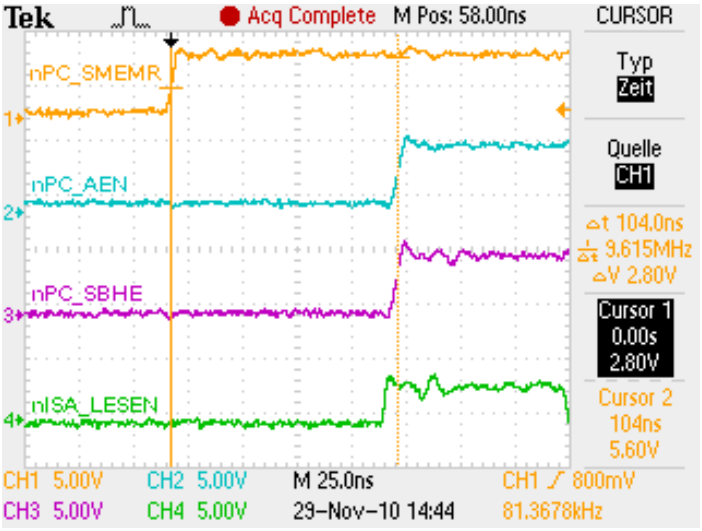


Abbildung 3.5.: Zeitmessung des Lesezugriffes mit Oszilloskop an ISA-Bus (nPC_SMEMR löschen)

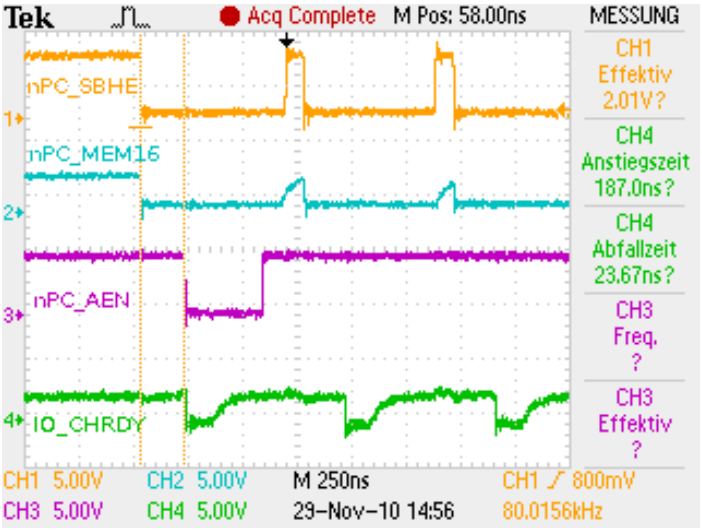


Abbildung 3.6.: Zeitmessung des Lesezugriffes mit Oszilloskop an ISA-Bus (Antwort der CNC)

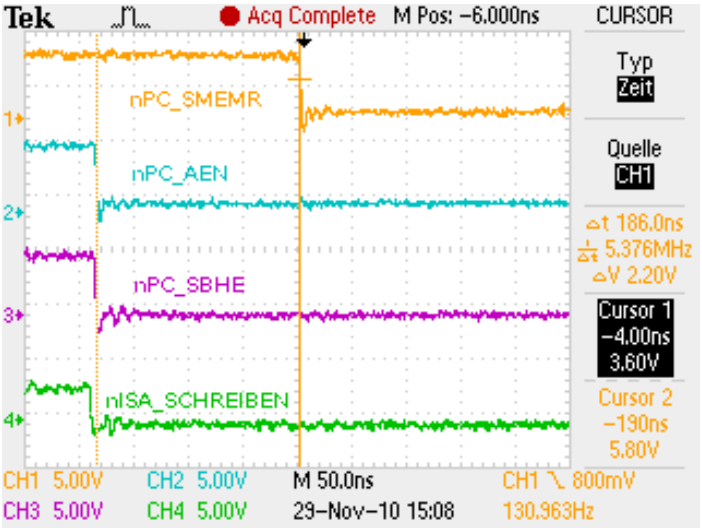


Abbildung 3.7.: Zeitmessung des 16-Bit-Schreibzugriffes mit Oszilloskop an ISA-Bus (nPC_SMEMW setzen)

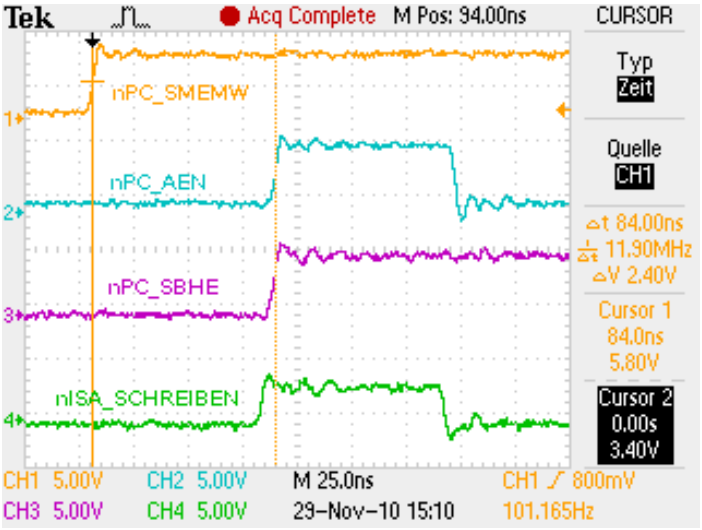


Abbildung 3.8.: Zeitmessung des 16-Bit-Schreibzugriffes mit Oszilloskop an ISA-Bus (nPC_SMEMW löschen)

Signal nPCMEM16 (CH2) beantwortet. Und auch das Signal IOCHRDY (CH4) wird unverzüglich nach Setzen des nPC_SMEMW-signals (CH3) von der CNC gesetzt.

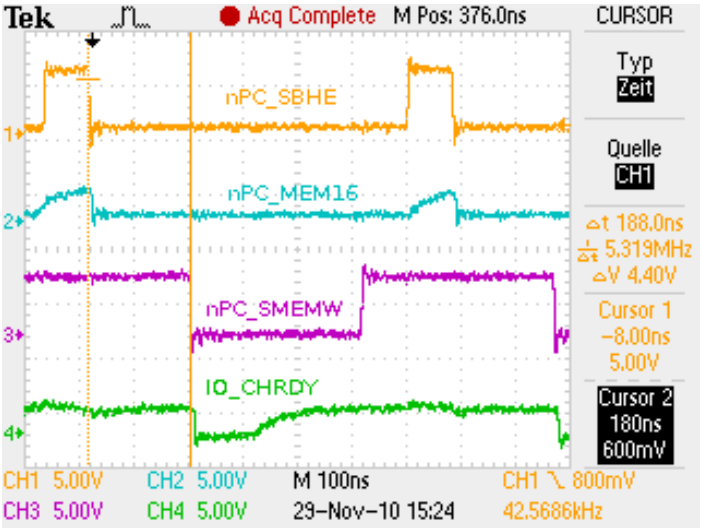


Abbildung 3.9.: Zeitmessung des 16-Bit-Schreibzugriffes mit Oszilloskop an ISA-Bus (Antwort der CNC)

3.2.2.3. 8-Bit-Schreibzugriff

In Abbildungen 3.10 und 3.11 ist zu sehen, dass das Signal nPC_AEN (CH2) 204ns vor dem Setzen und 64ns nach dem Löschen von nPC_SMEMW (CH1) aktiv ist. Ebenfalls zu erkennen ist, dass das Signal nPC_SBHE nicht gesetzt wird, da keine 16-Bit-Kommuniktion ansteht.

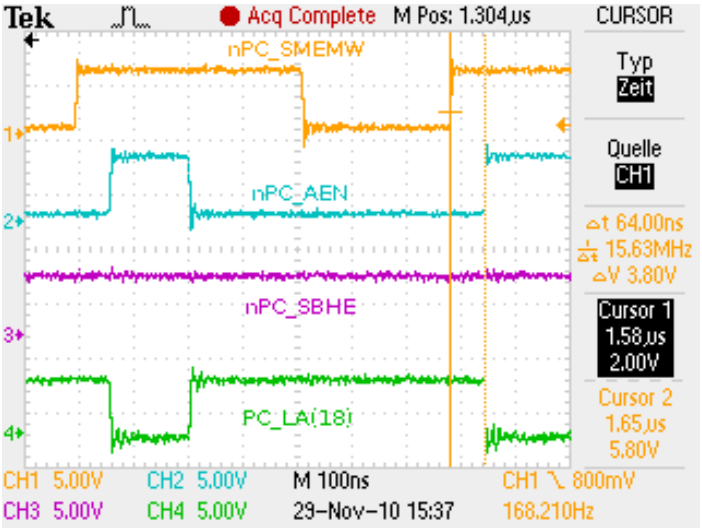


Abbildung 3.10.: Zeitmessung des 8-Bit-Schreibzugriffes mit Oszilloskop an ISA-Bus (nPC_SMEMW setzen)

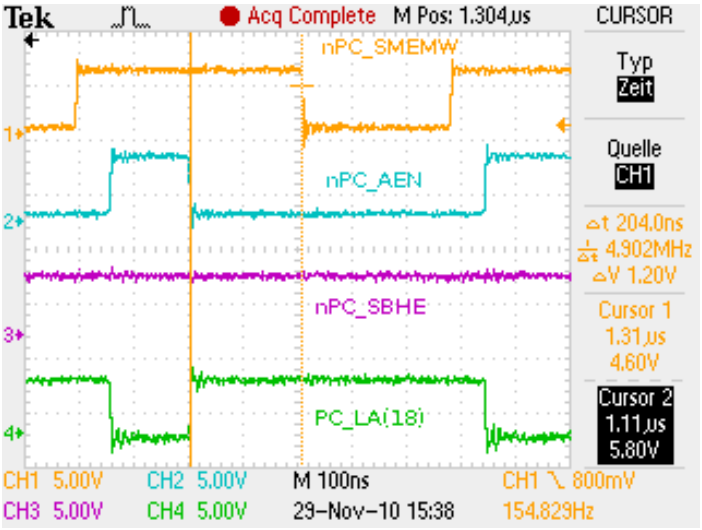


Abbildung 3.11.: Zeitmessung des 8-Bit-Schreibzugriffes mit Oszilloskop an ISA-Bus (nPC_SMEMW löschen)

Tabelle 3.6.: Vergleich der Zeiten eines 16-Bit-Lesezugriffes

Zeiten aus Spezifikation in ns	Simulierte Zeiten in ns	in Hardware gemessene Zeiten in ns
min. 108	124,992	nicht messbar
min. 39	208,32	208,0
min. 120	208,32	208,0
min. 102	104,16	nicht messbar
min. 11	208,32	208,0
min. 11	104,16	104,0
min. 70	104,16	nicht messbar

3.2.3. Vergleich der Simulations- und Messergebnisse

Der Verglich zwischen den Zeiten der Simulation und der Messung an der Hardware zeigt nur geringfügige Unterschiede, die auf Messungenauigkeiten und Laufzeiten der Hardware zurückzuführen sind.

Bei den Signalverläufen fällt nur ein Unterschied bei den Signalen auf, die von der CNC erzeugt werden. Wie zum Beispiel bei dem nPC_MEM16 Signal (CH2) in Abbildung 3.6. Hier ist zu sehen, dass das Signal langsam von einem Low- in einen Highpegel übergeht. Das liegt daran, dass das Signal nicht auf logisch Eins gesetzt wird, sondern in den Tri-State-Zustand übergeht.

In Tabellen 3.6 bis 3.8 ist ein Vergleich zwischen den Zeiten aus der Simulation, aus der Messung an der Hardware sowie aus der ISA-Spezifikation zu sehen. Wie zu erkennen ist, werden alle Zeiten korrekt eingehalten. In einigen Fällen werden diese Zeiten sogar überschritten. Dies macht die ISA-Kommunikation allerdings nur unbedeutend langsamer.

3.2.4. Funktionstest

Die USB-Umsetzter-Karte wurde zu Testzwecken in eine CNC46 und in eine CNC48 eingebaut. An die USB-Umsetzter-Karte wurde ein Computer mit der benötigten Sieb & Meyer Software angeschlossen und ein CNC-Programm auf die CNC überspielt. Anschließend wurde dieses Programm an einer CNC-Leiterplattenbohrmaschine ausgeführt. Das Programm ließ sich problemlos auf die CNC übertragen und auch abarbeiten. Damit ist die Funktion der USB-Umsetzter-Karte nachgewiesen.

Tabelle 3.7.: Vergleich der Zeiten eines 16-Bit-Schreibzugriffes

Zeiten aus Spezifikation in ns	Simulierte Zeiten in ns	in Hardware gemessene Zeiten in ns
min. 108	124,992	nicht messbar
min. 39	187,488	186,0
min. 53	83,328	84,0
min. 120	187,488	186,0
min. 102	104,16	nicht messbar
min. 11	187,488	186,0
min. 11	83,328	84,0
min. 70	83,328	nicht messbar

Tabelle 3.8.: Vergleich der Zeiten eines 8-Bit-Schreibzugriffes

Zeiten aus Spezifikation in ns	Simulierte Zeiten in ns	in Hardware gemessene Zeiten in ns
min. 170	166,656	nicht messbar
min. 102	208,32	204,0
min. 53	62,496	64,0
min. 83	208,32	204,0
min. 11	208,32	204,0
min. 11	62,496	64,0

4. Zusammenfassung und Ausblick

Die USB-Umsetzter-Karte sollte es ermöglichen, die CNC46/48 mit einem gewöhnlichen Computer, anstelle des internen Computers der CNC, zu betreiben. Diese Aufgabe erfüllt die USB-Umsetzter-Karte vollständig. So ist es nun möglich CNC46 und CNC48 Steuerungen, in denen der Computer defekt ist, weiterhin zu betreiben.

Um die Effizienz der ISA-Kommunikation zu steigern, könnte der Zustandsautomat noch näher an die Eigenschaften der CNC46 und CNC48 angepasst werden. So sind einige Wartezeiten in den ISA-Spezifikationen länger, als sie für die Kommunikation aufgrund der Beschaffenheit der CNC, notwendig währen.

Für zukünftige Steuerungen empfehle ich gleich eine Schnittstelle vorzusehen, so dass der Nutzer die Möglichkeit besitzt, einen eigenen Computer an die Steuerung anzuschließen.

