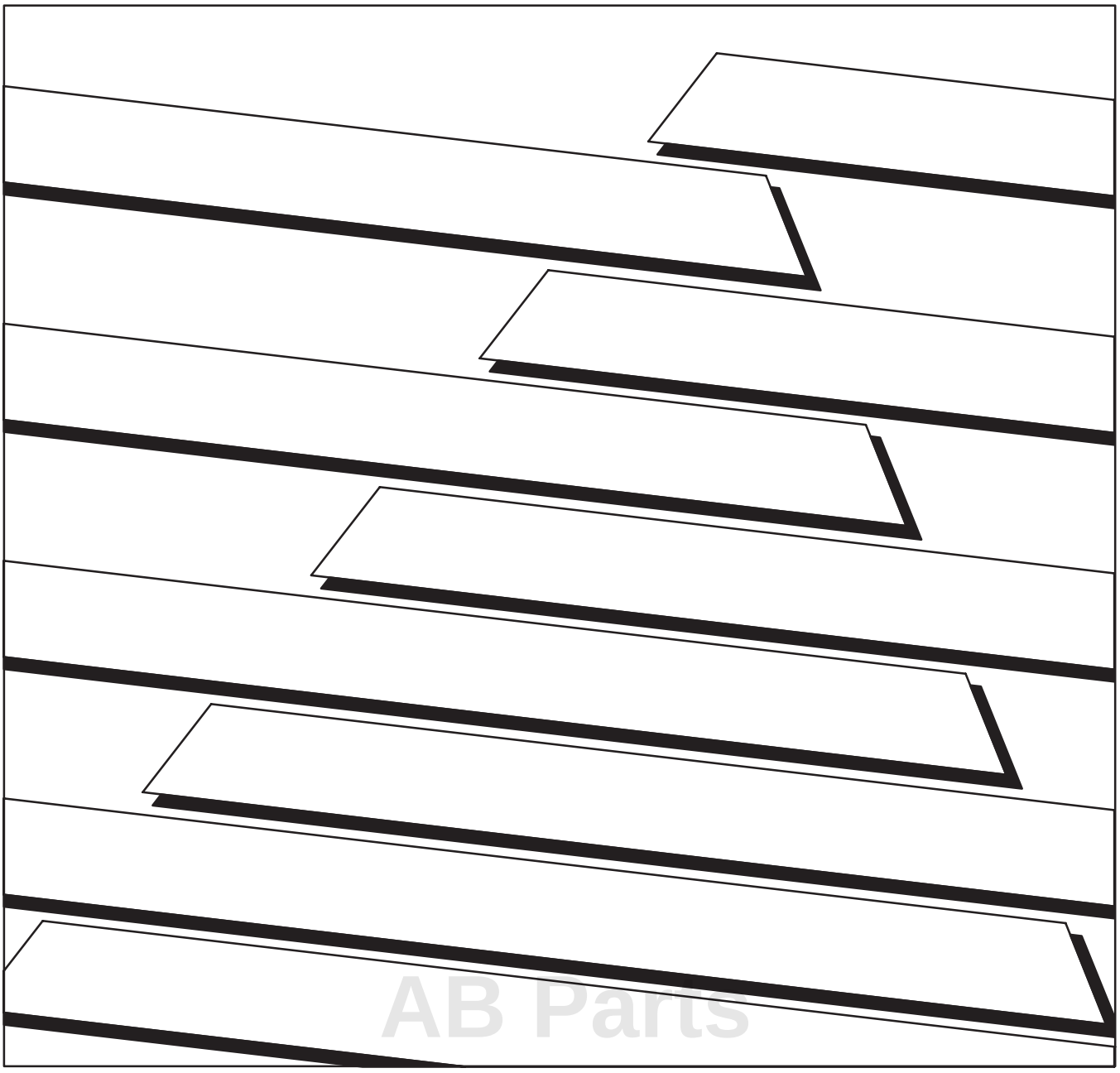
ALLEN-BRADLEY

Scanner 1784-KT*x*

(Bestell-Nr. 1784-KTX, -KTXD und -KTS)

Bedienerhandbuch



Wichtige Anwendungshinweise

Aufgrund der vielfältigen Einsatzmöglichkeiten der in dieser Publikation beschriebenen Produkte müssen Sie als Verantwortlicher für die Anwendung und den Einsatz dieser Steuerung sicherstellen, daß jede Anwendung bzw. jeder Einsatz alle Leistungs- und Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Die in diesem Handbuch dargestellten Abbildungen, Tabellen, Programm- und Layout-Beispiele sind ausschließlich zur besseren Texterläuterung dieses Handbuchs aufgeführt. Aufgrund der vielfachen Möglichkeiten und Anforderungen jedes einzelnen Verwendungszwecks kann Allen-Bradley keine Verantwortung oder Haftung (einschließlich Haftung für geistiges Eigentum) für tatsächliche Einsätze, die auf in dieser Publikation beschriebenen Beispielen beruhen, übernehmen.

Die Allen-Bradley Publikation SGI-1.1, "*Safety Guidelines For The Application, Installation and Maintenance of Solid State Control*" (erhältlich über Ihre örtliche Allen-Bradley Geschäftsstelle) behandelt einige wichtige Unterschiede zwischen elektronischen und elektromechanischen Geräten, die bei der Anwendung der in dieser Publikation beschriebenen Produkte berücksichtigt werden sollten.

Besondere Hinweise in diesem Handbuch sollen den Anwender auf mögliche Verletzungen oder Geräteschäden unter bestimmten Umständen aufmerksam machen.

ATTENTION: Weist auf Informationen über Verfahrensweisen oder Umstände hin, die zu Körperverletzungen oder Tod, Sachschaden oder wirtschaftlichem Verlust führen können.

