auszulösen. Im normal arbeitenden ST-System ist dieser HBlank-Interrupt jedoch nicht vorgesehen. Ein laufender Prozeß würde nämlich in der Monochrom-Betriebsart sonst alle 28 µSek für mindestens sieben µSek unterbrochen. Solange dauert es nämlich, bis der Prozessor den Interrupt erkannt, quittiert und abgeschlossen hat. Dazu kommt natürlich noch die Zeit für die eigentliche Interrupt-Routine, die ausgeführt werden soll. Der HBlank-Interrupt würde somit schon mindestens 25% der Prozessorzeit beanspruchen; ein schöner Bremsklotz also.

Jede Menge Manipulationen möglich

Die Programmierung des ST-Video-Controllers wird dem Programmierer durch eine Reihe von XBIOS-Aufrufen erleichtert. Es sind dies die XBIOS-Funktionen #2 ("Physbase"), #3 ("Logbase"), #4 ("Getrez"), #5 ("Setscreen"), #6 ("Setpallette"), #7 ("Setcolor") und #37 ("Vsync"). Nähere Einzelheiten dazu im Teil I, Kapitel 1, in der "XBIOS-Referenz".

Die komfortablen Möglichkeiten des ST, Text und Grafiken gleichermaßen gut darzustellen, liegen darin begründet, daß es keinen speziellen Character- und Grafik-Modus gibt.

In herkömmlichen Systemen hat man üblicherweise einen sogenannten Charactergenerator (Zeichengenerator), der in Zusammenarbeit mit einem Video-Controller die Darstellung von Textzeichen auf dem Bildschirm ermöglicht. Die Zeichen können so nur in einem relativ groben Raster auf dem Bildschirm angeordnet werden (Zeichenmodus). Ein zusätzlicher Grafikmodus ermöglich das Setzen und Rücksetzen von einzelnen Bildpunkten.

Beim ST gibt es nur den Grafikmodus. Alle Zeichen und Buchstaben werden durch einzelnes Setzen und Rücksetzen von Pixeln gebildet. Das beansprucht natürlich einiges an Prozessorzeit und Speicherplatz, gestattet dafür aber das komfortable Mischen von Grafik und Text, da Textzeichen für den ST nichts anderes als Gruppen von Grafikpunkten darstellen. Außerdem lassen sich so Zeichen im Pixelraster beliebig auf dem Bildschirm plazieren (z. B. Textverarbeitung "Signum!"). Proportionalschrift ist somit auch möglich!

Der Blitter

Zur Entlastung der CPU und Beschleunigung der Graphikausgabe wurde von ATARI schon bei der Entwicklung der ST-Computerserie ein Hilfsbaustein vorgesehen, der wesentliche Aufgaben bei der Aktualisierung des Bildschirmspeichers übernehmen sollte. Zunächst wurden die Geräte der MEGA ST-Serie mit dem Bit-Block-Transfer-Processor (Blitter) ausgestattet. Der Blitter übernimmt dabei die Aufgaben, welche bisher von der BitBlt-Funktion der Line-A-Routine #$A007 ausgeführt wurden. Es handelt sich hierbei also um eine hardwaremäßige Realisierung des BitBlt-Algorithmus.

Bei einem Bit-Block-Transfer können Bitfelder-Daten von einer Ausgangsposition im Bildschirmspeicher an eine Zielposition übertragen werden.

Dabei ist es möglich, die Ursprungsdaten über eine wählbare logische Verknüpfung mit dem bereits im Zielspeicherbereich vorhandenen Bitmuster zu verküpfen. Es ist aber ebenfalls möglich, einen Zielbereich mit einem vorher definiertem Muster zu füllen und so beispielsweise Bildschirmausschnitte zu löschen oder vorzubesetzen.

Unter Zuhilfenahme der BitBlt-Funktion lassen sich so relativ einfach Funktionen wie Text-Scrolling, Textumsetzungen in Fettschrift, Italic oder Outline realisieren. Funktionen wie Window-Updating und -Verschiebung profitieren ebenfalls von den Möglichkeiten, welche die BitBlt-Funktion bietet.

Der Atari ST-Blitter

Der Grundalgorithmus der BitBlt-Funktion wurde erstmalig durch Newman & Sproull im Jahre 1979 bei der Beschreibung der RasterOp-Funktion, im Buch "Principles of interactive computer graphics" definiert. Diese Definition sah einen "Bit für Bit"-Block-Transfer vor. Es waren nur wenige logische Verknüpfungsmöglichkeiten zwischen Quell- und Zielbitfeldern vorgesehen.

Im Laufe der Zeit wurden die Grundfunktionen um weitere Verknüpfungsmöglichkeiten erweitert und die Parallelverarbeitung von Bits implementiert.

Außerdem wurde vorgesehen, die Ursprungsdaten mit einem frei definierbaren Grundmuster "vorzuverarbeiten" (sogen. Halftone Pattern) und erst das daraus entstehende Ergebnis mit den Zieldaten zu verknüpfen. Ein nach diesen Definitionen erstellter RasterOp-Chip (die "VL16160 RasterOp Graphics/Boolean Operation ALU" von VLSI-Technology) konnte jedoch immer nur Einzeldaten bearbeiten. Die Quell-, Halftone- und Zieldaten mußten laufend von der CPU in den RasterOp-Chip nachgeladen werden. DMA-Betrieb war nicht möglich.

Der ATARI ST-Blitter ist ein eigenständiger Prozessor, der alle Definitionen der BitBlt-Funktion erfüllt. Er verarbeitet die Daten wortweise (also maximal 16 Bits auf einmal) und braucht von der CPU nicht für jedes Datenwort mit den Quell-, Halftone- und Zieldaten nachgeladen zu werden.

Die erforderlichen Quell- und Zieldaten holt er sich als DMA-Baustein selbst aus dem RAM und legt sie nach Verarbeitung auch wieder dort ab. Außerdem verfügt er über ein eigenes schnelles Halftone-RAM. Zugriffe auf das RAM des ST muß der Blitter mit der CPU und der DMA-Einheit "absprechen".