Literaturverzeichnis

[Ampro Computers 1998] AMPRO COMPUTERS, Inc.: *ISA Bus Timing Diagrams*. Revision A, 1998

[Anderson und Shanley 1999] ANDERSON, Don ; SHANLEY, Tom: *ISA System Architecture*. Third Edition. Addison-Wesley Professional, 1999. – ISBN 0-2014-0996-8

[Axelson 2006] AXELSON, Jan: *USB 2.0 Handbuch für Entwickler*. 3., 2. überarbeitete Auflage. mitp, November 2006. – ISBN 3-8266-1690-1

[Cypress Semiconductor Corporation 2005] CYPRESS SEMICONDUCTOR CORPORATION: *CY7C68013A, CY7C68014A, CY7C68015A, CY7C68016A*, September 2005

[Fairchild Semiconductor Corporation 2003] FAIRCHILD SEMICONDUCTOR CORPORATION: *FAN1086*. REV. 1.0.7, Oktober 2003

[Gajski 1997] GAJSKI, Daniel D.: *Principles Of Digital Design*. Prentice Hall, 1997. – ISBN 0-13-301144-5

[IC Microsystems 1991] IC MICROSYSTEMS: *X24C04*, 1991

[Kohm und Jens-Uwe-Morawski 2010] KOHM, Markus ; JENS-UWE-MORAWSKI: *KOMA - Script*. 3. Auflage, September 2010

[Lattice Semiconductor Corp. 2000] LATTICE SEMICONDUCTOR CORP.: *ispMACH™4A CPLD Family*. Rev: D, August 2000

[Lattice Semiconductor Corp. 2009] LATTICE SEMICONDUCTOR CORP.: *LatticeXP2™ Family Handbook*. Version 02.4, Mai 2009

[Lattice Semiconductor Corp. 2010] LATTICE SEMICONDUCTOR CORP.: *Lattice Diamond Tutorial*, Juni 2010

[Moses 2007] MOSES, Brooks: *The Listings Package*. Version 1.4, Februar 2007

[National Semiconductor Corporation 2002] NATIONAL SEMICONDUCTOR CORPORATION: *LM809/LM810*, September 2002

[Philips Semiconductors 1990] PHILIPS SEMICONDUCTORS: *74HC/HCT541*, Dezember 1990

[Philips Semiconductors 1997] PHILIPS SEMICONDUCTORS: *74HC/T High-Speed CMOS User Guide*, November 1997

[Reichardt und Schwarz 2000] REICHARDT, Jürgen ; SCHWARZ, Bernd: *VHDL-Synthese*. 4. Auflage. Oldenbourg, 2000. – ISBN 3-48-658192-9

[SAMSUNG Electronics 1996] SAMSUNG ELECTRONICS: *KM68257C/CL CMOS SRAM*. Rev 3.0, Februar 1996

[Schlosse 2009] SCHLOSSE, Joachim: *Wissenschaftliche Arbeiten schreiben mit LaTeX*. 3., überarbeitete Auflage. mitp, 2009. – ISBN 3-82-665892-2

[Tantau 2008] TANTAU, Till: *TikZ & PGF*. Manual for version 2.00, Februar 2008

[Texas Instruments Incorporated 2002] TEXAS INSTRUMENTS INCORPORATED: *Ultralow-Noise, High PSRR, Fast RF 200mA LDO Linear Regulators*. Rev. C, April 2002

[Woitowitz und Urbanski 2007] WOITOWITZ, Roland ; URBANSKI, Klaus: *Digitaltechnik*. 5., neu bearb. u. erw. Aufl. Springer, 2007. – ISBN 978-3-540-73672-1