Achtungshinweise helfen Ihnen:

- eine Gefahr festzustellen.
- die Gefahr zu vermeiden.
- die Konsequenzen zu erkennen.

Important: Weist auf Informationen hin, die äußerst wichtig für die erfolgreiche Anwendung und für die Vertrautheit mit dem Produkt sind.

Hinweise zu diesem Handbuch

Inhalt dieses Handbuchs

Verwenden Sie dieses Dokument, um die Scannerfunktion der Kommunikationsschnittstellenkarten 1784-KTX, 1784-KTXD und 1784-KTS zu verstehen, zu programmieren und einzusetzen.

In diesem Dokument werden die Karten 1784-KTX, 1784-KTXD und 1784-KTS als “1784-KT*x*” oder “Scanner” bezeichnet.

Leserkreis

Um dieses Produkt effektiv zu verwenden, müssen Sie Kenntnisse in den folgenden Bereichen aufweisen:

- C-Programmierung
- speicherprogrammierbare Steuerungen von Allen-Bradley

Dieses Dokument wurde für **Anwendungsprogrammierer** verfaßt.

Konventionen

In diesem Handbuch werden die folgenden Konventionen verwendet:

- Befehlsnamen sehen wie folgt aus: Set Mode
- Variablen und Funktionsnamen sehen wie folgt aus: get_scan_list
- Programmkonstanten und Makros sehen wie folgt aus: SUCCESS oder INIT_IN_PROGRESS

Hinweise zu den Beispielen

Die Beispiele in diesem Handbuch sollen den Anwendungsprogrammierer in die Host/KT*x*-Schnittstelle und -Befehle einführen. Die Beispiele demonstrieren alle Host/KT*x*-Befehle sowie andere häufig auszuführende Aufgaben, wie z.B.:

- Lesen von und Schreiben in E/A-Datentafeln
- Lesen der Statustabelle
- Ein- und Abschalten
- Watchdog-Zeitwerk

Im Gegensatz zu einem typischen Anwendungsprogramm sollen diese Programmbeispiele nicht Nutzen aus der Fähigkeit der KT*x*-Karte, den Host zu unterbrechen, wenn ein Host/KT*x*-Befehl nahezu abgeschlossen ist, ziehen. Einigen Befehlen, wie z.B. Blocktransfers, gibt die KT*x*-Karte 4 Sekunden zur Ausführung ihrer Aufgaben, bevor sie einen Timeout-Fehler erklärt. Bei den meisten Anwendungen wäre es unpraktisch, die Verarbeitung so lange zu unterbrechen.

Ein Quellcode für eine Interrupt-Bearbeitungsroutine ist enthalten, um zu demonstrieren, wie eine Anwendung KT*x*-Interrupts bearbeiten würde. Die Interrupt-Bearbeitungsroutine wurde nicht mit dem Rest des Quellcode-Beispiels integriert. Diese Aufgabe ist dem Anwendungsprogrammierer überlassen.

Zugehörige Publikationen

Titel der Publikation	Pub.-Nr.
1784-KT*x* Dual-port Interface Specification Reference Manual	1784-6.5.21
1784-KT*x* Communication Interface Card User Manual	1784-6.5.22DE
1784-CP12 Cable Packing Data	1784-2.41
1784-CP13 Cable Packing Data	1784-2.44
I/O Concepts Manual for Industry Bus Products	6008–6.5.1
Installationshandbuch für das Data Highway/Data Highway Plus/ Data Highway II/Data Highway-485 Kabel	1770-6.2.2DE

Inhaltsverzeichnis
Bedienerhandbuch für den
Scanner 1784-KTx

Erläuterung der Konzepte eines E/A-Scanners

Kapitel 1

Ein- und Abschalten

Kapitel 2

Programmierüberblick

Kapitel 3

Erteilen von Scanner-Verwaltungsbefehlen

Kapitel 4

Kapitel 1

Erläuterung der Konzepte eines E/A-Scanners

Kapitelinhalt

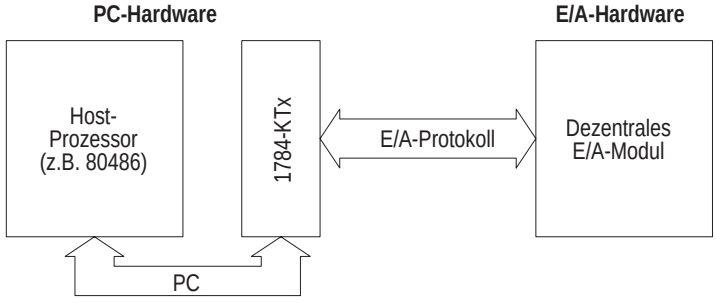
Dieses Kapitel erläutert grundlegende Konzepte und bietet einen Überblick über die Funktionsweise des PC-E/A-Scanners. Nach dem Lesen des Kapitels sollten Sie verstehen, wie:

- Informationen zwischen dem Programm und der Peripherie übertragen werden
- das Programm Befehle erteilen kann, um den Betrieb des Scanners zu beeinflussen