Bit-Block Verarbeitung

Beim Bit-Block-Transfer werden Bitfeld-Daten aus einem Ursprungs-Speicherbereich mit den Daten eines Bitfelds im Ziel-Speicherbereich unter Verwendung eines Booleschen Operators verknüpft.

Die Verknüpfung wird dabei jeweils auf die zueinandergehörenden Bits des Quell- und Zielbitfelds angewendet.

Folgende logische Verknüpfungsmöglichkeiten ergeben sich dabei:

Verknüpfgs.-Nr.	Ergebnis der Verknüpfung im Zielbitfeld
0	Zielbit = 0
1	Zielbit = Quellbit AND Zielbit
2	Zielbit = Quellbit AND NOT Zielbit
3	Zielbit = Quellbit
4	Zielbit = NOT Quellbit AND Zielbit
5	Zielbit = Zielbit
6	Zielbit = Quellbit XOR Zielbit
7	Zielbit = Quellbit OR Zielbit
8	Zielbit = NOT Quellbit AND NOT Zielbit
9	Zielbit = NOT Quellbit XOR Zielbit
10	Zielbit = NOT Zielbit
11	Zielbit = Quellbit OR NOT Zielbit
12	Zielbit = NOT Quellbit
13	Zielbit = NOT Quellbit OR Zielbit
14	Zielbit = NOT Quellbit OR NOT Zielbit
15	Zielbit = 1

Bit-Block-Transfer mit dem Blitter

Der ST-Blitter verfügt über eine Anzahl von Registern, die an den in der Abbildung 2.13 aufgeführten Adressen im Speicherraum des ST zu finden sind (sofern der Blitter eingebaut ist!).

Blitter-Registerzugriffe müssen jedoch immer im Supervisor-Modus erfolgen, sonst "wirft" der ST Bomben!

Zur Erläuterung des Ablaufes eines Bit-Block-Transfers mit dem Blitter soll das in Abbildung 2.14 gezeigte Datenflußbild dienen.