A. ASM-Diagramme des ISA-Zustandsautomaten

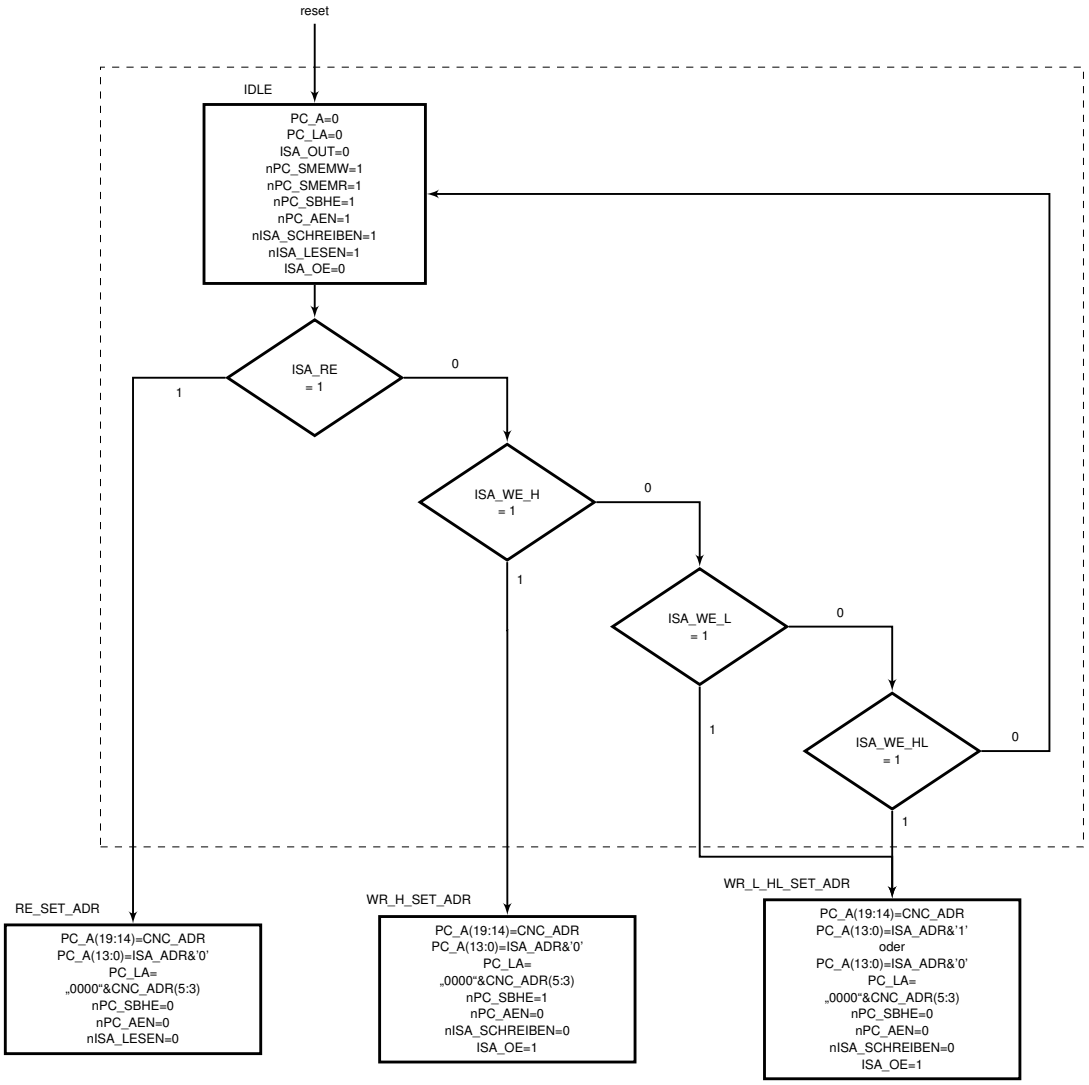


Abbildung A.1.: ASM-Diagramm des ISA-Zustandsautomaten I

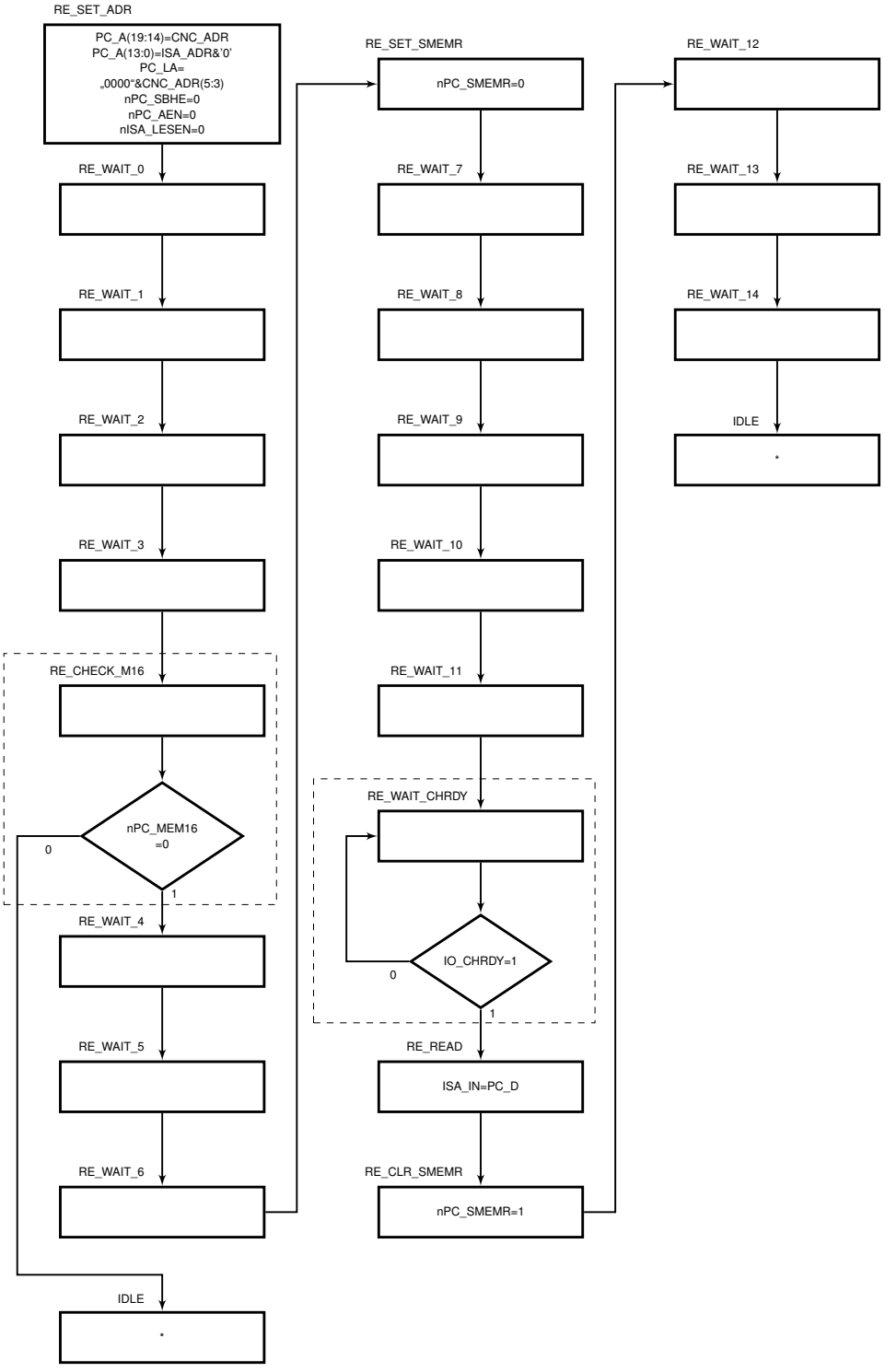


Abbildung A.2.: ASM-Diagramm des ISA-Zustandsautomaten II

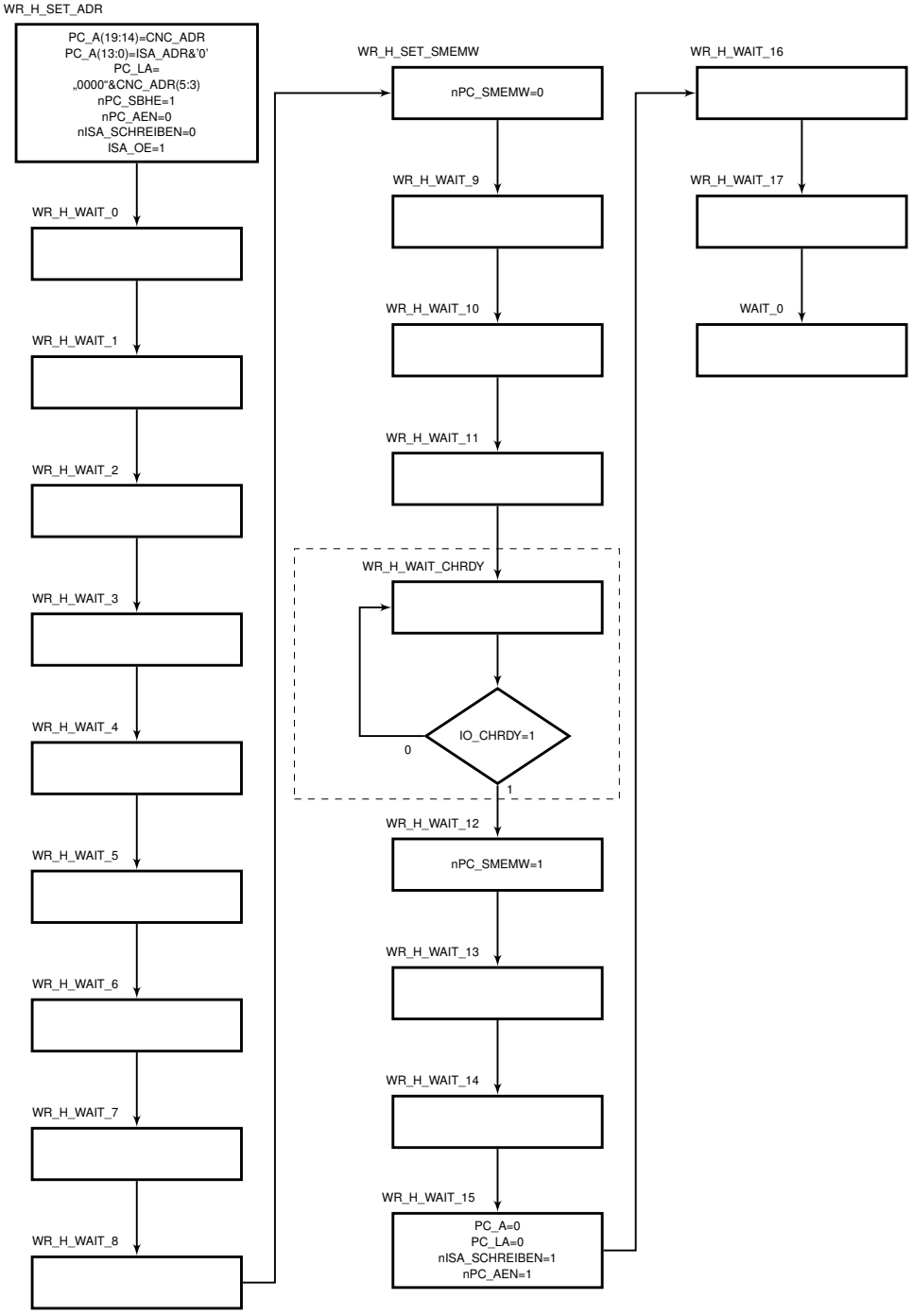


Abbildung A.3.: ASM-Diagramm des ISA-Zustandsautomaten III

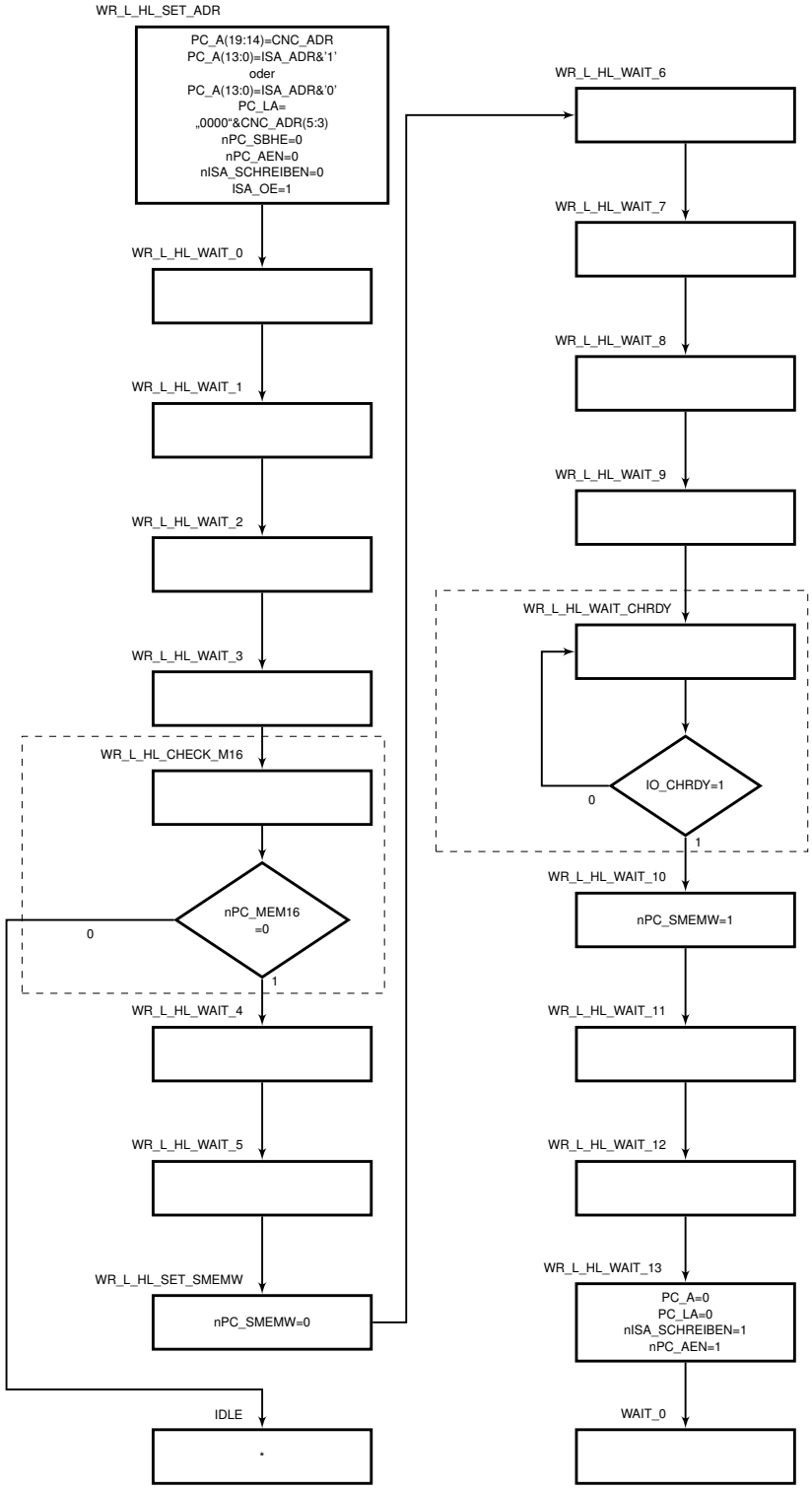


Abbildung A.4.: ASM-Diagramm des ISA-Zustandsautomaten IV

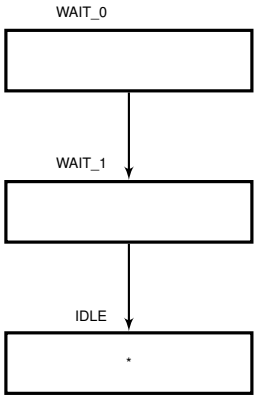


Abbildung A.5.: ASM-Diagramm des ISA-Zustandsautomaten V

B. Quelltext des ISA-Zustandsautomaten

Quelltext B.1: VHDL-Quelltext des ISA-Zustandsautomaten

```
--================================================================
-- ISA.vhd
--       ISA-Bus-Kommunikation fuer 045010023 USB-Umsetzer
--
-- Author    : Daniel Sabotta
--
-- Date      : 24.11.2010
--
-- Version   : 1.0
--
--================================================================

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity ISA_COMMUNICATION is

    port(
        -- Reset (low aktiv)
        nRESET          : in    std_logic;
        -- 48MHz Takt
        CLK_48MHZ       : in    std_logic;

        ------------------------
        -- Pins zum ISA-Bus --
        ------------------------
        -- Dateinleitungen
        -- System Data bits
        PC_D            : inout std_logic_vector (15 downto 0);

        -- Adressleitungen (Bits 17, 18 und 19 kommen in beden vor!)
        -- Adressbits (niederwertige Adressbits 0 bis 19)
        PC_A            : out std_logic_vector (19 downto 0);
        -- Latched Adressbits (hoeherwertige Adressbits 17 bis 23)

        -- Steuerleitungen
        -- Channel Ready
        IO_CHRDY        : in std_logic;
        -- Memory Chip Select 16
        nPC_MEM16       : in std_logic;
        -- Address Enable
```

```
44          PC_AEN              : out std_logic;
            -- System Byte High Enable
46          nPC_SBHE            : out std_logic;
            -- System Memory Read
48          nPC_SMEM_R          : out std_logic;
            -- System Memory Write
50          nPC_SMEM_W          : out std_logic;

52          -- Signale fuer Bustreiber
            -- schreibe auf Datenleitungen
54          nISA_SCHREIBEN      : out std_logic;
            -- Lese von Dateinleitungen
56          nISA_LESEN          : out std_logic;

58          ------------------------------------
            -- Signale zum COMMAND-HANDLER --
60          ------------------------------------
            -- Steuersignale der ISA_COMMUNICATION
62          -- High- und Lowbyte lesen
            ISA_RE              : in std_logic;
64          -- High- und Lowbyte schreiben
            ISA_WE_HL           : in std_logic;
66          -- Highbyte schreiben
            ISA_WE_H            : in std_logic;
68          -- Lowbyte schreiben
            ISA_WE_L            : in std_logic;
70
            -- Daten, die ueber den Bus geschickt werden sollen
72          ISA_DATA_OUT_BUFFER : in std_logic_vector (15 downto 0);
            -- Eingehende Daten
74          ISA_DATA_IN_BUFFER  : out std_logic_vector (15 downto 0);
            -- Adresse
76          ISA_ADDRESS         : in std_logic_vector (12 downto 0);

78          -- ISA-Handler ist beschaeftigt
            ISA_BUSY            : out std_logic
80          );
    end ISA_COMMUNICATION;
82
    ------------------------------------------------------------------------

84
    architecture BEHAVIORAL of ISA_COMMUNICATION is
86      ------------------------------------
        -- Signale fuer den ISA-Handler --
```

```
88      ------------------------------------
        type ISA_STATES is (S_ISA_IDLE,
90
                            -- Status fuer das Lesen (16 Bit)
92                          S_ISA_RE_SET_ADR,

94                          S_ISA_RE_WAIT_0,
                            S_ISA_RE_WAIT_1,
96                          S_ISA_RE_WAIT_2,
                            S_ISA_RE_WAIT_3,
98
                            S_ISA_RE_CHECK_M16,
100
                            S_ISA_RE_WAIT_4,
102                         S_ISA_RE_WAIT_5,
                            S_ISA_RE_WAIT_6,
104
                            S_ISA_RE_SET_SMEMR,
106
                            S_ISA_RE_WAIT_7,
108                         S_ISA_RE_WAIT_8,
                            S_ISA_RE_WAIT_9,
110                         S_ISA_RE_WAIT_10,
                            S_ISA_RE_WAIT_11,
112
                            S_ISA_RE_WAIT_CHRDY,
114                         S_ISA_RE_READ,
                            S_ISA_RE_CLR_SMEMR,
116
                            S_ISA_RE_WAIT_12,
118                         S_ISA_RE_WAIT_13,
                            S_ISA_RE_WAIT_14,
120

122                         -- Status fuer das Schreiben des Highbytes
                            -- 8Bit
124                         S_ISA_WR_H_SET_ADR_DATA,

126                         S_ISA_WR_H_WAIT_0,
                            S_ISA_WR_H_WAIT_1,
128                         S_ISA_WR_H_WAIT_2,
                            S_ISA_WR_H_WAIT_3,
130                         S_ISA_WR_H_WAIT_4,
                            S_ISA_WR_H_WAIT_5,
```

```
132                 S_ISA_WR_H_WAIT_6,
                    S_ISA_WR_H_WAIT_7,
134                 S_ISA_WR_H_WAIT_8,

136                 S_ISA_WR_H_SET_SMEMW,

138                 S_ISA_WR_H_WAIT_9,
                    S_ISA_WR_H_WAIT_10,
140                 S_ISA_WR_H_WAIT_11,

142                 S_ISA_WR_H_WAIT_CHRDY,

144                 S_ISA_WR_H_WAIT_12,
                    S_ISA_WR_H_WAIT_13,
146                 S_ISA_WR_H_WAIT_14,

148                 S_ISA_WR_H_WAIT_15,
                    S_ISA_WR_H_WAIT_16,
150                 S_ISA_WR_H_WAIT_17,

152
                    -- Status fuer das Schreiben von 16 Bit
154                 --      oder des Lowbytes
                    S_ISA_WR_LH_L_SET_ADR_DATA,
156
                    S_ISA_WR_LH_L_WAIT_0,
158                 S_ISA_WR_LH_L_WAIT_1,
                    S_ISA_WR_LH_L_WAIT_2,
160                 S_ISA_WR_LH_L_WAIT_3,

162                 S_ISA_WR_LH_L_CHECK_M16,

164                 S_ISA_WR_LH_L_WAIT_4,
                    S_ISA_WR_LH_L_WAIT_5,
166
                    S_ISA_WR_LH_L_SET_SMEMW,
168
                    S_ISA_WR_LH_L_WAIT_6,
170                 S_ISA_WR_LH_L_WAIT_7,
                    S_ISA_WR_LH_L_WAIT_8,
172                 S_ISA_WR_LH_L_WAIT_9,

174                 S_ISA_WR_LH_L_WAIT_CHRDY,
```

```
176                             S_ISA_WR_LH_L_WAIT_10,
                                S_ISA_WR_LH_L_WAIT_11,
178                             S_ISA_WR_LH_L_WAIT_12,

180                             S_ISA_WR_LH_L_WAIT_13,

182                             -- Wartezyklen vor neuem Schreib-/Lesezyklus
                                S_ISA_WAIT_CYCLE_0,
184                             S_ISA_WAIT_CYCLE_1);

186     signal ISA_STATE : ISA_STATES;

188     -- Enable-Signal fuer Tri State Ausgangsstufe
        signal ISA_OE                   : std_logic;
190
        -- einsynchronisiertes Channel Ready (IO_CHRDY)
192     signal IO_CHRDY_INT             : std_logic;
        -- einsynchronisiertes Memory Chip Select 16 (PC_MEM16)
194     signal nPC_MEM16_INT            : std_logic;
        -- Hilfssignal zum Synchronisieren
196     signal SYNC_IO_CHRDY            : std_logic;
        -- Hilfssignal zum Synchronisieren
198     signal nSYNC_PC_MEM16           : std_logic;
begin
200     ------------------------------------------
        -- Eingangssignale synchronisieren (2FF) --
202     ------------------------------------------
        SYNC: process (CLK_48MHZ, nRESET)
204     begin
            if (nRESET = '0') then
206             nPC_MEM16_INT <= '1';
                nSYNC_PC_MEM16 <= '1';
208             IO_CHRDY_INT <= '0';
                SYNC_IO_CHRDY <= '0';
210         elsif (CLK_48MHZ = '1' and CLK_48MHZ'event) then
                -- IO_CHRDY synchronisieren
212             IO_CHRDY_INT <= SYNC_IO_CHRDY;
                SYNC_IO_CHRDY <= IO_CHRDY;
214
                --PC_MEM16 synchronisieren
216             nPC_MEM16_INT <= nSYNC_PC_MEM16;
                nSYNC_PC_MEM16 <= nPC_MEM16;
218         end if;
        end process SYNC;
```

```
220
        ------------------------
222     -- ISA-Kommunikation --
        ------------------------
224
        -- Zeigt, ob der ISA-Handler beschaeftigt ist
226     ISA_BUSY <= '0' when (ISA_STATE = S_ISA_IDLE and ISA_RE = '0' and
            ISA_WE_HL = '0' and ISA_WE_H = '0' and ISA_WE_L = '0') else '1';

228     -- Tri-State Ausgangstreiber
        PC_D <= ISA_DATA_OUT_BUFFER when (ISA_OE = '1') else (others =>'Z');
230
        ISA_COMMUNICATION: process (CLK_48MHZ, nRESET)
232     begin
            if (nRESET = '0') then
234             ISA_STATE <= S_ISA_IDLE;

236             PC_A <= (others => '0');
                PC_LA <= (others => '0');
238             PC_AEN <= '1';
                nPC_SBHE <= '1';
240             nPC_SMEM_R <= '1';
                nPC_SMEM_W <= '1';
242             nISA_SCHREIBEN  <= '1';
                nISA_LESEN  <= '1';
244             ISA_OE <= '0';

246         elsif (CLK_48MHZ = '1' and CLK_48MHZ'event) then

248             -- Standard-Werte
                PC_AEN <= '1';
250             nPC_SBHE <= '1';
                nPC_SMEM_R <= '1';
252             nPC_SMEM_W <= '1';
                nISA_SCHREIBEN  <= '1';
254             nISA_LESEN  <= '1';
                ISA_OE <= '0';
256
                case ISA_STATE is
258                 when S_ISA_IDLE =>
                        PC_A <= (others => '0');
260                     PC_LA <= (others => '0');
                        PC_AEN <= '1';
262                     nPC_SBHE <= '1';
```

```
                nPC_SMEM_R <= '1';
264             nPC_SMEM_W <= '1';
                nISA_SCHREIBEN <= '1';
266             nISA_LESEN <= '1';
                ISA_OE <= '0';
268
                if (ISA_RE = '1') then
270                 ISA_STATE <= S_ISA_RE_SET_ADR;

272                 -- Adresse anlegen
                    -- Adressbereich beginnt bei D0000h
274                 -- PC_A und PC_LA Adressen ueberschneiden sich
                    PC_A(19 downto 14) <= "110100";   -- 1101b = Dh
276                 PC_A(13 downto 0) <= ISA_ADDRESS & '0';