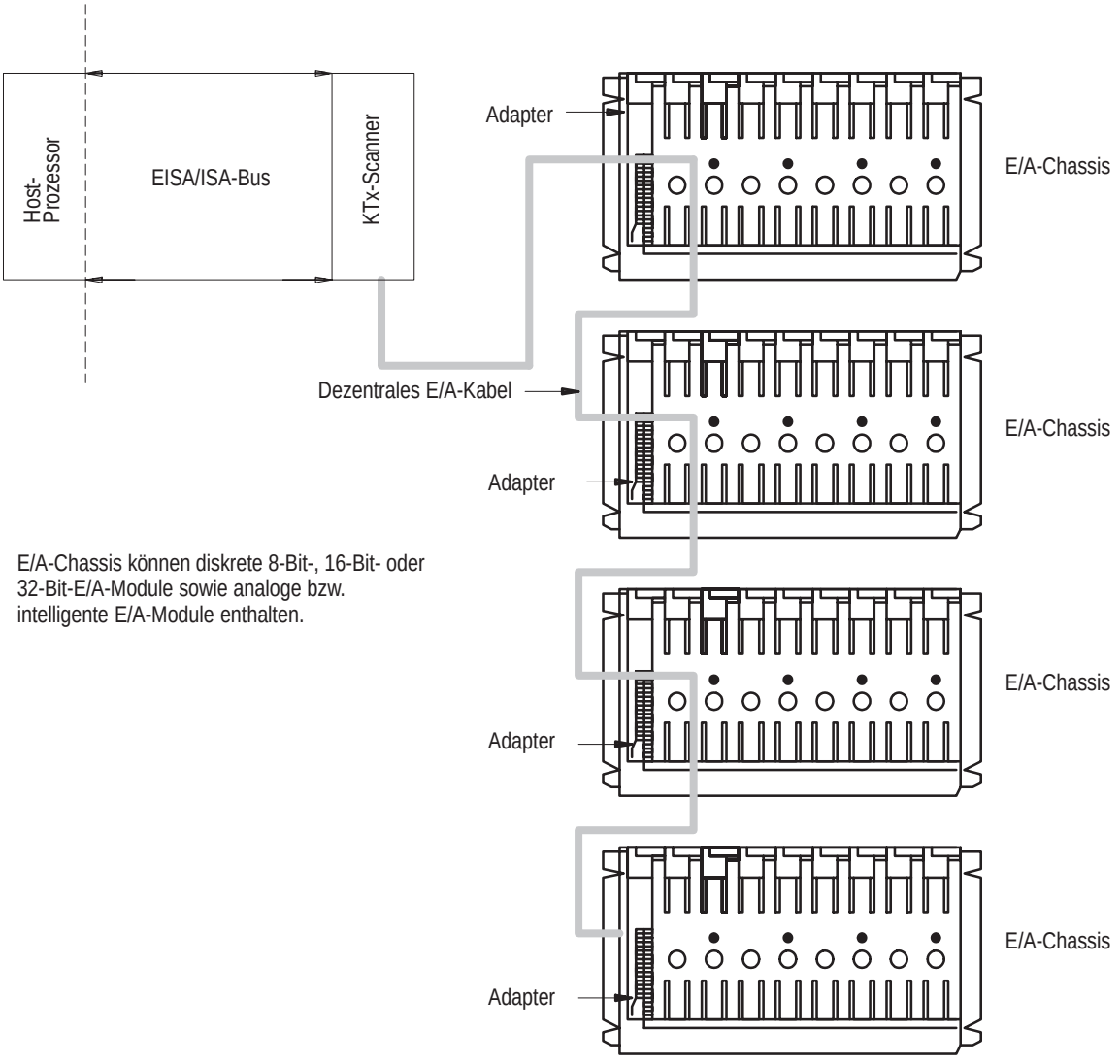
Verhältnis von Scanner zu E/A

Der Scanner verwendet das dezentrale E/A-Protokoll zur Kommunikation mit E/A-Modulen von Allen-Bradley. Sie brauchen mit den Einzelheiten des Protokolls nicht vertraut zu sein, um den Scanner zusammen mit den E/A-Modulen einzusetzen, müssen jedoch einige Begriffe kennen.

Der Scanner ist eine Kommunikationskarte von Allen-Bradley, die eine in einem kompatiblen Host-Computer (oder Host) zu installierende 16-Bit-Backplane erfordert.



E/A-Module befinden sich in einem bzw. mehreren Chassis. Ein E/A-Chassis ist ein Gehäuse, das einen Adapter und bis zu 16 E/A-Module enthält. Der Adapter ist die Kommunikationsschnittstelle zwischen Scanner und Chassis. Der Scanner kommuniziert über ein von Allen-Bradley zugelassenes Kabel mit dem Adapter. Der Adapter wiederum überwacht und steuert die E/A-Module über die Backplane des Chassis. Sie können Chassis beliebig kombinieren und somit bis zu 32 Adapteradressen verwenden.



Wir können E/A und somit E/A-Module in diskrete und intelligente Module unterteilen.

Diskrete E/A zeichnen sich durch einen Anschluß (bzw. einen Punkt) je E/A-Datentafelbit aus. Das Programm bearbeitet diskrete E/A über E/A-Datentafeln, wobei jeder Eingangs- bzw. Ausgangsanschluß einem der 4096 Eingangs- und 4096 Ausgangsdatentafelbits (256 x 16 Bits = 4096 Bits) entspricht.

Die Eingangsdatentafel ist ein Bereich des Speichers, der die Anschlüsse diskreter Eingangsmodule überwacht. Wird ein Eingangsschalter geschlossen, wird das entsprechende Bit auf 1 gesetzt. Die Ausgangsdatentafel ist ein Bereich des Speichers, der die Ausgangsanschlüsse der Ausgangsmodule überwacht. Nach dem Setzen eines Bit auf 1 wird der entsprechende Schalter geschlossen bzw. der Anschluß eingeschaltet.

Ein Modul mit normaler Schreibdichte ist ein diskretes Eingangs- bzw. Ausgangsmodul, das 4, 6 oder in der Regel 8 Eingangs- bzw. Ausgangsanschlüsse besitzt. Ein Modul mit hoher Schreibdichte ist ein diskretes Eingangs- bzw. Ausgangsmodul, das 16 Eingangs- bzw. Ausgangsanschlüsse besitzt. Ein Modul mit vierfacher Schreibdichte ist ein diskretes Eingangs- bzw. Ausgangsmodul, das 32 Eingangs- bzw. Ausgangsanschlüsse besitzt.