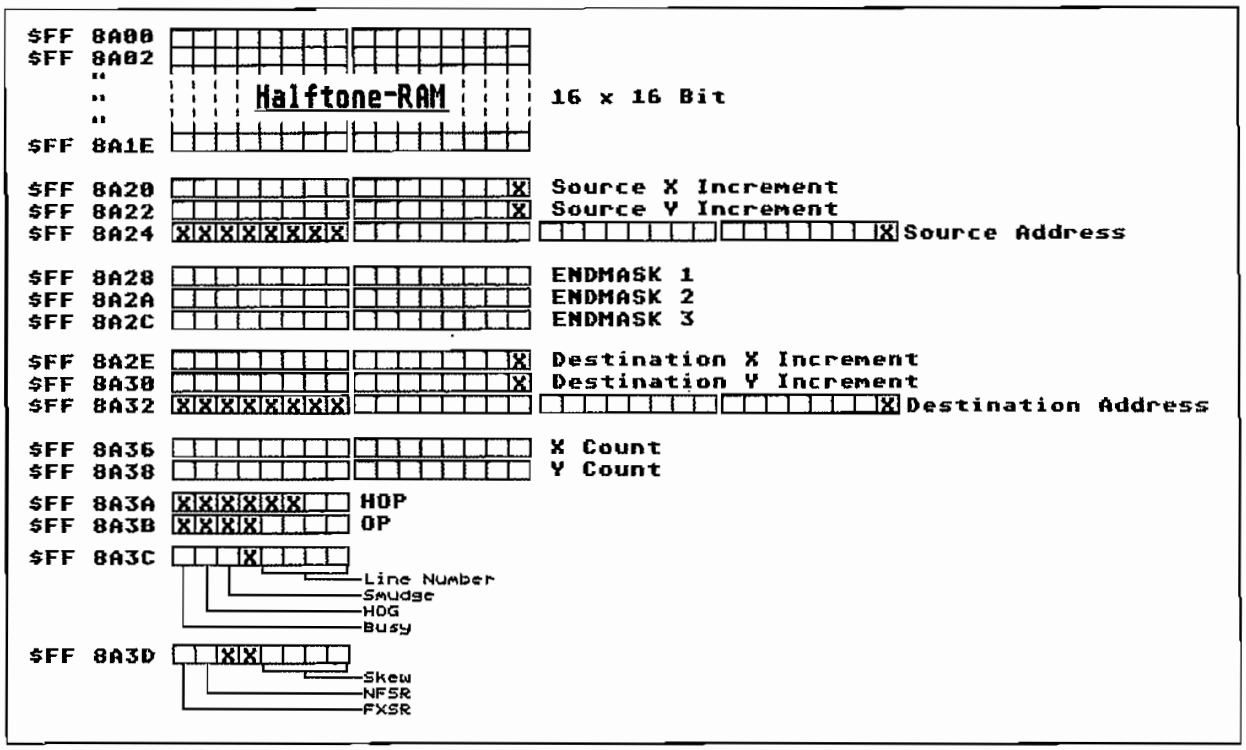


Abb. 2.13: Die Register des Blitter-Chips

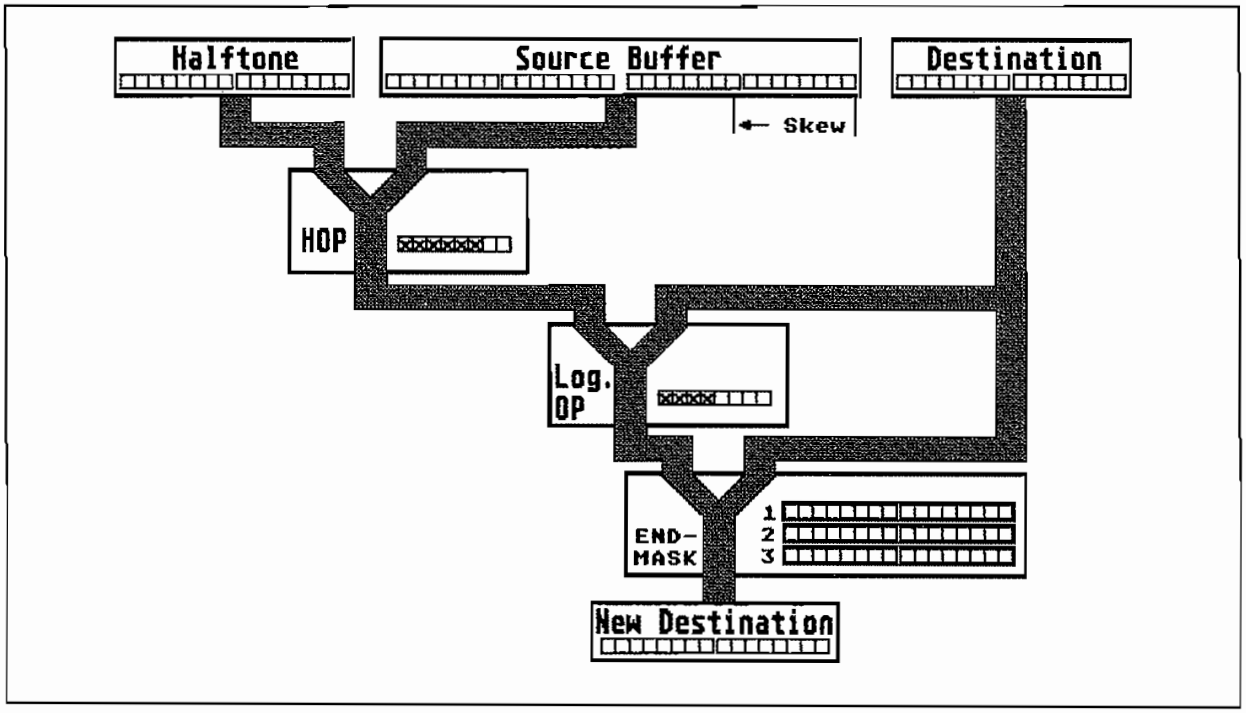


Abb. 2.14: Der Datenfluß beim Bit-Block-Transfer

Nun zur Arbeitsweise des Blitters. Betrachten wir zunächst eine einfache Blitter-Operation: Ein Bit-Block soll vollständig mit 0- oder 1-Bits gefüllt werden.

Zur Wahl der Verknüpfung muß das

Operation-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A3B	R/W	Verknüpfungsart festlegen	Byte	OP

im Blitter mit der entsprechenden Verknüpfungs-Nummer geladen werden (Beispiel: Füllen des Zielbitfeldes mit 0-Bits = OP-Nr. 0, Füllen mit 1-Bits = OP-Nr. 15). Daten aus dem Quellbitfeld und Halftone-RAM sind in diesem Fall nicht erforderlich (der Source-Buffer und das Halftone-RAM werden also nicht benötigt). Der für die Durchführung der logischen Verknüpfung zuständige Funktionsblock (im Datenflußbild mit Log. OP bezeichnet) erzeugt die nötigen 0- bzw. 1-Bits für die Fülloperation.

Bei jeder Blitter-Operation wird der Funktionsblock ENDMASK gebraucht. Hier muß nämlich festgelegt werden, welche Bits im Zieldatenwort geändert werden müssen und welche nicht. Wenn alle Bits eines Wortes im Zielbitfeld mit 0- oder 1-Bits gefüllt werden, schreibt der Blitter einfach ein Datenwort nach dem anderen, ohne jemals Daten aus dem Zielbitfeld lesen zu müssen. Sind dagegen jedoch nicht alle Bits eines Wortes im Zielbitfeld zu ändern, so geben die ENDMASK-Register die Bitpositionen an, die im Zieldatenwort geändert werden müssen.

ENDMASK1-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A28	R/W	Zu ändernde Zielbits (Zeilenanfang)	Word	Endmask1

Dieses Register legt fest, welche Bits im ersten Datenwort einer Zeile des Zielbitfeldes geändert werden sollen.

ENDMASK2-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A2A	R/W	Zu ändernde Zielbits	Word	Endmask2

Wird benutzt, wenn keines der anderen beiden ENDMASK-Register angewendet wird (das ist für alle Datenworte zwischen dem ersten und letzten Datenwort einer Zeile des Zielbitfeldes der Fall).

ENDMASK3-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A2C	R/W	Zu ändernde Zielbits (Zeilenende)	Word	Endmask2

Für das Schreiben des letzten Datenworts einer Zeile im Zielbitfeld wird dieses Register als Maske benutzt, um festzulegen, welche Bits geändert werden müssen.

Folglich ist also immer dann ein READ-MODIFY-WRITE-Zyklus für das Zieldatenwort erforderlich, wenn in dem zugehörigen ENDMASK-Register nicht alle Bits gesetzt sind (zur Veranschaulichung dient die Abbildung 2.15).