                    -- die untersten 3 Bits sind gleich dem
278                 --    obersten der PC_A, die Restichen '0'
                    PC_LA <= "0000110";
280                 -- da 16 Bit ausgelesen werden sollen
                    nPC_SBHE <= '0';
282                 -- Adresse ist gueltig
                    PC_AEN <= '0';
284
                    -- Bustreiber schalten
286                 nISA_LESEN <= '0';
                    nISA_SCHREIBEN <= '1';
288
                elsif (ISA_WE_H = '1') then
290                 ISA_STATE <= S_ISA_WR_H_SET_ADR_DATA;

292                 PC_A(19 downto 14) <= "110100";
                    PC_A(13 downto 0) <= ISA_ADDRESS & '0';
294                 -- die untersten 3 Bits sind gleich dem
                    --    obersten der PC_A, die Restichen '0'
296                 PC_LA <= "0000110";
                    -- da nur das Highbyte (gerade Adresse)
298                 --     geschrieben werden soll
                    nPC_SBHE <= '1';
300                 -- Adresse ist gueltig
                    PC_AEN <= '0';
302                 -- Daten auf den Bus legen
                    ISA_OE <= '1';
304                 nISA_SCHREIBEN <= '0';
                    nISA_LESEN <= '1';
306
```

```
                elsif (ISA_WE_L = '1') then
308                 ISA_STATE <= S_ISA_WR_LH_L_SET_ADR_DATA;

310                 PC_A(19 downto 14) <= "110100";
                    PC_A(13 downto 0) <= ISA_ADDRESS & '1';
312                 -- die untersten 3 Bits sind gleich dem
                    --     obersten der PC_A, die Restlichen '0'
314                 PC_LA <= "0000110";
                    ISA_OE <= '1';
316                 nPC_SBHE <= '0';
                    PC_AEN <= '0';
318                 nISA_SCHREIBEN <= '0';

320             elsif (ISA_WE_HL = '1') then
                    ISA_STATE <= S_ISA_WR_LH_L_SET_ADR_DATA;
322
                    PC_A(19 downto 14) <= "110100";
324                 PC_A(13 downto 0) <= ISA_ADDRESS & '0';
                    -- die untersten 3 Bits sind gleich dem
326                 --     obersten der PC_A, die Restlichen '0'
                    PC_LA <= "0000110";
328                 ISA_OE <= '1';
                    nPC_SBHE <= '0';
330                 PC_AEN <= '0';
                    nISA_SCHREIBEN <= '0';
332
                else
334                 ISA_STATE <= S_ISA_IDLE;
                end if;
336

                ------------------
338             -- Lesezugriff --
                ------------------
340         when S_ISA_RE_SET_ADR =>
                nPC_SBHE <= '0';
342             PC_AEN <= '0';
                nISA_LESEN <= '0';
344             ISA_STATE <= S_ISA_RE_WAIT_0;

346         -- 4 Takte warten (min 66ns)
            when S_ISA_RE_WAIT_0 =>
348             nPC_SBHE <= '0';
                PC_AEN <= '0';
350             nISA_LESEN <= '0';
```

```
                ISA_STATE <= S_ISA_RE_WAIT_1;
352
            when S_ISA_RE_WAIT_1 =>
354             nPC_SBHE <= '0';
                PC_AEN <= '0';
356             nISA_LESEN <= '0';
                ISA_STATE <= S_ISA_RE_WAIT_2;
358
            when S_ISA_RE_WAIT_2 =>
360             nPC_SBHE <= '0';
                PC_AEN <= '0';
362             nISA_LESEN <= '0';
                ISA_STATE <= S_ISA_RE_WAIT_3;
364
            when S_ISA_RE_WAIT_3 =>
366             nPC_SBHE <= '0';
                nISA_LESEN <= '0';
368             PC_AEN <= '0';
                ISA_STATE <= S_ISA_RE_CHECK_M16;
370

372         -- Pruefen ob der Slave zur 16-Bit-Uebertragung
            --     faehig ist
374         when S_ISA_RE_CHECK_M16 =>
                nPC_SBHE <= '0';
376             PC_AEN <= '0';
                nISA_LESEN <= '0';
378             if (nPC_MEM16_INT = '0') then
                    nPC_SBHE <= '0';
380                 ISA_STATE <= S_ISA_RE_WAIT_4;
                else
382                 -- fals die 16 Kommunikation nicht moeglich ist,
                    -- kehrt die State Machine in den Idle
384                 -- Status zurueck, ohne Daten gelesen zu haben.
                    -- Da die 16-bit-Kommunikation mit der CNC auf
386                 -- jedenfall moeglich ist,
                    -- ist diese Abfrage nur fuer die
388                 -- Vollstaendigkeit vorhanden.
                    -- Falls dieser Fall dennoch auftreten sollte,
390                 -- bemerkt der PC diesen Fehler
                    -- (da keine Antwort kommt)
392                 -- und fuert einen Reset durch.
                    PC_A <= (others => '0');
394                 PC_LA <= (others => '0');
```

```
                ISA_STATE <= S_ISA_IDLE;
396         end if;

398       -- 4 Takte warten (min 70 ns)
          when S_ISA_RE_WAIT_4 =>
400         nPC_SBHE <= '0';
            PC_AEN <= '0';
402         nISA_LESEN <= '0';
            nPC_SBHE <= '0';
404         ISA_STATE <= S_ISA_RE_WAIT_5;

406       when S_ISA_RE_WAIT_5 =>
            nPC_SBHE <= '0';
408         PC_AEN <= '0';
            nISA_LESEN <= '0';
410         nPC_SBHE <= '0';
            ISA_STATE <= S_ISA_RE_WAIT_6;
412
          when S_ISA_RE_WAIT_6 =>
414         nPC_SBHE <= '0';
            PC_AEN <= '0';
416         nISA_LESEN <= '0';
            ISA_STATE <= S_ISA_RE_SET_SMEMR;
418
          -- lese Anforderung setzen
420       when S_ISA_RE_SET_SMEMR =>
            nPC_SBHE <= '0';
422         PC_AEN <= '0';
            nISA_LESEN <= '0';
424         nPC_SMEM_R <= '0';
            ISA_STATE <= S_ISA_RE_WAIT_7;
426
          -- 4 Takte warten (min 159ns)
428       when S_ISA_RE_WAIT_7 =>
            nPC_SBHE <= '0';
430         PC_AEN <= '0';
            nISA_LESEN <= '0';
432         nPC_SMEM_R <= '0';
            ISA_STATE <= S_ISA_RE_WAIT_8;
434


436       when S_ISA_RE_WAIT_8 =>
            nPC_SBHE <= '0';
438         PC_AEN <= '0';
```

```
                nISA_LESEN <= '0';
440             nPC_SMEM_R <= '0';
                ISA_STATE <= S_ISA_RE_WAIT_9;
442
            when S_ISA_RE_WAIT_9 =>
444             nPC_SBHE <= '0';
                PC_AEN <= '0';
446             nISA_LESEN <= '0';
                nPC_SMEM_R <= '0';
448             ISA_STATE <= S_ISA_RE_WAIT_10;

450         when S_ISA_RE_WAIT_10 =>
                nPC_SBHE <= '0';
452             PC_AEN <= '0';
                nISA_LESEN <= '0';
454             nPC_SMEM_R <= '0';
                ISA_STATE <= S_ISA_RE_WAIT_11;
456
            when S_ISA_RE_WAIT_11 =>
458             nPC_SBHE <= '0';
                PC_AEN <= '0';
460             nISA_LESEN <= '0';
                nPC_SMEM_R <= '0';
462             ISA_STATE <= S_ISA_RE_WAIT_CHRDY;

464         -- warten bis Slave sich mit IO_CHRDY = 1 fertig meldet
            when S_ISA_RE_WAIT_CHRDY =>
466             nPC_SBHE <= '0';
                PC_AEN <= '0';
468             nISA_LESEN <= '0';
                nPC_SMEM_R <= '0';
470             if (IO_CHRDY_INT = '1') then
                    ISA_STATE <= S_ISA_RE_READ;
472                 ISA_DATA_IN_BUFFER <= PC_D;
                else
474                 ISA_STATE <= S_ISA_RE_WAIT_CHRDY;
                end if;
476
            -- von Datenleitungen lesen
478         when S_ISA_RE_READ =>
                nPC_SBHE <= '0';
480             PC_AEN <= '0';
                nISA_LESEN <= '0';
482             ISA_STATE <= S_ISA_RE_CLR_SMEMR;
```

```
484                 -- lesen beendet
                    when S_ISA_RE_CLR_SMEMR =>
486                     nPC_SBHE <= '0';
                        PC_AEN <= '0';
488                     nISA_LESEN <= '0';
                        ISA_STATE <= S_ISA_RE_WAIT_12;
490
                    -- 3 Takte warten
492                 when S_ISA_RE_WAIT_12 =>
                        nPC_SBHE <= '0';
494                     PC_AEN <= '0';
                        nISA_LESEN <= '0';
496                     ISA_STATE <= S_ISA_RE_WAIT_13;

498                 when S_ISA_RE_WAIT_13 =>
                        nPC_SBHE <= '0';
500                     PC_AEN <= '0';
                        nISA_LESEN <= '0';
502                     ISA_STATE <= S_ISA_RE_WAIT_14;

504                 when S_ISA_RE_WAIT_14 =>
                        nPC_SBHE <= '0';
506                     PC_AEN <= '0';
                        nISA_LESEN <= '0';
508                     PC_A <= (others => '0');
                        PC_LA <= (others => '0');
510                     ISA_STATE <= S_ISA_WAIT_CYCLE_0;

512                 ------------------------------------
                    -- Schreibzugriff Highbyte (8bit) --
514                 ------------------------------------

516                 -- Adresse und Daten auf den Bus legen
                    when S_ISA_WR_H_SET_ADR_DATA =>
518                     PC_AEN <= '0';
                        ISA_OE <= '1';
520                     nISA_SCHREIBEN <= '0';
                        ISA_STATE <= S_ISA_WR_H_WAIT_0;
522
                    -- 9 Takte warten (min 183ns)
524                 when S_ISA_WR_H_WAIT_0 =>
                        PC_AEN <= '0';
526                     ISA_OE <= '1';
```

```
                nISA_SCHREIBEN <= '0';
528             ISA_STATE <= S_ISA_WR_H_WAIT_1;

530         when S_ISA_WR_H_WAIT_1 =>
                PC_AEN <= '0';
532             ISA_OE <= '1';
                nISA_SCHREIBEN <= '0';
534             ISA_STATE <= S_ISA_WR_H_WAIT_2;

536         when S_ISA_WR_H_WAIT_2 =>
                PC_AEN <= '0';
538             ISA_OE <= '1';
                nISA_SCHREIBEN <= '0';
540             ISA_STATE <= S_ISA_WR_H_WAIT_3;

542         when S_ISA_WR_H_WAIT_3 =>
                PC_AEN <= '0';
544             ISA_OE <= '1';
                nISA_SCHREIBEN <= '0';
546             ISA_STATE <= S_ISA_WR_H_WAIT_4;

548         when S_ISA_WR_H_WAIT_4 =>
                PC_AEN <= '0';
550             ISA_OE <= '1';
                nISA_SCHREIBEN <= '0';
552             ISA_STATE <= S_ISA_WR_H_WAIT_5;

554         when S_ISA_WR_H_WAIT_5 =>
                PC_AEN <= '0';
556             ISA_OE <= '1';
                nISA_SCHREIBEN <= '0';
558             ISA_STATE <= S_ISA_WR_H_WAIT_6;

560         when S_ISA_WR_H_WAIT_6 =>
                PC_AEN <= '0';
562             ISA_OE <= '1';
                nISA_SCHREIBEN <= '0';
564             ISA_STATE <= S_ISA_WR_H_WAIT_7;

566         when S_ISA_WR_H_WAIT_7 =>