Intelligente E/A zeichnen sich durch die Übertragung eines bzw. mehrerer 16-Bit-Worte in einem bestimmten Format an ein bzw. aus einem E/A-Modul aus. Ein Blocktransfer (BT) ist die Übertragung von Daten an ein bzw. aus einem intelligenten E/A-Modul.
Bei einem BT-Lesevorgang werden Informationen (normalerweise analoge Eingangs- und Statusdaten) aus dem Modul an den Host übertragen; bei einem BT-Schreibvorgang werden Daten (normalerweise analoge Ausgangs- und Konfigurationsdaten) aus dem Host an das Modul übertragen.

E/A-Adressierung

Jedem Adapter wird durch Einstellen der Schalter am Adapter eine E/A-Racknummer (0 – 31) zugeordnet. Ein logisches Rack kann ein einzelnes Chassis sein, oder eine Racknummer kann sich aus zwei bis vier Racks zusammensetzen, oder ein einzelnes Chassis kann als zwei Racks adressiert sein. Racknummern müssen nicht sequentiell zugeordnet werden: Sie könnten z.B. ein volles Rack 0, ein halbes Rack in Rack 3 und ein viertel Rack in Gruppe 6-7 des Racks 7 haben.

Zu Adressierungszwecken entspricht jedes Rack einem Block von 8 E/A-Gruppen in der E/A-Datentafel. Gruppen innerhalb eines Racks sind mit 0 bis 7 numeriert. Eine E/A-Gruppe besteht aus zwei 16-Bit-Worten mit derselben Adresse, eines aus der Ausgangsdatentafel und eines aus der Eingangsdatentafel. Weitere Informationen finden Sie im “I/O Concepts Manual”, Publikation 6008-6.5.1. Bei den meisten Anwendungen wird für eine bestimmte E/A-Gruppe nur das Eingangswort bzw. nur das Ausgangswort verwendet.

Es folgt ein Beispiel für das Layout der Ausgangsdatentafel, das die ersten 8 Racks zeigt:

Wortnr. (hex)	Rack	Gruppe							
		0	1	2	3	4	5	6	7
00–07	0	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
08–0F	1	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
10–17	2	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
18–1F	3	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
20–27	4	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
28–2F	5	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
30–37	6	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
38–3F	7	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx

Die obigen Wortnummern können als Tiefzahlen verwendet werden (nähere Einzelheiten hierzu finden Sie in Kapitel 6, Diskrete E/A). Jedes 16-Bit-Wort entspricht 16 diskreten E/A-Anschlußpositionen, Anschluß 17 oktal (15 dezimal) bis zum höherwertigen Bit und Anschluß 00 bis zum niederwertigen Bit.

Genauso wie jede E/A-Gruppe besitzt auch jeder Adapter eine Adresse. Adapteradressen werden in der Abfrageliste (siehe Seite 1-6, Funktionsweise des Scanners) verwendet. Die Adapteradresse ist die Adresse der ersten vom Adapter abgedeckten E/A-Gruppe, dividiert durch 2. Dies entspricht auf numerischer Ebene (Rack x 4) + (beginnende Gruppe / 2), wobei Rack und Gruppe zwischen 0 und 7 numeriert sind (siehe oben). Sie können 1/4 Racks auch als zwischen 0 und 3 numeriert ansehen, wobei die Adapteradresse dann (Rack x 4) + (Viertel) ist.

Ein Steckplatz ist eine Position in einem E/A-Chassis für ein E/A-Modul. Bei der 1-Slot-Adressierung stellt eine E/A-Gruppe einen einzelnen Steckplatz dar. Bei der 2-Slot-Adressierung stellt eine E/A-Gruppe zwei Steckplätze dar.

Logisches Rack / Beginnendes Viertel und Verbundadressen

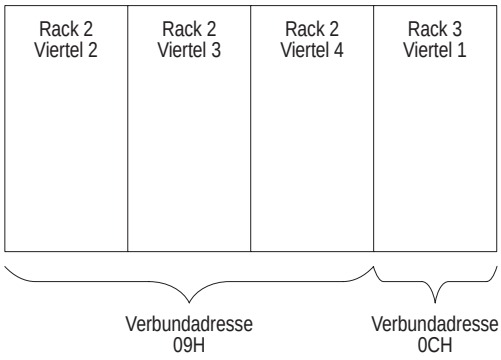
Adapter-Verbundadressen setzen sich aus einer Racknummer und einem beginnenden Viertel für ein bestimmtes Rack zusammen. Die zwei niederwertigen Bits der Adresse sind das beginnende Viertel des Racks, und die nächsten 5 Bits sind die Rackadresse. Das höherwertige Bit ist bei der Verwendung der KT*x*-Karte stets Null.

Beispiel: Rack 7, beginnendes Viertel 3 hat eine Verbundadresse von 00011111binär oder 1F hexadezimal. Bei binärer Darstellung setzt sich die Adresse folgendermaßen zusammen: 0 – 00111 – 11. Das höherwertige Bit ist 0, die Rackadresse ist 00111, und das beginnende Viertel ist 11.

Sehen Sie sich hierzu die Beispiele auf Seite 1-5 an.

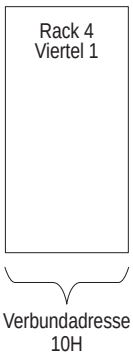
Das folgende physikalische Rack besitzt 2 Verbundadressen und enthält 3/4 des logischen Racks 2 und 1/4 des logischen Racks 3.

Das volle Rack beginnt bei der "ersten E/A-Gruppe" 1 (zweites Viertel) der Racknummer 2, wobei 2-Slot-Adressierung verwendet wird.