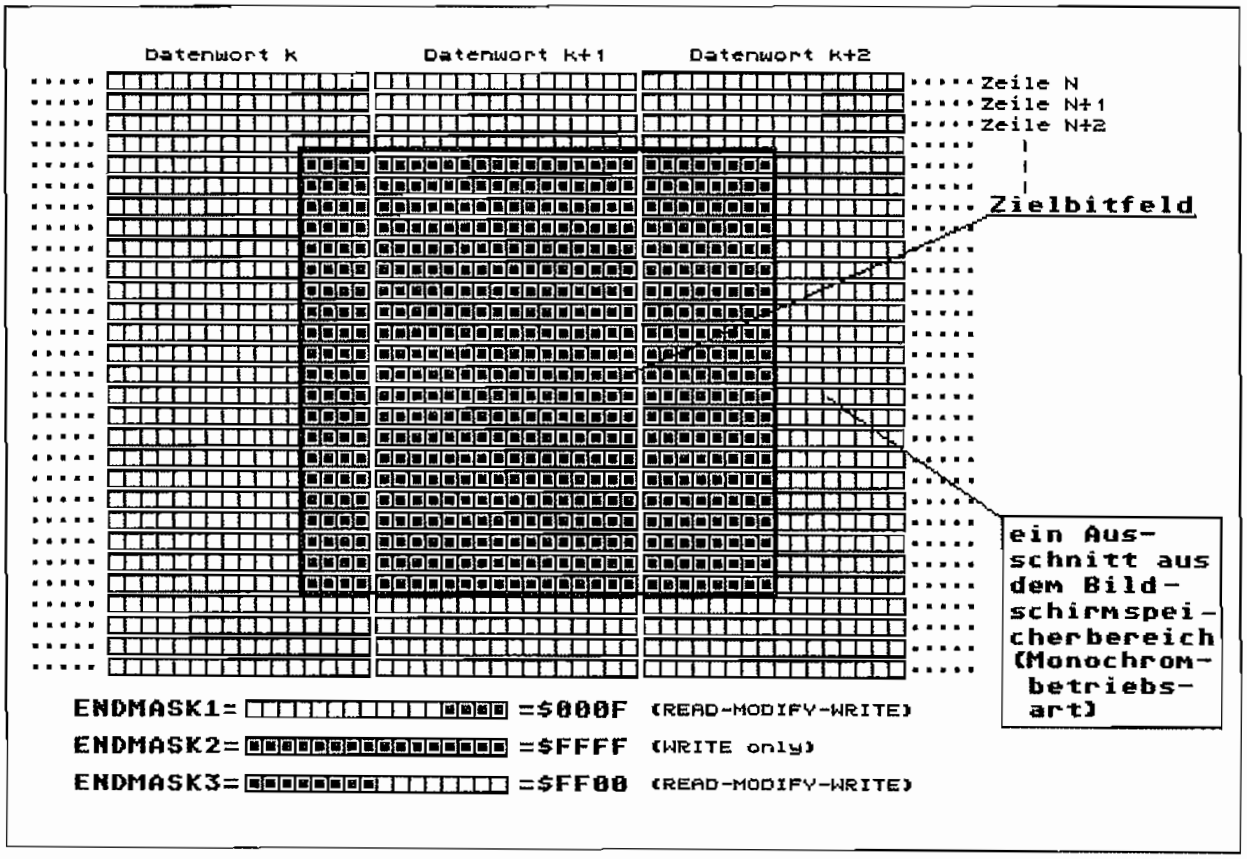


Abb. 2.15: Eine einfache Aufgabe für den Blitter: das Füllen eines Zielbitfeldes mit 1-Bits

Ist eine Zeile des Zielbitfelds übrigens "schmaler" als ein Datenwort (z. B. nur 12 Bit breit), so wird nur das ENDMASK1-Register benutzt.

Nachdem der Blitter ein Datenwort im Zielspeicherbereich bearbeitet hat, muß die nächste zu bearbeitende Adresse ermittelt werden. Dazu benötigt der Blitter die Angaben im Destination-X-Increment-, Destination-Y-Increment-, X-Count-, Y-Count- und Destination-Address-Register.

Mit diesen Registern wird sozusagen der zu bearbeitende Bitblock von seinen Abmessungen her festgelegt.

X-Count-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A36	R/W	Anzahl Words/Zeile im Zielbitfeld	Word	X_Count

Hiermit wird festgelegt, wie viele Datenworte zu einer Zeile des Zielbitfeldes gehören (in dem Beispiel in der Abbildung 2.15 steht also hier ein Wert von 3).

Der kleinste Wert ist 1 (mindestens ein Datenwort muß im Zielspeicherbereich schon zu bearbeiten sein, sonst braucht man ja keine Transferoperation durchzuführen!), und der größte Wert ist 65536 (wird durch einen Wert von $0000 dargestellt).

Der Blitter zählt jedesmal dann den Inhalt des Registers um 1 herunter, wenn er ein Datenwort geschrieben hat. Bei Erreichen von $0000 wird automatisch das X-Count-Register wieder auf den ursprünglich programmierten Wert gesetzt. (Ähnlichkeiten in der Arbeitsweise mit dem Timer-Data-Register des MFP sind zufällig und nicht beabsichtigt.) Beim Auslesen erhält man die Zahl der Datenworte, die in der augenblicklich in Bearbeitung befindlichen Zeile noch geschrieben werden müssen.

Y-Count-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A38	R/W	Zeilenzahl im Zielbitfeld	Word	Y_Count

Gibt an, aus wie vielen Zeilen das Zielbitfeld besteht. Wertebereich wie beim X-Count-Register (mindestens eine Zeile, maximal 65536 Zeilen sind möglich. Im Beispiel in Abbildung 2.15 also ein Wert von 21).

Nachdem eine komplette Zeile des Zielbitfeldes geschrieben ist, wird der Inhalt des Registers jeweils um 1 vermindert. Bei $0000 ist der Transfer dann beendet! Ein Lesezugriff liefert die Zahl der Zeilen, die noch geschrieben werden müssen.

Destination-X-Increment-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A2E	R/W	Distanz zwischen zwei Words im Zielbitfeld	Word	Dst_Xinc

Wird als vorzeichenbehaftete Zahl interpretiert (höchstwertiges Bit wird für Vorzeichen benutzt). Das niedrigstwertige Bit wird nicht benutzt, weil der Blitter immer mit Wordzugriffen arbeitet. Und die sind nur auf gerade Adressen erlaubt.

Anzugeben ist in diesem Register der Abstand in Bytes zwischen zwei Datenwörtern einer Zeile des Zielbitfeldes (Achtung bei Bitplanes! In Low-Resolution steht hier also 8 oder –8, bei Mid-Resolution dann 4 bzw. –4 und im Monochrom-Modus entspr. 2 oder –2). Ein negativer Wert weist darauf hin, daß die Zeile von rechts nach links bearbeitet wird.

Nachdem der Blitter ein Zieldatenwort geschrieben hat, wird der Wert aus dem Destination-X-Increment-Register unter Berücksichtigung seines Vorzeichens zu dem Inhalt im Destination-Address-Register addiert (jedoch nur, wenn das X-Count-Register einen Wert <>1 hat, also eine Zeile im Zielbitfeld aus mehr als einem Datenwort besteht).

So kann der Blitter in einstellbaren Schrittweiten den Zielspeicherbereich bearbeiten und nicht nur unmittelbar nebeneinanderliegende Datenwörter ansprechen (wichtig bei Bit-Block-Transfers auf den Bitplanes in der niedrigen und mittleren Bildschirmauflösung, bei denen ja die zu einem Plane gehörigen Datenwörter nicht unmittelbar nebeneinander, sondern vier bzw. acht Bytes auseinanderliegen!).