                PC_AEN <= '0';
568             ISA_OE <= '1';
                nISA_SCHREIBEN <= '0';
570             ISA_STATE <= S_ISA_WR_H_WAIT_8;
```

```
572                 when S_ISA_WR_H_WAIT_8 =>
                        PC_AEN <= '0';
574                     ISA_OE <= '1';
                        nISA_SCHREIBEN <= '0';
576                     nPC_SMEM_W <= '0'; -- Daten schrieben
                        ISA_STATE <= S_ISA_WR_H_SET_SMEMW;
578
                    -- Daten schreiben
580                 when S_ISA_WR_H_SET_SMEMW =>
                        PC_AEN <= '0';
582                     ISA_OE <= '1';
                        nISA_SCHREIBEN <= '0';
584                     nPC_SMEM_W <= '0';
                        ISA_STATE <= S_ISA_WR_H_WAIT_9;
586
                    -- 4 Takte warten (min 70ns)
588                 when S_ISA_WR_H_WAIT_9 =>
                        PC_AEN <= '0';
590                     ISA_OE <= '1';
                        nISA_SCHREIBEN <= '0';
592                     nPC_SMEM_W <= '0';
                        ISA_STATE <= S_ISA_WR_H_WAIT_10;
594
                    when S_ISA_WR_H_WAIT_10 =>
596                     PC_AEN <= '0';
                        ISA_OE <= '1';
598                     nISA_SCHREIBEN <= '0';
                        nPC_SMEM_W <= '0';
600                     ISA_STATE <= S_ISA_WR_H_WAIT_11;

602                 when S_ISA_WR_H_WAIT_11 =>
                        PC_AEN <= '0';
604                     ISA_OE <= '1';
                        nISA_SCHREIBEN <= '0';
606                     nPC_SMEM_W <= '0';
                        ISA_STATE <= S_ISA_WR_H_WAIT_CHRDY;
608

610                 -- warten bis Slave sich mit IO_CHRDY = 1 fertig meldet
                    when S_ISA_WR_H_WAIT_CHRDY =>
612                     if (IO_CHRDY_INT = '1') then
                            -- Schreibanforderung loeschen
614                         PC_AEN <= '0';
```

```
                ISA_OE <= '1';
616             nISA_SCHREIBEN <= '0';
                ISA_STATE <= S_ISA_WR_H_WAIT_12;
618         else
                PC_AEN <= '0';
620             ISA_OE <= '1';
                nISA_SCHREIBEN <= '0';
622             nPC_SMEM_W <= '0';
                ISA_STATE <= S_ISA_WR_H_WAIT_CHRDY;
624         end if;

626

        when S_ISA_WR_H_WAIT_12 =>
628         PC_AEN <= '0';
            ISA_OE <= '1';
630         nISA_SCHREIBEN <= '0';
            ISA_STATE <= S_ISA_WR_H_WAIT_13;
632

634     when S_ISA_WR_H_WAIT_13 =>
            PC_AEN <= '0';
636         ISA_OE <= '1';
            nISA_SCHREIBEN <= '0';
638         ISA_STATE <= S_ISA_WR_H_WAIT_14;

640

        when S_ISA_WR_H_WAIT_14 =>
642         -- Signale in den IDLE Zustand setzen
            PC_A <= (others => '0');
644         PC_LA <= (others => '0');
            ISA_STATE <= S_ISA_WR_H_WAIT_15;
646

        -- 3 Takte warten
648     when S_ISA_WR_H_WAIT_15 =>
            ISA_STATE <= S_ISA_WR_H_WAIT_16;
650

        when S_ISA_WR_H_WAIT_16 =>
652         ISA_STATE <= S_ISA_WR_H_WAIT_17;

654     when S_ISA_WR_H_WAIT_17 =>
            ISA_STATE <= S_ISA_WAIT_CYCLE_0;
656

658
```

```
660                 -- ------------------------------------
                    -- Schreibzugriff Highbyte/16-Bit --
                    -- ------------------------------------
662
                    -- Adresse und Daten auf den Bus legen
664                 when S_ISA_WR_LH_L_SET_ADR_DATA =>
                        ISA_OE <= '1';
666                     nPC_SBHE <= '0';
                        PC_AEN <= '0';
668                     nISA_SCHREIBEN <= '0';
                        ISA_STATE <= S_ISA_WR_LH_L_WAIT_0;
670
672                 -- 4 Takte warten (min 66ns)
                    when S_ISA_WR_LH_L_WAIT_0 =>
674                     ISA_OE <= '1';
                        nPC_SBHE <= '0';
676                     PC_AEN <= '0';
                        nISA_SCHREIBEN <= '0';
678                     ISA_STATE <= S_ISA_WR_LH_L_WAIT_1;
680
                    when S_ISA_WR_LH_L_WAIT_1 =>
682                     ISA_OE <= '1';
                        nPC_SBHE <= '0';
684                     PC_AEN <= '0';
                        nISA_SCHREIBEN <= '0';
686                     ISA_STATE <= S_ISA_WR_LH_L_WAIT_2;
688
                    when S_ISA_WR_LH_L_WAIT_2 =>
690                     ISA_OE <= '1';
                        nPC_SBHE <= '0';
692                     PC_AEN <= '0';
                        nISA_SCHREIBEN <= '0';
694                     ISA_STATE <= S_ISA_WR_LH_L_WAIT_3;
696
                    when S_ISA_WR_LH_L_WAIT_3 =>
698                     ISA_OE <= '1';
                        nPC_SBHE <= '0';
700                     PC_AEN <= '0';
                        nISA_SCHREIBEN <= '0';
702                     ISA_STATE <= S_ISA_WR_LH_L_CHECK_M16;
```

```
704                 -- Pruefen ob der Slave zur 16-Bit-Uebertragung
                    --     faehig ist
706                 when S_ISA_WR_LH_L_CHECK_M16 =>
                        ISA_OE <= '1';
708                     nPC_SBHE <= '0';
                        PC_AEN <= '0';
710                     nISA_SCHREIBEN <= '0';
                        if (nPC_MEM16_INT = '0') then
712                         ISA_STATE <= S_ISA_WR_LH_L_WAIT_4;
                            nISA_SCHREIBEN <= '0';
714                     else
                            -- fals die 16 Kommunikation nicht moeglich ist,
716                         -- kehrt die State Machine in den Idle
                            -- Status zurueck, ohne Daten gelesen zu haben.
718                         -- Da die 16-bit-Kommunikation mit der CNC auf
                            -- jedenfall moeglich ist,
720                         -- ist diese Abfrage nur fuer die
                            -- Vollstaendigkeit vorhanden.
722                         -- Falls dieser Fall dennoch auftreten sollte,
                                bemerkt der PC diesen Fehler
                            -- (da keine Antwort kommt)
724                         -- und fuert einen Reset durch.
                            PC_A <= (others => '0');
726                         PC_LA <= (others => '0');
                            ISA_STATE <= S_ISA_IDLE;
728                     end if;

730                 -- 2 Takte warten (min 54ns)
                    when S_ISA_WR_LH_L_WAIT_4 =>
732                     ISA_OE <= '1';
                        nPC_SBHE <= '0';
734                     PC_AEN <= '0';
                        nISA_SCHREIBEN <= '0';
736                     ISA_STATE <= S_ISA_WR_LH_L_WAIT_5;

738                 when S_ISA_WR_LH_L_WAIT_5 =>
                        ISA_OE <= '1';
740                     nPC_SBHE <= '0';
                        PC_AEN <= '0';
742                     nISA_SCHREIBEN <= '0';
                        ISA_STATE <= S_ISA_WR_LH_L_SET_SMEMW;
744
                    -- Daten schreiben
```

```
746                 when S_ISA_WR_LH_L_SET_SMEMW =>
                        ISA_OE <= '1';
748                     nPC_SBHE <= '0';
                        PC_AEN <= '0';
750                     nISA_SCHREIBEN <= '0';
                        nPC_SMEM_W <= '0';
752                     ISA_STATE <= S_ISA_WR_LH_L_WAIT_6;

754                 -- 4 Takte warten (min 70ns)
                    when S_ISA_WR_LH_L_WAIT_6 =>
756                     ISA_OE <= '1';
                        nPC_SBHE <= '0';
758                     PC_AEN <= '0';
                        nISA_SCHREIBEN <= '0';
760                     nPC_SMEM_W <= '0';
                        ISA_STATE <= S_ISA_WR_LH_L_WAIT_7;
762
                    when S_ISA_WR_LH_L_WAIT_7 =>
764                     ISA_OE <= '1';
                        nPC_SBHE <= '0';
766                     PC_AEN <= '0';
                        nISA_SCHREIBEN <= '0';
768                     nPC_SMEM_W <= '0';
                        ISA_STATE <= S_ISA_WR_LH_L_WAIT_8;
770
                    when S_ISA_WR_LH_L_WAIT_8 =>
772                     ISA_OE <= '1';
                        nPC_SBHE <= '0';
774                     PC_AEN <= '0';
                        nISA_SCHREIBEN <= '0';
776                     nPC_SMEM_W <= '0';
                        ISA_STATE <= S_ISA_WR_LH_L_WAIT_9;
778
                    when S_ISA_WR_LH_L_WAIT_9 =>
780                     ISA_OE <= '1';
                        nPC_SBHE <= '0';
782                     PC_AEN <= '0';
                        nISA_SCHREIBEN <= '0';
784                     nPC_SMEM_W <= '0';
                        ISA_STATE <= S_ISA_WR_LH_L_WAIT_CHRDY;
786
                    -- warten bis Slave sich mit IO_CHRDY = 1 fertig meldet
788                 when S_ISA_WR_LH_L_WAIT_CHRDY =>
                        ISA_OE <= '1';
```

```
790                 nPC_SBHE <= '0';
                    PC_AEN <= '0';
792                 nISA_SCHREIBEN <= '0';
                    if (IO_CHRDY_INT = '1') then
794                     -- Schreibanforderung loeschen
                        nPC_SMEM_W <= '1';
796                     ISA_STATE <= S_ISA_WR_LH_L_WAIT_10;
                    else
798                     nPC_SMEM_W <= '0';
                        ISA_STATE <= S_ISA_WR_LH_L_WAIT_CHRDY;
800                 end if;

802             -- 3 Takte warten (min 53ns)
                when S_ISA_WR_LH_L_WAIT_10 =>
804                 ISA_OE <= '1';
                    nPC_SBHE <= '0';
806                 PC_AEN <= '0';
                    nISA_SCHREIBEN <= '0';
808                 ISA_STATE <= S_ISA_WR_LH_L_WAIT_11;

810

                when S_ISA_WR_LH_L_WAIT_11 =>
812                 ISA_OE <= '1';
                    nPC_SBHE <= '0';
814                 PC_AEN <= '0';
                    nISA_SCHREIBEN <= '0';
816                 ISA_STATE <= S_ISA_WR_LH_L_WAIT_12;

818

                when S_ISA_WR_LH_L_WAIT_12 =>
820                 ISA_OE <= '1';
                    nPC_SBHE <= '0';
822                 PC_AEN <= '0';
                    nISA_SCHREIBEN <= '0';
824                 ISA_STATE <= S_ISA_WR_LH_L_WAIT_13;

826             when S_ISA_WR_LH_L_WAIT_13 =>
                    -- Signale in den IDLE Zustand setzen
828                 PC_A <= (others => '0');
                    PC_LA <= (others => '0');
830                 ISA_STATE <= S_ISA_WAIT_CYCLE_0;

832

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