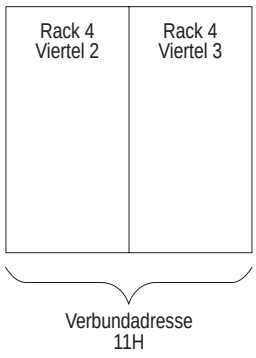


Im folgenden Beispiel machen 2 physikalische Racks, von denen jedes eine Verbundadresse besitzt, 3/4 des logischen Racks 4 aus.

Das viertel Rack beginnt bei der "ersten E/A-Gruppe" 0 (erstes Viertel) der Racknummer 4, wobei 2-Slot-Adressierung verwendet wird.

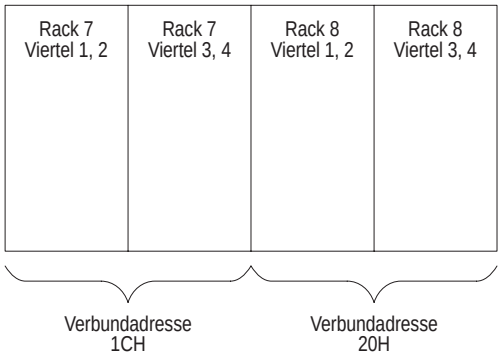


Das halbe Rack beginnt bei der "ersten E/A-Gruppe" 1 (zweites Viertel) der Racknummer 4.



Das folgende physikalische Rack besitzt 2 Verbundadressen und enthält 2 volle logische Racks (Rack 7 und 8).

Das volle Rack beginnt bei der "ersten E/A-Gruppe" 0 (erstes Viertel) der Racknummer 7, wobei 1-Slot-Adressierung verwendet wird.



Umwandlung von Adressen

Sie können anhand der folgenden Makros Verbundadressen schnell in logische Adressen und umgekehrt umwandeln. Die Klammern sind notwendig, um zu gewährleisten, daß die Operationen in der richtigen Reihenfolge ausgeführt werden.

Umwandlung von Verbundadressen in logische Adressen
Zum Umwandeln einer Verbundadresse in eine logische Adresse verschieben Sie diese zweimal nach rechts.

#define LINK_TO_LOGICAL(link,logical) ((logical)=((link)>>2))

Umwandlung von logischen Adressen in Verbundadressen
Zum Umwandeln einer logischen Adresse in eine Verbundadresse verschieben Sie diese zweimal nach rechts und fügen das beginnende Viertel hinzu.

#define LOGICAL_TO_LINK(link,logical,sq)
((link)=(((logical)<<2)+sq))

Adressenbereich
Der Scanner 1784-KT*x* kann bis zu 32 logische Racks (32 physikalische Netzknotenadressen) mit maximal 4096 diskreten Eingängen und 4096 diskreten Ausgängen bearbeiten.

Funktionsweise des Scanners

Der Scannerbetrieb verläuft asynchron zum Betrieb des Hosts (Personal Computers). Wenn der Scanner die Aufmerksamkeit des Hosts erregen will oder umgekehrt, muß der Scanner bzw. der Host einen Hardware-Interrupt erteilen. Informationen werden über einen Dual-Port-RAM ausgetauscht. Sowohl der Host als auch der Scanner kann die Bearbeitung eines Interrupts aufschieben, wenn gerade eine andere interruptgesteuerte Aufgabe ausgeführt wird.

Der Scanner verwaltet eine Abfrageliste. Es handelt sich dabei um eine Liste der vom Scanner zu bearbeitenden Adapter. Ein bestimmter Adapter kann einmal, mehrere Male oder überhaupt nicht in der Liste auftreten. Die Abfrageliste ist leer, bis einer der folgenden Scannerbefehle ausgeführt wird: Autoconfigure oder Set Scan List.

Ein Austausch (oder eine Adapterabfrage) ist der Informationsaustausch des Scanners mit einem Adapter. Während eines Austauschs kann der Scanner Daten oder Statusinformationen aus dem Adapter empfangen bzw. Daten oder Befehle an den Adapter senden bzw. beides. Sowohl Blocktransfers als auch diskrete E/A-Transfers können während desselben Austauschs durchgeführt werden, falls das Chassis des Adapters beide Arten von E/A-Modulen enthält.

Der Host erteilt Befehle, indem er an den Dual-Port schreibt. Der Scanner bearbeitet Befehle bei ihrer Ankunft. Wenn es der aktuelle Betriebsmodus zuläßt, führt der Scanner den Befehl aus. Besitzt der Scanner eine Bestätigung dieses Befehls bzw. eines zuvor ausgeführten Befehls, legt er die Bestätigung im Dual-Port-RAM ab und unterbricht den Host.

Die Abfrageliste ist kreisförmig: jedesmal, wenn der Scanner das Ende der Abfrageliste erreicht, beginnt er wieder von vorne. Im Laufe einer E/A-Abfrage (die manchmal auch nur als Abfrage bezeichnet wird) durchläuft der Scanner die Abfrageliste einmal vollständig, d. h. von Punkt A bis Punkt A. Sie können somit versichert sein, daß die E/A-Datentafeln einmal pro Zyklus durch die Abfrageliste aufgefrischt werden.