Destination-Y-Increment-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A30	R/W	Distanz zwischen zwei Zeilen im Zielbitfeld	Word	Dst_Yinc

Daten werden auch hier vorzeichenbehaftet betrachtet. Das Register enthält den Byte-Abstand von der *letzten* Zieladresse einer Zeile zur *ersten* Zieladresse in der nächsten Zeile des Zielbitfeldes. Dieser Wert wird nach dem Schreiben des letzten Datenworts einer Zeile

vorzeichenrichtig zum Inhalt des Destination-Address-Registers addiert. Somit zeigt das Destination-Address-Register dann wieder auf das nächste zu bearbeitende Zieldatenwort.

Sollte das X-Count-Register von vornherein nur mit einem Wert von 1 geladen worden sein (weil eine Zeile im Zielbitfeld max. ein Datenwort breit ist), arbeitet der Blitter bei der Ermittlung der nächsten Zieldatenadresse ausschließlich mit dem Destination-Y-Increment-Register. Eine Zeile im Zielbitfeld besteht dann ja nur aus einem Datenwort, und nach Bearbeitung dieses einen Wortes ist schon die nächste Zeile an der Reihe.

Destination-Adress-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A32	R/W	Zeiger auf (nächstes) Zieldatenwort	Long	Dst_Addr

Zu Beginn eines Bit-Block-Transfers ist hier die Anfangsadresse des ersten Wortes des Zielbitfeldes einzutragen. Während des Transfers wird das Register ständig mit Hilfe der Werte im Destination-X-Increment- und Destination-Y-Increment-Register auf den aktuellen Stand gebracht. Man würde deshalb bei einem Lesezugriff während eines Transfers die Adresse des nächsten zu bearbeitenden Zieldatenwortes erhalten.

Für etwas komplexere Operationen werden noch einige weitere Register gebraucht. Unter anderem werden diese erforderlich, wenn z. B. ein Zielbitfeld mit Daten eines Musters AND-verknüpft werden soll. Als Ergebnis bleiben also im Zielbitfeld nur jene Bits gesetzt, die auch im Muster gesetzt sind. Hier kommt nun das Halftone-RAM (Größe: 16 Datenwörter) ins Spiel.

Zunächst wird das Muster, mit dem das Zielbitfeld verknüpft werden soll, als 16x16 Bitfeld ins Halftone-RAM eingeschrieben. Dann wird das OP-Register mit dem Wert für die AND-Verknüpfung (OP-Nr.1) geladen. Anschließend muß dem Blitter noch mitgeteilt werden, daß die Verknüpfung von Daten im Zielbitfeld nur mit dem Muster aus dem Halftone-RAM durchgeführt werden soll.

Das geschieht über das

Halftone-Operation-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A3A	R/W	Halftone-Verknüpfungsvorschrift	Byte	HOP

Folgende Möglichkeiten sind einstellbar:

HOP	Verknüpfungsergebnis
0	Alle Bits = 1
1	Nur Halftone-Daten
2	Nur Source-Daten
3	Source AND Halftone

Das HOP-Register muß bei obigem Beispiel also mit dem Wert von 1 geladen werden, weil ja nur die Musterdaten aus dem Halftone-RAM benötigt werden (Achtung beim Registerzugriff! Das HOP-Register ist nur 1 Byte breit.)

Außerdem kommt jetzt noch ein weiterer Parameter ins Spiel:

Line-Number-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A3C	R/W	Zeiger auf Halftone-Verknüpfungsword	Byte	Line_Num

Bei einer Halftone-Operation gibt die Line Number an (gebildet aus den niedrigstwertigen vier Bits in diesem Register), welches der 16 Wörter im Halftone-RAM als Verknüpfungsdatenwort für die gerade zu bearbeitende Zeile benutzt wird.

Abhängig vom Destination-Y-Increment-Register wird am Ende einer Zeile die Line Number um 1 erhöht (bei positivem Wert in Destination-Y-Increment-Register) oder um 1 erniedrigt (bei negativem Wert im Destination-Y-Increment-Register), um so das nächste Verknüpfungsdatenwort im Halftone-RAM zu adressieren. Die Line Number wird "rund" gezählt, d. h., bei einem Wert von 15 wird bei weiterer Erhöhung der Zähler wieder bei 0 beginnen. Bei einer weiteren Dekrementierung bei einem Zählerstand von 0 wird die Line Number entsprechend auf 15 springen.

Die schon bekannten Register (Destination-Address-, Destination-X-Increment, Destination-Y-Increment-, X-Count- und Y-Count-Register) werden entsprechend mit Werten, die sich aus der Lage und den "Abmessungen" des zu füllenden Zielbitfeldes ergeben, geladen.

Im Prinzip funktioniert die ganze Angelegenheit genauso, wenn man statt des Musters aus dem Halftone-RAM ein Quellbitfeld zur Verknüpfung mit dem Zielbitfeld heranzieht. Nur das HOP-Register muß dann anders gesetzt werden (HOP-Nr.2 oder HOP-Nr.3).

Der Quellbitblock hat dann die gleichen "Abmessungen" wie der Zielbitblock (X-Count und Y-Count gelten sowohl für den Source- als auch den Destination-Bit-Block!). Unterscheiden dürfen sich jedoch die Werte für X-Increment und Y-Increment, weshalb auch für den Quellbitblock eigene Register zur Aufnahme dieser beiden Werte vorgesehen sind.

Source-X-Increment-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A20	R/W	Distanz zwischen zwei Words im Quellbitfeld	Word	Src_Xinc

Hier wird der Abstand (in Bytes) zwischen zwei Datenwörtern in einer Zeile des Quellbitblocks eingetragen (unter Berücksichtigung des Vorzeichens!). Nachdem jeweils ein Source-Wort gelesen wurde, wird der Wert dieses Registers zum Inhalt des Source-Address-Registers vorzeichenrichtig addiert (solange X-Count <>1 ist!).