```
834                     -- Zwischen den Uebertragungen warten --
                        ------------------------------------------
836
                        -- 3 Takte warten (min 108ns)
838                     when S_ISA_WAIT_CYCLE_0 =>
                            ISA_STATE <= S_ISA_WAIT_CYCLE_1;
840
                        when S_ISA_WAIT_CYCLE_1 =>
842                         ISA_STATE <= S_ISA_IDLE;
                    end case;
844             end if;

846         end process ISA_COMMUNICATION;

848 end BEHAVIORAL;
```

C. Quelltext der Testbench für den ISA-Zustandsautomaten

Quelltext C.1: Testbench für den ISA-Zustandsautomaten

```
--============================================================
-- TestBench_usb_isa.vhd
--       Funktionstest der ISA Logik (ISA.vhd)
--
-- Author    : Daniel Sabotta
--
-- Date      : 29.09.2010
--
-- Version   : 1.0
--
--============================================================

library ieee;
use ieee.STD_LOGIC_UNSIGNED.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.Numeric_Std.all;
library std;
use std.textio.all;



entity ISA_TESTBENCH is
    generic(
        -- periode des Taktsignals
        PERIODE_48MHZ    :    time := 20.833ns;
        -- Adresse die an LA anliegen muss um die CNC anzusprechen
        PC_LA_COMP       :    std_logic_vector (6 downto 0) := "0000110";
        -- die ersten 5 Bits der CNC-Adresse
        PC_A_COMP        :    std_logic_vector (19 downto 15) := "11010";
        -- Groesse der einzelnen RAM-Bausteine
        RAM_SIZE   : integer := 32768;
        -- Wortbreite der des RAMs
        RAM_DATA_LENGTH :    integer := 8;
        -- Datei in der der Inhalt des RAMs geschrieben wird
        OUTPUT_FILE      :    string := "RAM-Simmulation.txt";
    );
end ISA_TESTBENCH;

architecture TB_ARCHITECTURE of ISA_TESTBENCH is
    -- Komponentendeklaration der ISA-Logik
    component ISA_COMMUNICATION
        port(
```

```
44          -- Reset (low aktiv)
            nRESET          : in    std_logic;
46          -- 48MHz Takt
            CLK_48MHZ       : in    std_logic;
48
            ------------------------
50          -- Pins zum ISA-Bus --
            ------------------------
52          -- Dateinleitungen
            -- System Data bits
54          PC_D            : inout std_logic_vector (15 downto 0);

56          -- Adressleitungen (Bits 17, 18 und 19 kommen in beden vor!)
            -- Adressbits (niederwertige Adressbits 0 bis 19)
58          PC_A            : out std_logic_vector (19 downto 0);
            -- Latched Adressbits (hoeherwertige Adressbits 17 bis 23)
60
            -- Steuerleitungen
62          -- Channel Ready
            IO_CHRDY        : in std_logic;
64          -- Memory Chip Select 16
            nPC_MEM16       : in std_logic;
66          -- Address Enable
            PC_AEN          : out std_logic;
68          -- System Byte High Enable
            nPC_SBHE        : out std_logic;
70          -- System Memory Read
            nPC_SMEM_R      : out std_logic;
72          -- System Memory Write
            nPC_SMEM_W      : out std_logic;
74
            -- Signale fuer Bustreiber
76          -- schreibe auf Datenleitungen
            nISA_SCHREIBEN  : out std_logic;
78          -- Lese von Dateinleitungen
            nISA_LESEN      : out std_logic;
80
            ------------------------------------
82          -- Signale zum COMMAND-HANDLER --
            ------------------------------------
84          -- Steuersignale der ISA_COMMUNICATION
            -- High- und Lowbyte lesen
86          ISA_RE          : in std_logic;
            -- High- und Lowbyte schreiben
```

```
88          ISA_WE_HL        : in std_logic;
            -- Highbyte schreiben
90          ISA_WE_H         : in std_logic;
            -- Lowbyte schreiben
92          ISA_WE_L         : in std_logic;

94          -- Daten, die ueber den Bus geschickt werden sollen
            ISA_DATA_OUT_BUFFER : in std_logic_vector (15 downto 0);
96          -- Eingehende Daten
            ISA_DATA_IN_BUFFER  : out std_logic_vector (15 downto 0);
98          -- Adresse
            ISA_ADDRESS         : in std_logic_vector (12 downto 0);
100
            -- ISA-Handler ist beschaeftigt
102         ISA_BUSY         : out std_logic
            );
104     end component ISA_COMMUNICATION;