Betriebsmodi

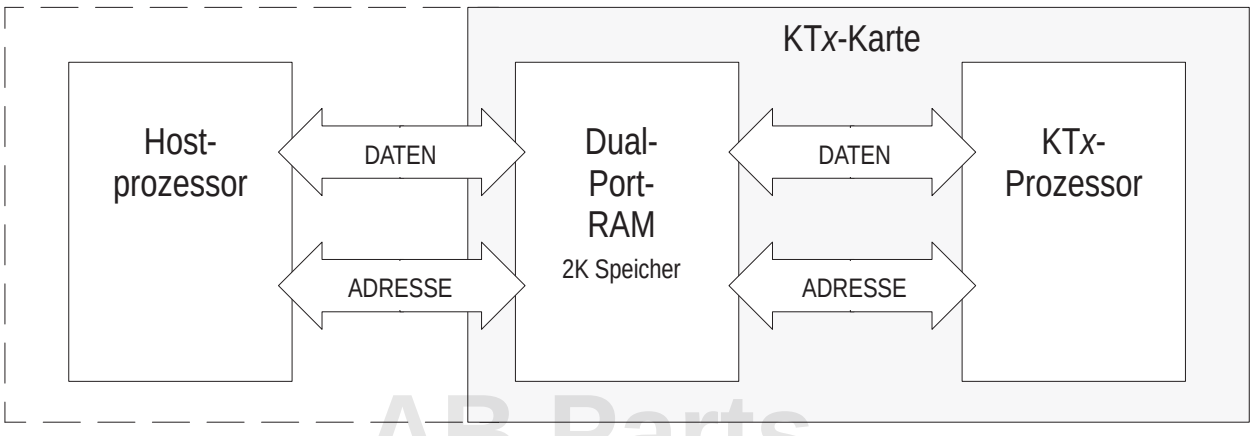
Der Scanner besitzt drei Betriebsmodi: Program-, Test- und Run-Modus. Diskrete Eingänge werden in allen drei Modi gelesen.

- Im *Program*-Modus werden keine diskreten Ausgänge an die Adapter gesandt, und die Adapter werden angewiesen, alle diskreten Ausgänge zurückgesetzt (Null) zu halten. Der Scanner behält alle Blocktransferanforderungen in seiner Warteschlange, ohne diese zu bearbeiten.
- Im *Test*-Modus werden die Adapter immer noch angewiesen, die diskreten Ausgänge zurückgesetzt zu halten; der Scanner sendet jedoch diskrete Informationen an diese. Blocktransfers können im Test-Modus stattfinden, ihre Ausgänge bleiben jedoch zurückgesetzt.
- Im *Run*-Modus werden diskrete Ausgangsinformationen an die Adapter gesandt, und die Adapter dürfen die Ausgangsmodule aktualisieren. Blocktransfers können ausgeführt werden.

Es kommt zu keiner Verbundaktivität, bis Sie einen Befehl Autoconfigure oder Set Scan List erteilen. Wenn das Programm den Scannerbetrieb beginnt, befindet sich der Scanner im Program-Modus. Das Programm muß einen Befehl zum Wechseln in den Run-Modus erteilen.

Dual-Port-RAM

Der Dual-Port-RAM bietet Ihnen jederzeit nahezu 100% Zugriff auf die Schnittstelle, was einen schnelleren Ablauf der gesamten Operation gewährleistet. Der Zugriff wird Ihnen nur dann verweigert, wenn der Host und der KTx-Prozessor zur gleichen Zeit auf dieselbe Speicherposition zuzugreifen versuchen.



Der KT*x*-Scanner besitzt integrierte Kollisionserkennung; ein Zuteiler reguliert den Verkehr im Dual-Port-RAM. Wenn der KT*x*-Prozessor und der Host-Prozessor versuchen, gleichzeitig auf dieselbe Speicherposition im Dual-Port-RAM zuzugreifen, wird dem Prozessor, der die Anforderung zuerst gestellt hat, Zugriff gewährt.

Aufgabenbereich des Hosts

Der Host hat zwei primäre Aufgabenbereiche; er:

- ändert und überwacht den Fluß diskreter Daten
- erteilt Scannerbefehle

Datenfluß und Datenpfade

Im nachstehenden Abschnitt wird der Pfad beschrieben, der von einem diskreten Eingangsbit befolgt wird:

- Ein externes Gerät bewirkt, daß ein Eingangsanschluß eines diskreten Eingangsmoduls eingeschaltet wird.
- Bei der nächsten Abfrage durch den Adapter berichtet das Eingangsmodul die neuen Eingangsinformationen. Der Adapter aktualisiert seine interne Eingangsdatentafel, indem er das dem jeweiligen Eingangspunkt entsprechende Bit setzt.
- Bei der nächsten Abfrage durch den Scanner berichtet der Adapter die neuen Eingangsinformationen. Der Scanner aktualisiert die Eingangsdatentafel im Dual-Port-RAM, indem er das dem jeweiligen Eingangspunkt entsprechende Bit setzt.
- Der Host kann den Dual-Port (diskrete Daten) jederzeit lesen.

Der Pfad eines Ausgangsbits ist im Grunde die Umkehrung des Eingangspfads:

- Das Programm setzt ein Bit in seiner Ausgangsdatentafel. Das Programm weiß, daß dieses Bit einem Ausgangsanschluß auf einem bestimmten Ausgangsmodul entspricht.
- Wenn der Scanner den Adapter, der das jeweilige Ausgangsmodul steuert, das nächste Mal abfragt, weist er den Adapter an, seine Ausgangsdatentafel mit den neuen Informationen zu aktualisieren.
- Der Adapter weist das diskrete Ausgangsmodul an, seine Ausgänge mit den neuen Informationen zu aktualisieren.
- Das diskrete Ausgangsmodul schaltet den Ausgang ein. Das an das Ausgangsmodul angeschlossene externe Gerät wird daraufhin aktiv.

Informationen zum Zeitverhalten finden Sie unter “Zeitverhalten diskreter E/A” in Kapitel 7, “Erläuterung diskreter E/A”.