Source-Y-Increment-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A22	R/W	Distanz zwischen zwei Zeilen im Quellbitfeld	Word	Src_Yinc

Abstand in Bytes zwischen dem letzten Wort einer Zeile des Quellbitfeldes und dem ersten Wort der nächsten Zeile (mit Vorzeichen!).

Wenn das letzte Wort einer Zeile des Quellbitblocks gelesen ist, wird der Wert im Source-Y-Increment-Register zum Inhalt des Source-Address-Registers vorzeichenrichtig addiert. Das Source-Address-Register zeigt danach wieder auf das nächste zu lesende Datenwort des Quellbitfeldes.

Source-Adress-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A24	R/W	Zeiger auf (nächstes) Zieldatenwort	Long	Src_Addr

Zu Beginn eines Bit-Block-Transfers wird hier die Anfangsadresse des ersten Wortes des Quellbitfeldes eingetragen.

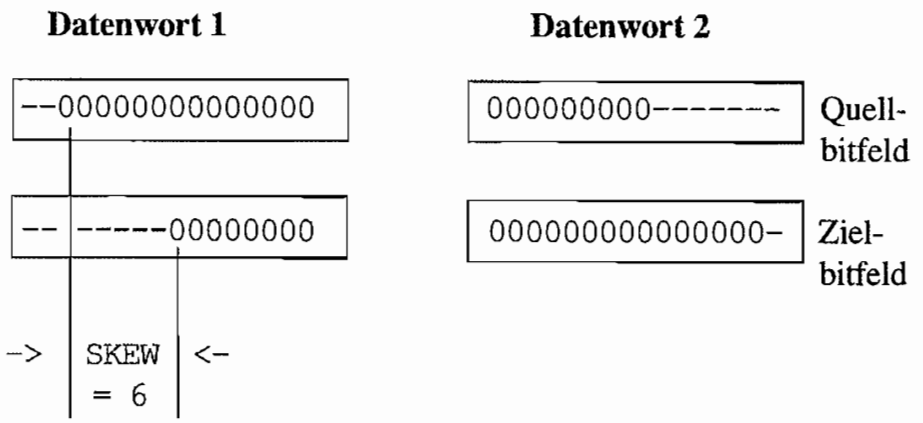
Während des Transfers wird das Source-Adress-Register ständig mit Hilfe der Werte im Source-X-Increment- und Source-Y-Increment-Register auf den aktuellen Stand gebracht. Bei einem Lesezugriff während des Transfers steht hier die Adresse des nächsten zu lesenden Quelldatenwortes. Werden Quellbitfeld-Daten in die Verarbeitung einbezogen, kommt der im Datenflußbild mit Source-Buffer bezeichnete Funktionsblock ins Spiel. Bei einer Bit-Block-Operation mit Sourcedaten spielt sich im Blitter dann folgendes ab (Zum besseren Verständnis sollte man die Abbildung 2.14 betrachten):

Die ersten beiden Sourceworte werden in den Source-Buffer eingelesen. Zur Verküpfung mit den Zielbits im Zieldatenwort muß eine Justierung (Bitverschiebung) der Sourcedaten im Source-Buffer erfolgen (das erste Bit des Quellbitfeldes soll ja mit dem ersten Bit des Zielbitfeldes verknüpft werden usw.). Den Grad der Justierung erfährt der Blitter aus dem SKEW-Register (dazu später mehr).

Ist das erste Quelldatenwort (nach entsprechender Justierung) mit dem ersten Zieldatenwort verknüpft, werden die 16 niederwertigen Bits im Source-Buffer in die 16 höherwertigen Bits des Source-Buffers übertragen. Das nächste zu lesende Sourcewort gelangt in die niederwertigsten 16 Bit des Registers. Es erfolgt wieder die Justierung im Source Buffer, dann die Verknüpfung mit dem Zieldatenwort usw.

Nun folgen noch die Erläuterungen für die verbleibenden Flags und Register. Unter der Adresse $FF 8A3D findet man die folgenden Flags und Werte:

SKEW (Label: "Skew") In die vier niederwertigsten Bits an Adresse $FF 8A3D muß der sogenannte Source-Skew eingetragen werden. Darunter versteht man die Zahl der nötigen Rechtsverschiebungen, die mit den Source-Daten im Source Buffer (siehe Datenflußschema) durchgeführt werden müssen, bevor diese Daten mit den Halftone- und Destination-Daten verknüpft werden dürfen. *Beispiel*: Dargestellt sind jeweils die ersten beiden Datenworte des Quell- und Zielbitfeldes.



Der Quellbitblock beginnt im Beispiel bei Bit 13 im Datenwort 1 des Quellbitfeldes, während der Zielbitblock erst bei Bit 7 im ersten Datenwort des Zielbitfeldes anfängt! Damit also jeweils die zusammengehörigen Bits (das erste Bit im Quellbitblock korrespondiert ja mit dem ersten Bit im Zielbitblock usw.) miteinander verknüpft werden können, müssen diese erst "übereinandergeschoben" werden.

Um wie viele Bitpositionen die Daten "geschoben" werden müssen, ist im Skew-Wert anzugeben.

FXSR (Force Extra Source Read) Ist das höchstwertige Bit im Blitter-Register an Adresse $FF 8A3D gesetzt, so wird zu Beginn einer jeden Zeile ein zusätzliches Datenwort eingelesen, um entsprechend "Luft" für Verschiebungen im Source-Buffer zu schaffen.

NFSR (No Final Source Read) Wenn Bit 6 im Blitter-Register an Adresse $FF 8A3D gesetzt ist, wird das letzte Quell-Datenwort einer Zeile nicht gelesen. Bei einigen Kombinationen von ENDMASK-Inhalten und SKEW-Werten kann auf das Lesen des letzten Datenwortes einer Zeile des Quellbitfeldes verzichtet werden.