106     ------------------------
        -- Stimulus Signale --
108     ------------------------
        signal nRESET        : std_logic := '0';
110     signal CLK_48MHZ     : std_logic := '0';
        signal IO_CHRDY      : std_logic := '0';
112     signal PC_MEM16      : std_logic := '0';
        signal ISA_RE        : std_logic := '0';
114     signal ISA_WE_L      : std_logic := '0';
        signal ISA_WE_H      : std_logic := '0';
116     signal ISA_WE_HL     : std_logic := '0';

118     signal PC_D          : std_logic_vector (15 downto 0) := (others => 'Z
            ');
        -- eingehende Adresse
120     signal ISA_ADDRESS   : std_logic_vector (12 downto 0) := (others => 'Z
            ');

122     -- eingehende Daten
        signal ISA_DATA_IN_BUFFER    : std_logic_vector (15 downto 0) := (
            others => 'Z');
124
        ------------------------
126     -- Ausgangssignale --
        ------------------------
128     signal PC_AEN            : std_logic;
```

```
    signal nPC_SBHE          : std_logic;
130 signal nPC_SMEM_R        : std_logic;
    signal nPC_SMEM_W        : std_logic;
132 signal nPC_MEM16         : std_logic;
    signal nISA_SCHREIBEN    : std_logic;
134 signal nISA_LESEN        : std_logic;
    signal ISA_BUSY          : std_logic;
136
    signal PC_A              : std_logic_vector (19 downto 0);
138 signal PC_LA             : std_logic_vector ( 6 downto 0);

140 signal ISA_DATA_OUT_BUFFER : std_logic_vector (15 downto 0);

142 ------------------------------------
    -- Signale des simulierten RAMS --
144 ------------------------------------
    type RAM_TYPE is array (RAM_SIZE downto 0) of std_logic_vector(
        RAM_DATA_LENGTH - 1 downto 0);
146
    -- RAM fuer das Lowbyte
148 signal RAM_LOW  : RAM_TYPE := (others => (others => '0'));
    -- RAM fuer das Highbyte
150 signal RAM_HIGH : RAM_TYPE := (others => (others => '0'));

152 -- Chip Selcet fuer beide RAMs
    signal nRAM_CS  : std_logic := '1';
154 -- Output Enable fuer beide RAMs
    signal nRAM_OE  : std_logic := '1';
156 -- Write Enable fuer das Lowbyte RAM
    signal nRAM_WL  : std_logic := '1';
158 -- Write Enable fuer das Highbyte RAM
    signal nRAM_WH  : std_logic := '1';
160
    -- 1 wenn die richtige Adresse anliegt
162 signal ADRESS_OK: std_logic := '0';

164 -- Datenleitungen
    signal PC_D_INT : std_logic_vector (15 downto 0);
166 signal PC_D_OE  : std_logic;

168 ------------------------------------------------------------
    -- Funktion: std_logic_vector in bit_vector umwandeln --
170 ------------------------------------------------------------
```

```
    function to_bit_vector (IN_STD_LOGIC_VECTOR : std_logic_vector)
        return bit_vector is
172     variable OUTPUT : bit_vector (IN_STD_LOGIC_VECTOR'length-1 downto
            0);
    begin
174     ITERATE: for I in 0 to IN_STD_LOGIC_VECTOR'length-1 loop
            if (IN_STD_LOGIC_VECTOR(I) = '1') then
176             OUTPUT(I) := '1';
            else
178             OUTPUT(I) := '0';
            end if;
180     end loop ITERATE;
        return OUTPUT;
182 end function;

184 ------------------------------------------------------------
    -- Funktion: schreibt die Werte des RAMs in eine Datei --
186 ------------------------------------------------------------
    function WRITE_RAM_INTO_FILE (FILENAME: string; RAM_L: RAM_TYPE;
        RAM_H: RAM_TYPE) return bit is
188     file OUT_FILE : text open write_mode is FILENAME;
        variable WR_LINE : line;
190 begin
        for ADR in 0 to RAM_SIZE loop
192         -- Highbyte (gerade Adressen)
            write(WR_LINE, integer'image(ADR * 2));
194         write(WR_LINE, "H: ");
            write(WR_LINE, to_bit_vector(RAM_H(ADR)));
196         writeline(OUT_FILE, WR_LINE);

198         -- Lowbyte (ungerade Adressen)
            write(WR_LINE, integer'image(ADR * 2 + 1));
200         write(WR_LINE, "L: ");
            write(WR_LINE, to_bit_vector(RAM_L(ADR)));
202         writeline(OUT_FILE, WR_LINE);
        end loop;
204     return '0';
    end function WRITE_RAM_INTO_FILE;
206

208

210
```

```
212

214 begin
    ------------------------------------------------------------------
216 -- Port Map der zu testenden Logik (UUT - unit under test) --
    ------------------------------------------------------------------
218 UUT : ISA_COMMUNICATION
    port map(
220     nRESET              => nRESET,
        CLK_48MHZ           => CLK_48MHZ,
222     PC_D                => PC_D,
        PC_A                => PC_A,
224     PC_LA               => PC_LA,
        IO_CHRDY            => IO_CHRDY,
226     nPC_MEM16           => nPC_MEM16,
        PC_AEN              => PC_AEN,
228     nPC_SBHE            => nPC_SBHE,
        nPC_SMEM_R          => nPC_SMEM_R,
230     nPC_SMEM_W          => nPC_SMEM_W,
        nISA_SCHREIBEN      => nISA_SCHREIBEN,
232     nISA_LESEN          => nISA_LESEN,
        ISA_RE              => ISA_RE,
234     ISA_WE_HL           => ISA_WE_HL,
        ISA_WE_H            => ISA_WE_H,
236     ISA_WE_L            => ISA_WE_L,
        ISA_BUSY            => ISA_BUSY,
238     ISA_DATA_OUT_BUFFER => ISA_DATA_OUT_BUFFER,
        ISA_DATA_IN_BUFFER  => ISA_DATA_IN_BUFFER,
240     ISA_ADDRESS         => ISA_ADDRESS
        );
242
    -- Reset-Signal
244 nRST_SIM: process
    begin
246     nRESET <= '0';
        wait for PERIODE_48MHZ;
248     nRESET <= '1';
        wait;
250 end process;

252
    -- Clock-Signal
254 CLK_SIM: process
    begin
```

```
256         CLK_48MHZ <= '0';
            wait for PERIODE_48MHZ / 2;
258         CLK_48MHZ <= '1';
            wait for PERIODE_48MHZ / 2;
260     end process CLK_SIM;


262

264     STIMULI: process
        variable RVAL : bit;
266     variable WRITE_LINE : line;
        file IN_FILE : text open read_mode is INPUT_FILE;
268         variable RE_LINE : line;
            variable readbuffer : string(RAM_DATA_LENGTH downto 1);
270         variable ramBuffer : std_logic_vector(7 downto 0);
            variable ramCount : integer := 0;
272     begin
            -- Low und highbyte schreiben
274         wait for 10ns;
            ISA_ADDRESS <= "0000000000000";
276         ISA_DATA_OUT_BUFFER <= x"0102";
            ISA_WE_HL <= '1';
278
            wait for 100ns;
280         ISA_WE_HL <= '0';
            wait until ISA_BUSY = '0';
282
            -- Highbyte schreiben
284         wait for 10ns;
            ISA_ADDRESS <= "0000000000001";
286         ISA_DATA_OUT_BUFFER <= x"FF88";
            ISA_WE_H <= '1';
288
            wait for 100ns;
290         ISA_WE_H <= '0';
            wait until ISA_BUSY = '0';
292
            -- Lowbyte schreiben
294         wait for 10ns;
            ISA_ADDRESS <= "0000000000001";
296         ISA_DATA_OUT_BUFFER <= x"FF88";
            ISA_WE_L <= '1';
298
            wait for 100ns;
```

```
300         ISA_WE_L <= '0';
            wait until ISA_BUSY = '0';
302
            -- Lesen
304         wait for 10ns;
            ISA_RE <= '1';
306         wait for 100ns;
            ISA_RE <= '0';
308         wait until ISA_BUSY = '0';

310         RVAL := WRITE_RAM_INTO_FILE(OUTPUT_FILE, RAM_LOW, RAM_HIGH);

312         write(WRITE_LINE, "-------------------------");
            writeline(OUTPUT, WRITE_LINE);
314         write(WRITE_LINE, "Ram in ");
            write(WRITE_LINE, OUTPUT_FILE);
316         write(WRITE_LINE, " geschrieben!");
            writeline(OUTPUT, WRITE_LINE);
318         write(WRITE_LINE, "-------------------------");
            writeline(OUTPUT, WRITE_LINE);
320         wait;
    end process;
322
    ------------------------------
324 -- CNC-ISA-RAM-Simulation --
    ------------------------------
326
    -- Adresse OK?
328 ADRESS_OK <= '1' when (PC_A(19 downto 15) = PC_A_COMP(19 downto 15)
        and (PC_LA = PC_LA_COMP)) else '0';

330 -- CSDPRAM
    nRAM_CS <= '0' when ((ADRESS_OK = '1') and ((nPC_SMEM_W = '0') or (
        nPC_SMEM_R = '0')) and (PC_AEN = '0')) else '1';
332

334 -- OEDPRAM
    nRAM_OE <= '0' when ((ADRESS_OK = '1') and (nPC_SMEM_R = '0') and (
        PC_AEN= '0')) else '1';
336

338 -- WLDPRAM
    nRAM_WL <= '0' when ((ADRESS_OK = '1') and (nPC_SMEM_W = '0') and  (
        nPC_SBHE = '0') and (PC_AEN = '0')) else '1';
```

```
340

342     -- WHDPRAM
        nRAM_WH <= '0' when ((ADRESS_OK = '1') and (nPC_SMEM_W = '0') and (
            PC_A(0) = '0') and (PC_AEN = '0')) else '1';
344

346     -- ist auch auf der CNC logische verknuepft
        IO_CHRDY <= '1' when ( (ADRESS_OK = '1') and (PC_AEN = '0') and (
            nPC_SMEM_R = '0' or nPC_SMEM_W = '0') ) else 'Z' after 1ns;
348     nPC_MEM16 <= (PC_A(15))  when ( (ADRESS_OK = '1') and (PC_AEN = '0')
            ) else 'Z' after 1ns;

350     -- Ausgangstreiber
        PC_D <= PC_D_INT when (nPC_SMEM_R = '0') else (others => 'Z');
352
        RAM_SIMULATION: process (nRAM_CS)
354     variable RAM_ADR: integer range 0 to RAM_DATA_LENGTH;
        begin
356         if (nRAM_CS = '0') then
                RAM_ADR := conv_integer(PC_A(15 downto 1));
358
                -- falls nur ein Byte gelesen wird,
360             -- ist das andere gleich null
                if (nRAM_OE = '0') then
362                 PC_D_INT <= (others => '0');
                end if;
364
                -- LOWbyte
366             if (nRAM_WL = '0') then
                    -- in RAM Speichern
368                 RAM_LOW(RAM_ADR) <= PC_D(7 downto 0);
                elsif (nRAM_OE = '0') then
370                 -- aus RAM lesen
                    PC_D_INT(7 downto 0) <= RAM_LOW(RAM_ADR);
372             end if;

374             -- HIGHbyte
                if (nRAM_WL = '0') then
376                 -- in RAM Speichern
                    RAM_HIGH(RAM_ADR) <= PC_D(15 downto 8);
378             elsif (nRAM_OE = '0') then
                    -- aus RAM lesen
380                 PC_D_INT(15 downto 8) <= RAM_HIGH(RAM_ADR);
```