Scannerbefehle

Im Rahmen seines zweiten primären Aufgabenbereichs erteilt der Host die folgenden Arten von Scannerbefehlen:

- Verwaltungsbefehle
- Blocktransferbefehle

Ein Verwaltungsbefehl beeinträchtigt den Betrieb des Scanners selbst. Es stehen fünf Verwaltungsbefehle zur Verfügung:

Befehl	Aktion
Set Mode	Ändert den Betriebsmodus des Scanners auf Program-, Test- oder Run-Modus
Autoconfigure	Greift auf den Verbund zu, um festzustellen, welche Geräte angeschlossen sind
Set Scan List	Ändert die Reihenfolge, in der Adapter abgefragt werden, sowie die relative Häufigkeit, mit der Adapter abgefragt werden
Get Scan List	Beschafft eine Kopie der aktuellen Abfrageliste
Set Fault Dependent Group	Kennzeichnet eine oder mehrere Gruppen von Adaptern so, daß wenn ein Adapter in der Gruppe fehlerhaft ist, der Scanner die anderen Adapter in der Gruppe anweist, ebenfalls fehlerhaft zu sein

Die Befehls-/Bestätigungspuffer werden zum Übertragen von Scannerverwaltungsbefehlen an den Scanner und zum Weitergeben von Statusinformationen an den Host verwendet. Darüber hinaus wird dieser Bereich des Dual-Port-RAM für Blocktransfers zwischen dem Host und intelligenten E/A-Geräten im E/A-System eingesetzt.

Informationen über Verwaltungsbefehle sind Kapitel 4 zu entnehmen.

Es stehen zwei Blocktransfer (BT)-Befehle zur Verfügung:

Befehl	Aktion
Host BT Read	Liest Daten aus einem intelligenten E/A-Modul und leitet Daten an den Host zurück
Host BT Write	Schreibt vom Host bereitgestellte Daten in ein intelligentes E/A-Modul

Informationen über Blocktransfers finden Sie in Kapitel 5.

Nächster Schritt

Kapitel 2 beschreibt Verfahrensweisen für das Ein- und Abschalten des Scanners.

Kapitel 2

Ein- und Abschalten

Kapitelinhalt

Dieses Kapitel beschreibt die Ein- und Abschaltphasen des Betriebs, einschließlich:

- Verwenden der im Speicher abgebildeten KTx-Hardware
- Ausführen des Diagnoseprogramms auf der KTx-Karte
- Ausführen des Host-Kompatibilitätstests (M16-Diagnosetest)
- Laden des Protokollfiles
- Initialisieren
- Abschalten

Verwenden der im Speicher abgebildeten KTx-Hardware

Bevor Sie mit dem Aufbau des Treibers beginnen, müssen Sie mit der im Speicher abgebildeten Hardware der KTx-Karte vertraut sein. Die folgenden Abschnitte beschreiben im Speicher abgebildete Hardware-Bytes :0800h bis :080Fh, die für Ihren Treiber erforderlich sein können.

Z80 aktivieren (freigeben) (Byte :0802h)

Schreiben Sie 01h in diese Adresse, um den Z80 zu aktivieren (freizugeben) und mit der Ausführung des geladenen Binärfiles zu beginnen. Nachfolgende Schreibvorgänge an diese Adresse bei aktiviertem Z80 haben keinerlei Auswirkungen.

Z80 deaktivieren (zurücksetzen) (Byte :0803h)

Schreiben Sie 01h in diese Adresse, um den Z80 zu deaktivieren (zurückzusetzen) und die Ausführung des geladenen Binärfiles zu unterbrechen. Nachfolgende Schreibvorgänge an diese Adresse bei deaktiviertem Z80 haben keinerlei Auswirkungen.

KT*x*-Statusregister (Byte :0804h)

Wichtig: Dieses Byte kann nur gelesen werden.

Dieses Byte berichtet den Status mehrerer Einstellungen der KT*x*-Karte. Die folgende Tabelle beschreibt den Inhalt jedes Bits.

Bit	Bedeutung des Statuscodes
7	0 = Z80 ist deaktiviert 1 = Z80 ist aktiviert
6	0 = Z80 befindet sich im Haltezustand 1 = Z80 befindet sich nicht im Haltezustand
5	0
4	0
3	0
2	0
1	0
0	0 = es ist gegenwärtig kein Hardware-Interrupt aktiviert 1 = ein Hardware-Interrupt ist gegenwärtig aktiviert (bleibt aktiv, bis der Host 0x1 in 808 schreibt)

Host-Interruptbestätigung (Byte :0808h)

Schreiben Sie 01h in diese Adresse, um einen Interrupt der KT*x*-Karte zu bestätigen. Es wird hierdurch Bit 0 des Statusregisters zurückgesetzt.

Kartensteuerungs-Schreibregister (Byte :080Ah)

Wichtig: Es kann in dieses Register nur geschrieben werden.

Bit	Bedeutung des Statuscodes
7	nicht belegt
6	
5	
4	
3	
2	
1	1 = versetzt die KT*x*-Karte in den E/ISA-16-Bit-Modus; funktioniert nicht mit 8-Bit-Geräten im selben 128K Speicherblock 0 = erweiterter 16-Bit-Modus; 8- und 16-Bit-Geräte können kombiniert werden; Vorgabewert = 0.
0	1 = Interrupts auf E/ISA-Ebene 0 = flankengesteuerte Interrupts; Vorgabewert = 0.

Kartensteuerungs-Leseregister (Byte :080Ch)

Wichtig: Dieses Register kann nur gelesen werden.

Bit	Bedeutung des Statuscodes
7	Zustand der LED 1 = ein 0 = aus
6	Farbe der LED 1 = rot 0 = grün
5	Blinkzustand der LED 1 = blinkt 0 = blinkt nicht
4	reserviert
3	Baudrate 00 = 230,4 kBaud 01 = 57,6 kBaud 10 = 115,2 kBaud 11 = 230,4 kBaud
2	
1	1 = versetzt die KT*x*-Karte in den E/ISA-16-Bit-Modus; funktioniert nicht mit 8-Bit-Geräten im selben 128K Speicherblock 0 = erweiterter 16-Bit-Modus; 8- und 16-Bit-Geräte können kombiniert werden; Vorgabewert = 0.
0	1 = Interrupts auf E/ISA-Ebene 0 = flankengesteuerte Interrupts; Vorgabewert = 0.