Zu den FXSR- und NFSR-Bits hier noch einige Erläuterungen:

Ein Block mit einer Breite von beispielsweise 20 Bits benötigt mindestens zwei Datenworte/Zeile zu seiner Darstellung. Je nachdem, bei welchem Bit des ersten Datenwortes das Bitfeld beginnt, können für die Darstellung dieses 20 Bit breiten Bitfeldes aber auch drei Datenworte/Zeile nötig sein! Es ist also vorstellbar, daß im Quellbitfeld für die 20 Bits schon zwei Datenworte/Zeile ausreichend sind, während im Zielbitfeld drei Datenworte erforderlich sind. Der umgekehrte Fall ist natürlich genauso denkbar.

Daraus folgt die Notwendigkeit, evtl. eine unterschiedliche Zahl von Lese- und Schreibzugriffen in einer Zeile des Bit-Blocks durchführen zu müssen. Für diese Steuerung sind die beiden Bits FXSR und NFSR erforderlich.

Die folgenden Beispiele zeigen, abhängig von der Lage der Quell- und Zielbitfelder zueinander, die jeweils nötige Einstellung der FXSR- und NFSR-Bits: (Die Bearbeitung soll von rechts nach links erfolgen!)

S = Sourcebits	**D = Destin.-Bits**	**FXSR**	**NFSR**
----SSSSSSSSSSSS	SSSSSSSS--------	0	0
-------DDDDDDDDD	DDDDDDDDDDD-----		

S = Sourcebits	D = Destin.-Bits	FXSR	NFSR
----SSSSSSSSSSS	SSSSSSSS--------	0	1
----------DD	DDDDDDDDDDDDDDD DD------		
----------S	SSSSSSSSSSSSSSS SSS-----	1	0
-------DDDDDDDDD	DDDDDDDDDD-----		
----SSSSSSSSSSS	SSSSSSSS--------	1	0
--DDDDDDDDDDDDD	DDDDDD----------		

An Adresse $FF 8A3C (Label: "Line_Num", also dort, wo auch die Line Number zu finden ist) existieren noch die folgenden Steuerbits:

SMUDGE Dieses Bit befindet sich an der gleichen Adresse wie die Line Number für das Halftone-RAM ($FF 8A3C). Ist es gesetzt, bestimmt nicht die Line Number, welches Datenwort aus dem Halftone-RAM bei einer Halftone-Operation zur Verknüpfung benutzt wird, sondern die niederwertigsten vier Bits der verschobenen (ge"SKEW"ten) Sourcedaten im Source-Buffer. Das ergibt quasi zufällige Verknüpfungen mit den Daten im Halftone-RAM und kann für "Schmiereffekte" benutzt werden.

HOG Hier wird die Betriebsart des Blitters eingestellt. Bei gelöschtem Bit teilen sich CPU und Blitter Zugriffe auf den Bus zu gleichen Teilen auf. Jeder darf für 64 Zyklen an den Bus, während der andere angehalten wird!

- Ist das HOG-Bit gesetzt, wird die CPU so lange angehalten, bis der Bit-Block-Transfer beendet ist. Es darf aber durchaus die CPU mal den einen oder anderen Befehl ausführen, wenn der Blitter zwischendurch mal den Bus freigibt! Mit gesetztem HOG-Bit kann der Bit-Block-Transfer bis zu doppelt so schnell abgewickelt werden, als wenn sich CPU und Blitter den Bus im Verhältnis 1:1 teilen müssen. In beiden Betriebsmodi nimmt der Blitter aber Rücksicht auf den DMA-Baustein. Dieser hat nämlich Vorrang!

- ATARI betreibt den Blitter standardmäßig mit gelöschtem HOG-Bit (CPU und Blitter teilen sich Buszugriffe für jeweils 64 Buszyklen). Um jedoch nahezu die gleiche Verarbeitungsgeschwindigkeit wie bei gesetztem HOG-Bit zu erreichen (ca. 90% des HOG-Modus), macht die CPU während "ihrer" 64 Busszyklen nichts anderes, als den Blitter sofort wieder neu zu starten, so daß dieser unmittelbar (nach ca. sieben CPU-Buszyklen) die Buskontrolle zurückbekommt!

Programmbeispiel:

```
          :
          :
START:
        lea     Line_Num,A0     ; Pointer auf HOG-Register -> A0
        bset.b  #HOGBIT,(A0)    ; HOG-Bit löschen (Kein HOG-Modus!)

RESTART:
        bset.b  #BUSYBIT,(A0)   ; BUSY-Bit testen u. setzen (startet
        nop                     ; den BLITTER sofort wieder neu!)
        bne.s RESTART           ; Der "nop"-Befehl wird noch ausgeführt,
          :                     ; bevor der Blitter neu startet!
          :
```

Abb. 2.16: Programmbeispiel

BUSY

Wenn alle anderen Register des Blitters gesetzt sind, wird durch Setzen des BUSY-Bits im Blitter der Transfer ausgelöst. Der auf das Setzen des BUSY-Bits folgende Befehl wird von der CPU in der Regel aber noch ausgeführt, bevor der Blitter das Regiment übernimmt!

Die Interrupt-Leitung zum MFP, Port I3 (GPU done), ist direkt mit diesem Bit gekoppelt. Wenn der Blitter fertig ist, wird das BUSY-Bit zurückgesetzt, und die GPU-done-Leitung geht zurück auf Low.

Wer's gerne knifflig mag,

kann den Blitter ja direkt programmieren. Die besprochenen Register und Steuerbits müssen jedoch alle vom Programmierer vor dem Bit-Block-Transfer selbst gesetzt werden! Die Beispiele für SKEW, NFSR, FXSR usw. waren ja noch relativ anschaulich. Wer Lust hat, kann sich ja mal überlegen, was alles zu beachten ist, wenn sich zu übertragende Bitblöcke überlappen (und dabei gibt's dann noch die Möglichkeiten, daß die Sourceadresse kleiner oder größer als die Zieladresse ist)!