```
                end if;
382           end if;
      end process RAM_SIMULATION;
384 end TB_ARCHITECTURE;
```

D. Spezifikationen der ISA-Zeiten

ISA Bus Timing Diagrams

P/N 5001321 Revision A

COMPUTERS, INCORPORATED

4757 Hellyer Avenue, San Jose, CA 95138
Phone: 408 360-0200, FAX: 408 360-0222, Web: www.ampro.com

REVISION HISTORY

Revision	Reason for Change	Date
A	Initial Release	6/98

ISA Bus Timing Diagrams

Ampro's ISA bus timing diagrams are derived from diagrams in the IEEE P996 draft specification which were, in turn, derived from the timing of the original IBM AT computer.

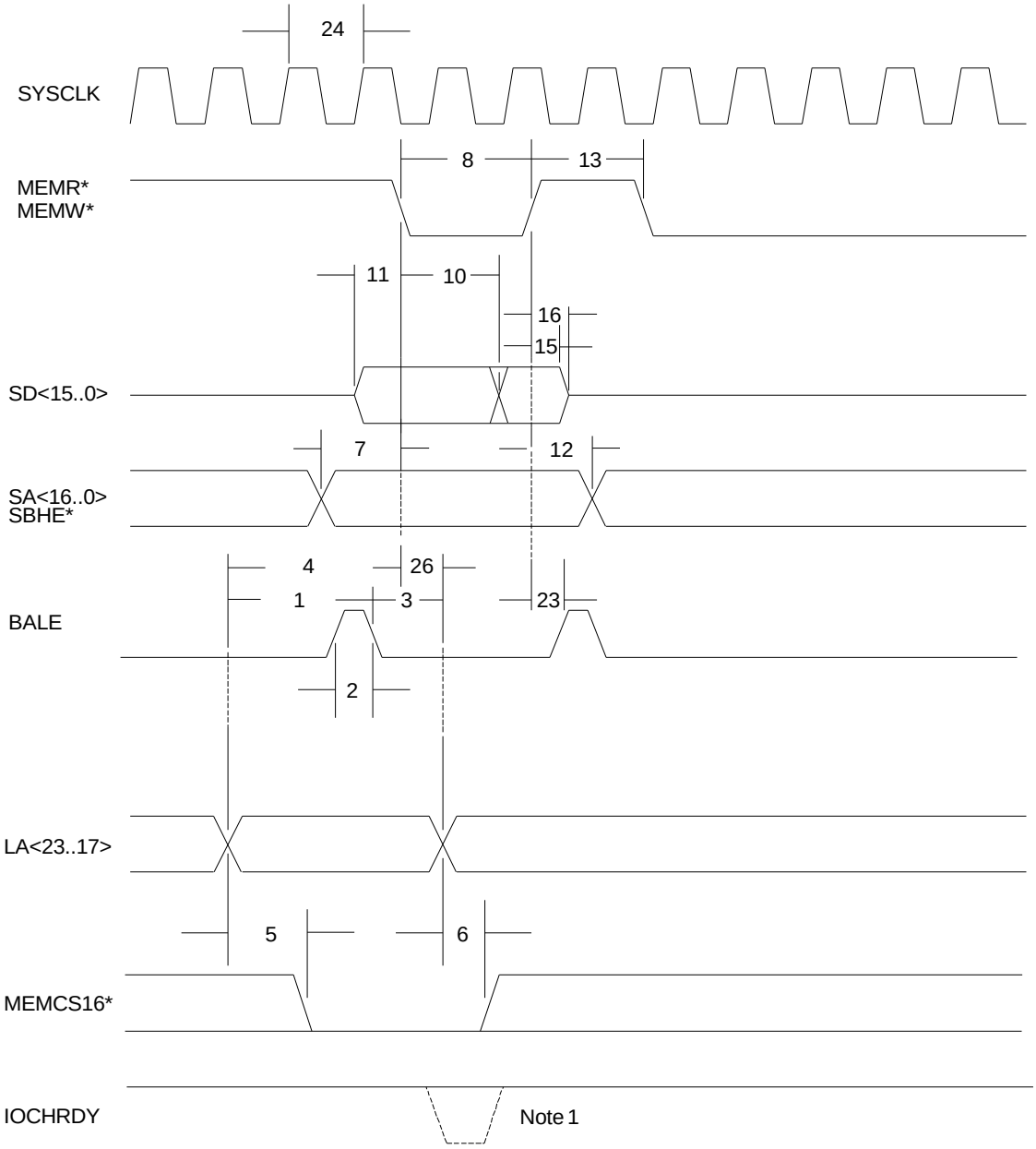
Please note that the IEEE P996 draft specification was never completed by the IEEE and is not an IEEE approved spec. Also, the "latest" IEEE draft is known to contain errors. In the absence of an approved IEEE specification, manufacturers of PC chip sets attempt to meet a "consensus" ISA bus standard. This has resulted in minor variations in signal interpretation and timing among the various PC chipset vendors. For this reason, Ampro recommends that designers of interfaces to the ISA bus use the minimum number of bus signals needed to perform a required function (e.g. chip selection or signal synchronization). For example, at least one popular chipset does not drive AEN high during REFRESH. In certain instances, Ampro has added logic to improve bus timing and/or signal relationships on CPU and peripheral boards.

Ampro's ISA bus timing diagrams include several corrections relative to the IEEE P996 draft specification. However, since these diagrams are derived from an uncompleted and unapproved IEEE specification, they may contain other errors.

For comprehensive technical details on the ISA architecture and bus, Ampro recommends the following book: *ISA & EISA Theory and Operation*, by Edward Solari; published by Annabooks (www.annabooks.com). This book contains a detailed technical exposition of the ISA and EISA buses and is written by the principal author of the IEEE P996 draft specification.

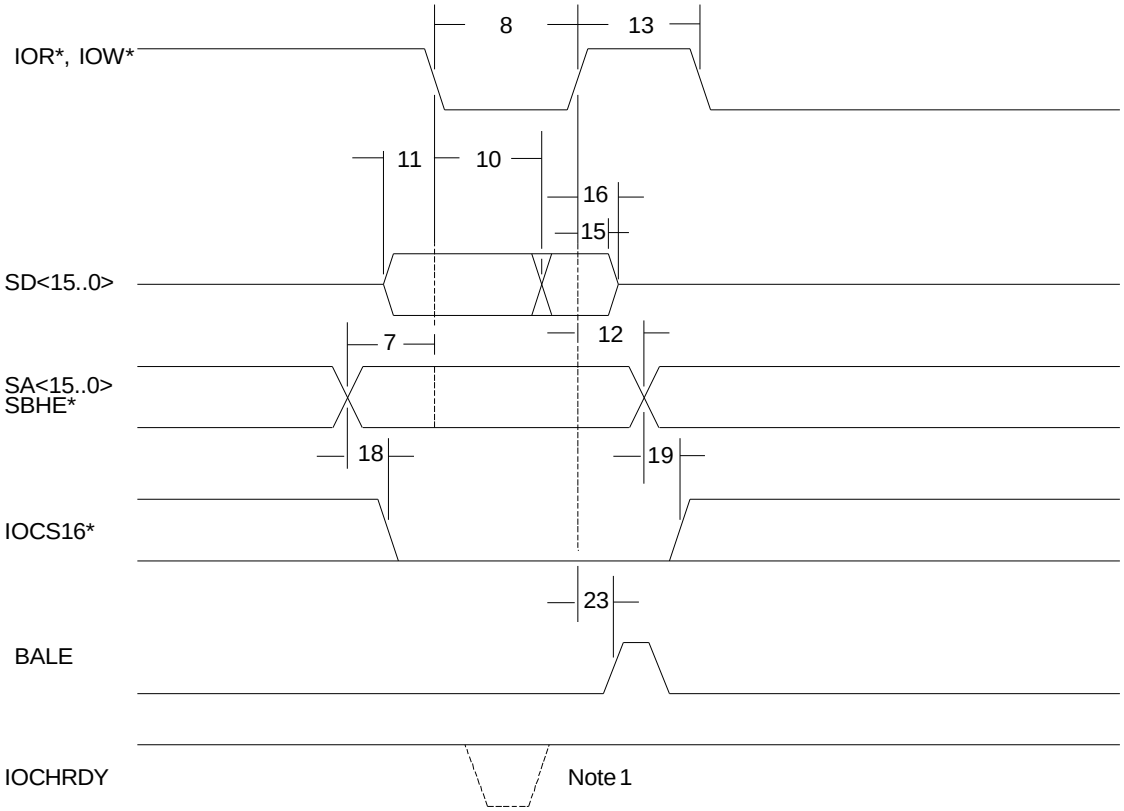
REF	TYPE	SIZE	DESCRIPTION	DRIVER		RECEIVER	
				MIN	MAX	MIN	MAX
1	M,IO	8/16	LA setup to BALE deasserted	111		100	
2	M,IO	8/16	BALE pulse width	61		50	
3	M,IO	8/16	LA hold from BALE deasserted	26		15	
4a	M	16	LA setup to MEMx* asserted	120		109	
4b	M	8	LA setup to MEMx* asserted	183		172	
5	M	8/16	MEMCS16* valid from LA		66		102
6	M	8/16	MEMCS16* hold from LA	0		0	
7a	M	16	SA, SBHE* setup to MEMx*	39		28	
7b	IO	16	SA, SBHE* setup to IOx*	102		91	
7c	M,IO	8	SA, SBHE* setup to IOx* or MEMx*	102		91	
8a	M	16	Command width	240		219	
8b	IO	16	Command width	165		154	
8c	M	16	Command width with ENDXFR* asserted	103		92	
8d	M,IO	8	Command width	541		530	
10a	M	16	Read data access		173		195
10b	IO	16	Read data access		110		132
10c	M,IO	16	Read data access with ENDXFR* asserted		48		70
10d	M,IO	8	Read data access		482		504
11a	M	16	Write data setup	-34		-45	
11b	IO	16	Write data setup	33		22	
11c	M,IO	8	Write data setup (even)	7		-4	
11d	M,IO	8	Write data setup (odd)	-45		-56	
12	M,IO	8/16	SA, SBHE* hold	53		42	
13a	M	16	Command deasserted	108		97	
13b	M	8	Command deasserted	170		159	
13c	IO	8/16	Command deasserted	170		159	
15a	M,IO	8/16	Read data hold	0		0	
15b	M,IO	8/16	Write data hold	25		25	
16	M,IO	8/16	Read command to SD disabled		30		30
17	M	16	ENDXFR* asserted from command		10		32
18	IO	8/16	IOCS16* asserted from SA		74		122
19	IO	8/16	IOCS16* hold from SA	0		0	
20a	M,IO	8/16	IOCHRDY valid from command asserted		70		159
20b	M,IO	8	IOCHRDY valid from command asserted		373		462
21	M,IO	8/16	IOCHRDY deasserted pulse width	125	15600	125	15611
22	M,IO	8/16	Command hold from IOCHRDY	125			
23	M,IO	8/16	BALE asserted from command deasserted	46		35	
24	M,IO	8/16	Clock period (Tclk)	120	167	120	167
25a	M,IO	8/16	Data setup to IOCHRDY deasserted (8-bit even)		85		74
25b	M,IO	8	Data setup to IOCHRDY deasserted (8-bit odd)		75		64
26a	M	16	LA hold to MEMx* active	41		30	
26b	M	8	LA hold to MEMx* active	-21		-32	
28	M	16	ENDXFR* setup to SYSCLK falling edge	22			
29	M	16	ENDXFR* hold from SYSCLK falling edge	22			
36	M	16	LA setup to ENDXFR* asserted		180		158
37	M	16	SA setup to ENDXFR* asserted		83		61

Table 1. Memory and I/O Timing



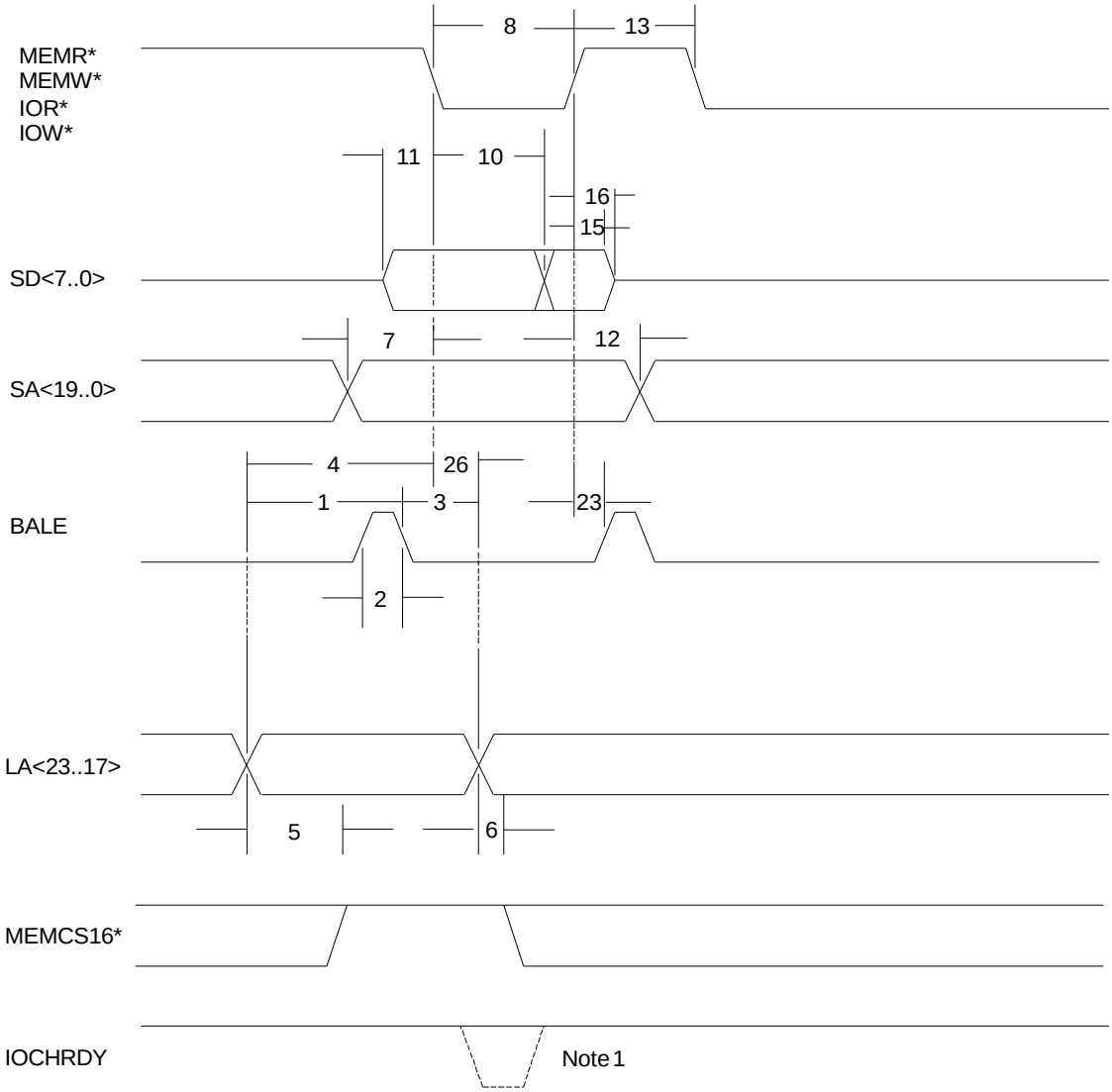
Note 1: IOCHRDY timings apply if deasserted. See Figure 4.

Figure 1. 16-bit Memory Timing



Note 1: IOCHRDY timings apply if deasserted. See Figure 4.

Figure 2. 16-bit I/O Timing



Note 1: IOCHRDY timings apply if deasserted. See Figure 4.

Figure 3. 8-bit Memory and I/O Timing

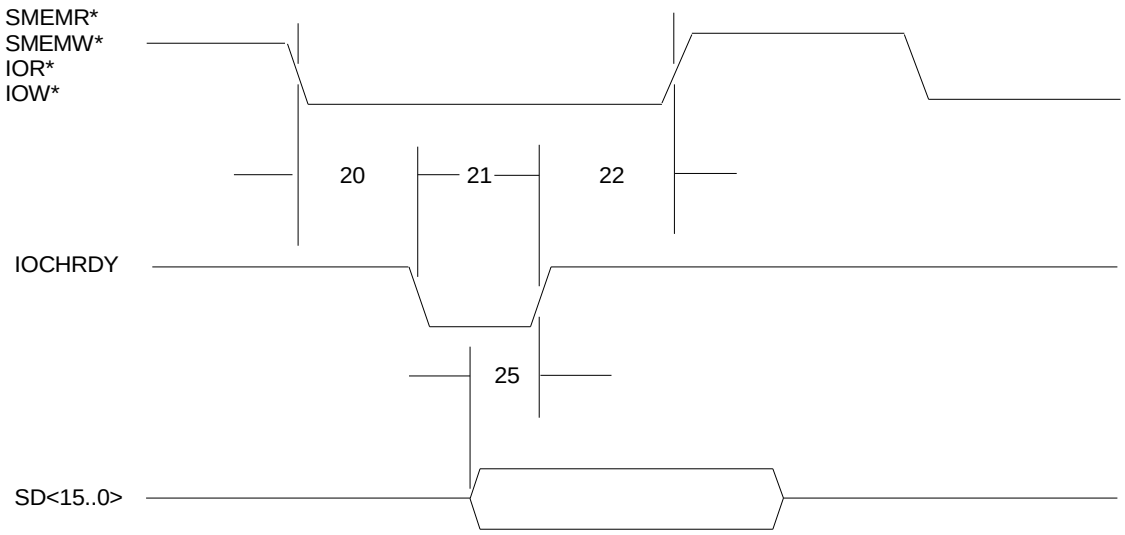
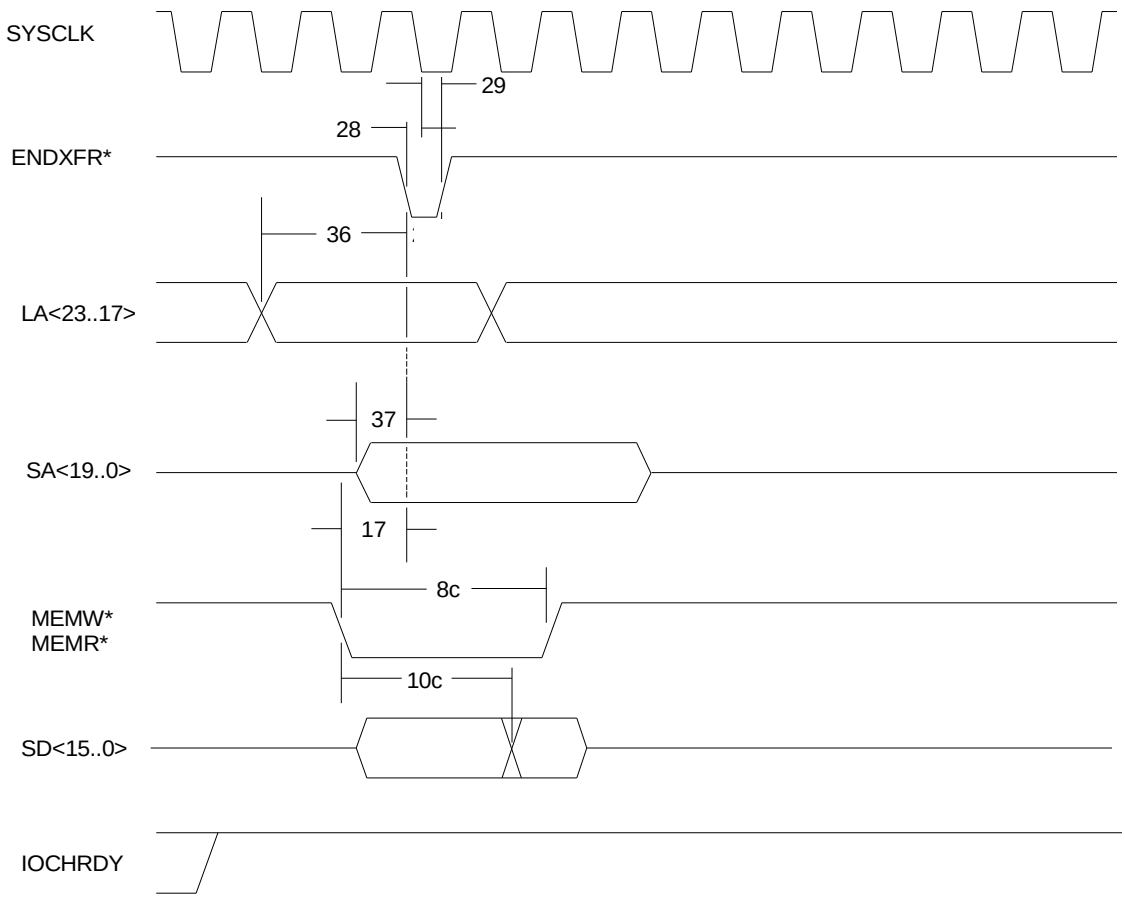


Figure 4. IOCHRDY Timing

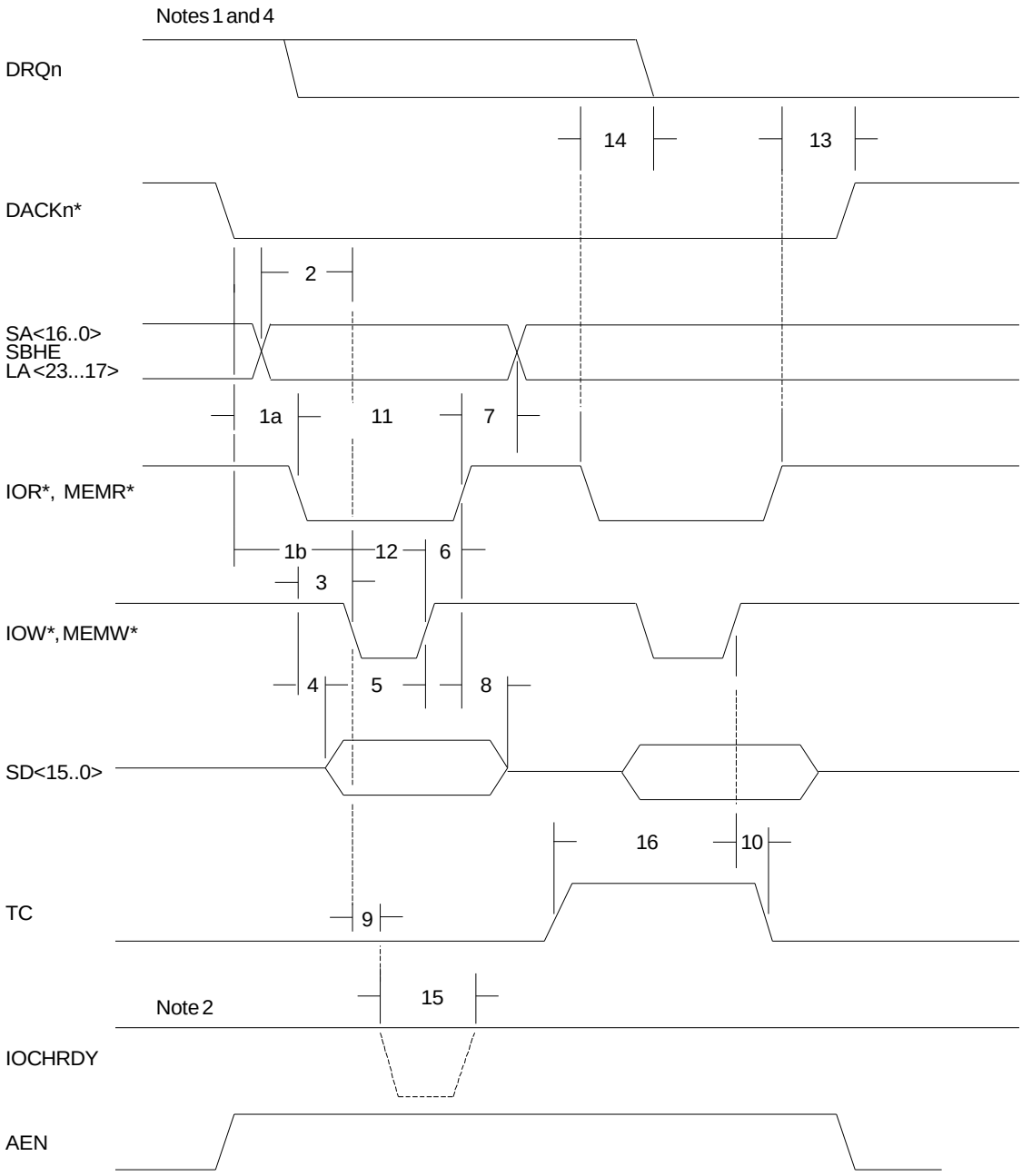


Note 1: Assertion of ENDXFR* within the maximum time from command is only required for a 16-bit cycle with zero wait states. Otherwise, ENDXFR* may be asserted at any time during the cycle while command is asserted.

Figure 5. ENDXFR Timing*

REF	DESCRIPTION	DRIVER		RECEIVER	
		MIN	MAX	MIN	MAX
1a 1b	DACKn*, AEN setup to IOR* DACKn*, AEN setup to IORW*	76 321		65 310	
2	Address setup to MEMW*, IOW*	102		91	
3a 3b	IOR* setup to MEMW* MEMR* setup to IOW*	246 0		234 0	
4a 4b 4c	Data access from IOR* 8/16bit Data access from MEMR* 16bit Data access from MEMR* 8bit		220 173 332		242 195 337
5	Data setup to IOW* unasserted	164		142	
6	Read command hold from write command	50		39	
7	SBHE*, address hold	53		42	
8	Data hold from read command	11		0	
9a 9b	IOCHRDY deasserted from 16bit memory command IOCHRDY deasserted from 8bit memory command		81 384		103 406
10	TC hold from command unasserted	60		49	
11a 11b	IOR* pulse width MEMR* pulse width	797 547		786 536	
12	IOW*, MEMW* width	500		489	
13a 13b 13c	DACKn* hold from IOW* DACKn* hold from IOW* AEN hold from command	114 173 41		103 162 30	
14	DREQ inactive from IOx*		119		141
15	IOCHRDY low width	Tclk	15600	Tclk	15611
16	TC setup to command unasserted	511		500	

Table 2. DMA Timing



Note 1: DRQn may be deasserted any time after DACKn* during a block mode DMA transfer.

Note 2: IOCHRDY may be deasserted to insert additional wait states. Additional bus wait states are added in units of two bus clocks.

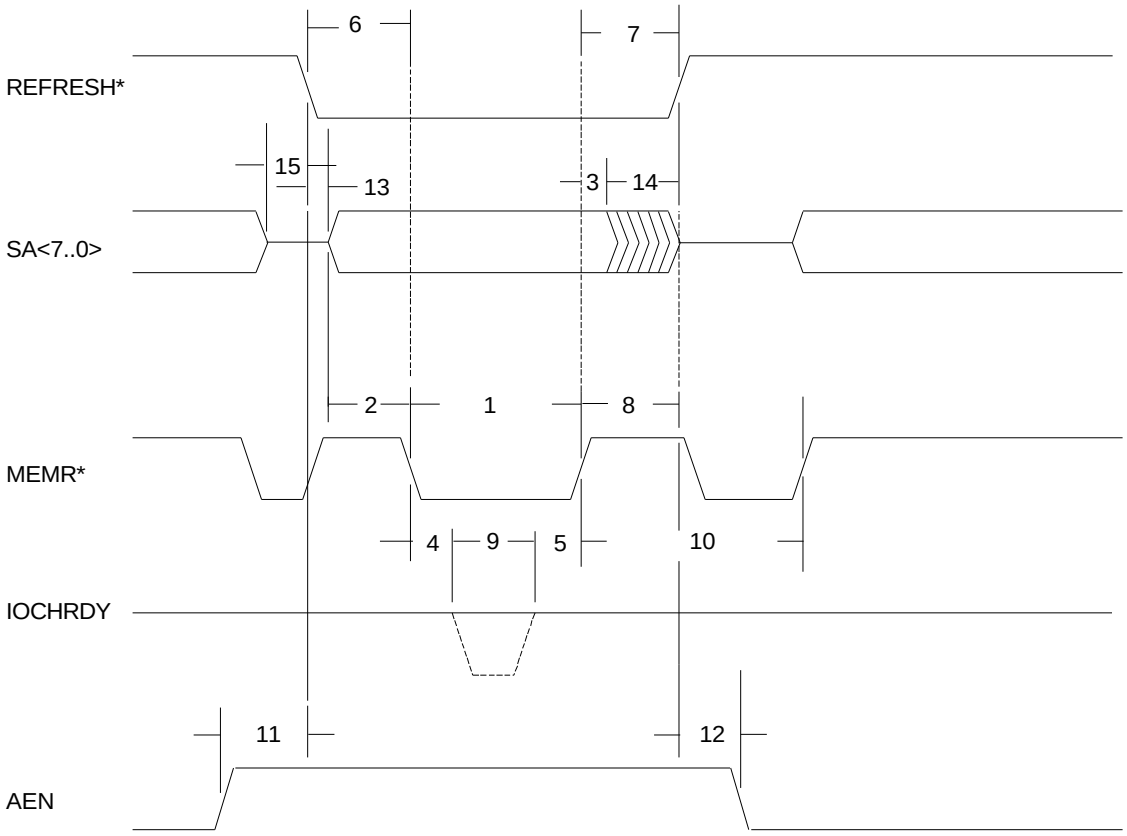
Note 3: The DMA controller activates TC during the last cycle of a DMA request.

Note 4: DMA transfers may be broken up into multiple back-to-back cycles where the DMA controller removes DACKn* and optionally releases the bus to allow higher priority cycles to occur. In this case, DACKn* will be temporarily deasserted even though DRQn is still asserted.

Figure 6. DMA Timing

REF	DESCRIPTION	DRIVER		RECEIVER	
		MIN	MAX	MIN	MAX
1	MEMR* pulse width	214		203	
2	SA<0...7> setup to MEMR*	81		70	
3	SA<0...7> hold from MEMR*	36		25	
4	IOCHRDY deasserted from MEMR*		81		159
5	MEMR* deasserted from IOCHRDY	125		125	
6	REFRESH* setup to MEMR*	125		114	
7	REFRESH* hold from MEMR* (Note 1)	31	250	20	239
8	SA<11...0> tri-state from MEMR* high		Tclk		
9	IOCHRDY width	Tclk		Tclk	
10	AM ownership delay (Note 2)	2*Tclk		2*Tclk	
11	AEN asserted to REFRESH* active	11		0	
12	AEN hold to REFRESH* inactive	11		0	
13	REFRESH* asserted to SA<0...7> valid	11		0	
14	REFRESH* hold from SA<0...7> valid	11		0	
15	Address and Control disabled to REFRESH* asserted	0		0	

Table 3. Refresh Timing



Note 1: The temporary master may exceed the maximum REFRESH* hold time in order to conduct another refresh operation.

Note 2: The temporary master, if the current master, must tri-state the address and command signals prior to driving REFRESH* high (1).

Figure 7. REFRESH Timing

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 14. Februar 2011	
Ort, Datum	Unterschrift