Aber ATARI hat den Bit-Block-Transfer in neueren TOS-Versionen "transparent" für den Programmierer ausgelegt. Das bedeutet nichts anderes, als daß die BitBlt-Funktion die mit der Line-A Funktion $A007 aufgerufen wird, sowohl durch den Blitter als auch in Softwareemulation durch die CPU (wie gehabt) ausgeführt werden kann. Wenn der Blitter eingebaut ist, bietet

einem das TOS in der Desktop-Menüleiste unter dem Eintrag "Extras" die Möglichkeit, diesen ein- und auszuschalten. Die gleiche Funktion ist mit dem XBIOS-Aufruf #64 ("Blitmode") möglich. Sie kann auch benutzt werden, um abzufragen, ob der Blitter eingebaut ist. Näheres dazu in Teil I, Kapitel 1, in der "XBIOS-Referenz". Zur hardwaremäßigen Einbindung des Blitters sei auf die Abbildung 2.17 verwiesen.

Wie dort zu sehen, handelt es sich beim Blitter mit seinen 23 Adreß- und 16 Datenleitungen sowie dem kompletten Satz an Steuer- und Meldeanschlüssen um einen vollwertigen Mikro-

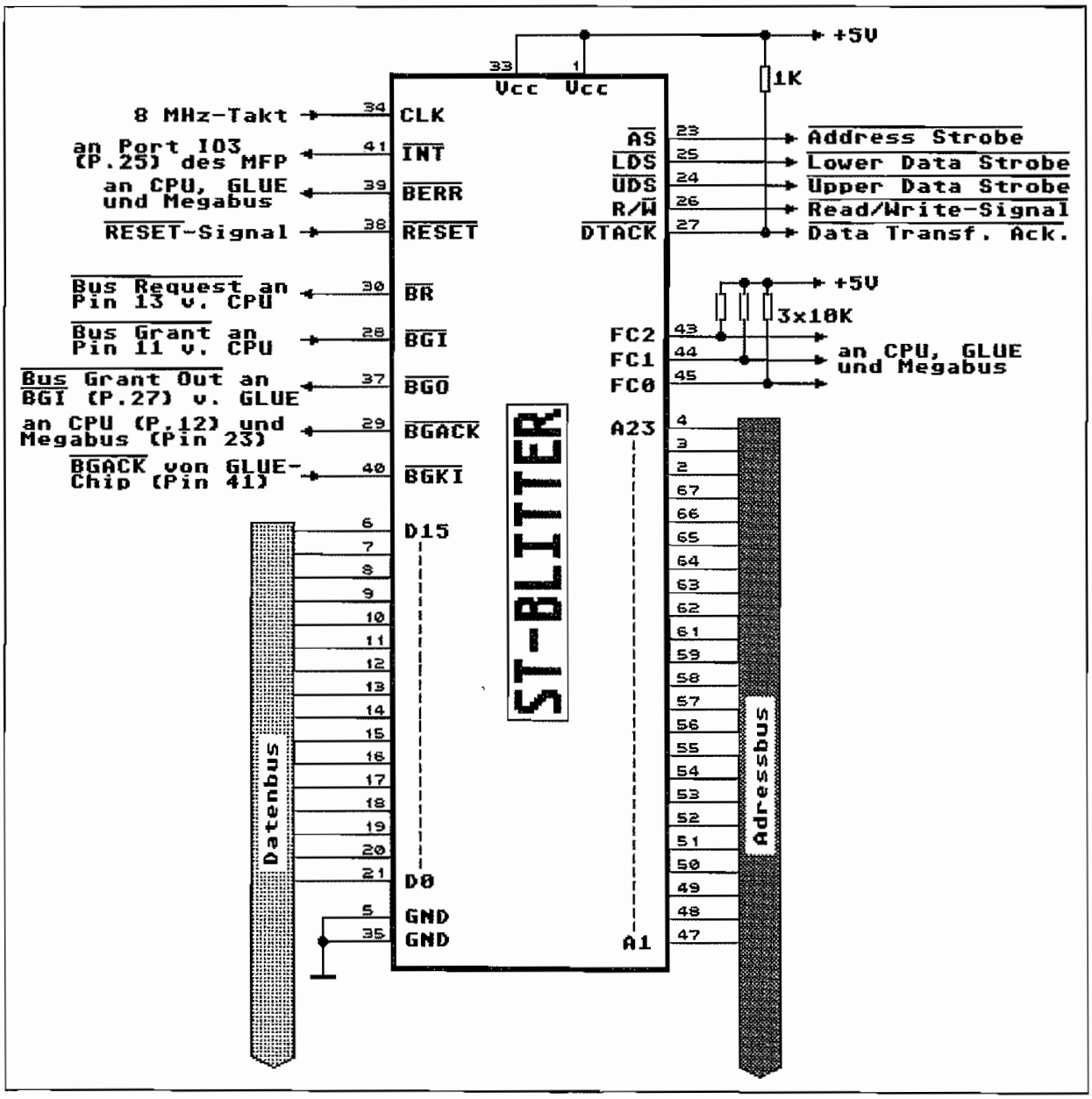


Abb. 2.17: Die Einbindung des BLITTERs in die ST-Hardware

prozessor, der sich in einem 68000er-System "zu Hause" fühlt. Als eigenständiges DMA-Device kann er deshalb in einem 68000er-System die Bussteuerung übernehmen.

Die Taktversorgung erfolgt ebenfalls mit dem gleichen 8-MHz-Takt, wie ihn die CPU des ST erhält.

Auf die einzelnen Steuer- und Meldeleitungen im einzelnen einzugehen,würde den Rahmen sprengen. Außerdem haben diese im Prinzip die gleiche Funktion wie bei einer 68000er-CPU (ein paar Informationen finden sich auch in Teil II, Kapitel 1, im Abschnitt über den Systembus des MEGA ST).

Die Anschlüsse BGI und BGO dienen der Verkettung des Blitters mit den anderen DMA-Einheiten des ST. Das Signal BG (Bus Grant = Bus Freigabesignal) gelangt nämlich von der CPU an BGI (Bus Grant In) des Blitters und von dessen BGO-Anschluß (Bus Grant Out) wieder hinaus an BGI des GLUE-Chips (der einen Großteil der DMA-Logik des ST enthält). Ebenfalls wird das BGACK-Signal vom GLUE erst durch BGKI - BGACK des Blitters "gefädelt", bevor es dann als BGACK-Signal an die CPU gelangt. Damit bekommt der Blitter immer als erster mit, wenn die DMA-Einheit des ST den Bus beansprucht.