Ausführen des Diagnoseprogramms auf der KT*x*-Karte

Beim Kaltstarten des Host-PC wird auch die KT*x*-Karte neu gestartet und in einen zurückgesetzten Zustand geschaltet. Der Treiber kann jetzt die Einschalt-Selbstdiagnosetests vor dem Laden des Protokollbinärfiles in die KT*x*-Karte durchführen.

Wichtig: Da die Selbstdiagnosetests über den Dual-Port ausgeführt werden, müssen Sie die Funktionstüchtigkeit des Dual-Ports testen, ***bevor*** Sie die Selbstdiagnosetests ausführen.

Funktionstest des Dual-Ports

Zum Überprüfen der Funktionstüchtigkeit der Dual-Port-Schnittstelle beachten Sie die folgenden Schritte:

1. Löschen Sie den Dual-Port-Speicher, indem Sie **00h** in Dual-Port-Speicherpositionen :0000h bis :07FFh schreiben.

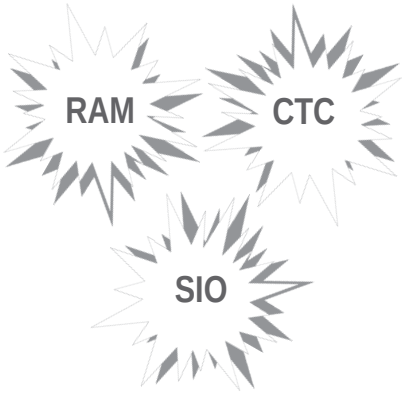
2. Lesen und überprüfen Sie Dual-Port-Speicherpositionen :0000h bis :07FFh auf ihre Korrektheit.

3. Schreiben Sie **FFh** in Dual-Port-Speicherpositionen :0000h bis :07FFh.

4. Lesen und überprüfen Sie Dual-Port-Speicherpositionen :0000h bis :07FFh auf ihre Korrektheit.

Drei Einschalt-Selbstdiagnosetests: RAM, CTC und SIO

Der Treiber muß die folgenden drei Einschalt-Selbstdiagnosetests (in der hier aufgelisteten Reihenfolge) ausführen:



1. **Direktzugriffsspeicher (RAM)-Test** — File KTXST0.BIN

   Dieser Test führt einen Musterschreibvorgang an den gesamten RAM-Speicher durch und überprüft diesen mit einem Lesezyklus. Die Testreihenfolge ist RAM1, RAM0, Dual-Port-RAM.

2. **Zähler-Zeitwerk-Stromkreis (CTC)-Test** — File KTXST1.BIN

   Dieses Programm führt einen Zeitverhaltenstest auf allen CTC-Kanälen der KT*x*-Karte durch.

3. **Serieller E/A (SIO)-Test** — File KTXST2.BIN

   Dieses Programm führt einen Rückkopplungstest auf den seriellen E/A der KT*x*-Karte durch.

Tabelle 2.A zeigt die zu diesen Tests gehörige Speicherbelegung des Diagnoseprogramms.

Tabelle 2.A
Speicherbelegung des Diagnoseprogramms im Dual-Port

Dual-Port-Adresse des Hosts	Inhalt des Dual-Ports
:0000h – :07FDh	KTXST0.BIN oder KTXST1.BIN oder KTXST2.BIN
:07FEh	Statuscode
:07FFh	Fertig

Die folgenden Bytes sind im Speicher abgebildete Hardware-Adressen:

:0800h	Reserviert
:0801h	KT*x* durch Schreiben einer 1 unterbrechen
:0802h	Z80 aktivieren (freigeben)
:0803h	Z80 deaktivieren (zurücksetzen)
:0804h	KT*x*-Statusregister
:0805h – :0807h	Reserviert
:0808h	KT*x*-Interrupt durch Schreiben einer 1 bestätigen
:080Ah	Kartensteuerungs-Schreibregister
:080Ch	Kartensteuerungs-Leseregister

Wichtig: Da die Selbstdiagnosetests über den Dual-Port ausgeführt werden, müssen Sie die Funktionstüchtigkeit des Dual-Ports testen, ***bevor*** Sie die Selbstdiagnosetests ausführen.

Zum Ausführen der drei Selbsttests wiederholen Sie die folgende Verfahrensweise für jeden Test:

Wichtig: Stellen Sie sicher, daß die KT*x*-Karte vor jedem Test zurückgesetzt wurde (schreiben Sie hierzu 01h in Byte :0803h).

1. Schreiben Sie **01h** in Byte :0803h, um die KT*x*-Karte ZURÜCKZUSETZEN.

2. Schreiben Sie **00h** in Byte :0000h bis :07FFh, um den Dual-Port-RAM zu LÖSCHEN.

3. Kopieren Sie den Binärfile des entsprechenden Tests in den Dual-Port. Beginnen Sie mit :0000h. Wählen Sie den richtigen Binärfile anhand der nachstehenden Tabelle aus.

Test	Binärfile
RAM	KTXST0.BIN
CTC	KTXST1.BIN
SIO	KTXST2.BIN

4. Schreiben Sie **01h** in Byte :0802h, um den Z80 FREIZUGEBEN.

5. Warten Sie, bis Byte :07FFh auf **01h** übergeht.

Wichtig: Erhält Byte :07FFh innerhalb von 5 Sekunden kein **01h**, ist der Test aus unbekannten Gründen fehlgeschlagen.

6. Lesen Sie den Testfehlerstatus in Byte :07FEh.

7. Schlagen Sie hierzu in der entsprechenden Statuscodetabelle weiter hinten in diesem Abschnitt nach.

8. Beheben Sie eventuell vorhandene Fehler.

Vergewissern Sie sich, daß die KT*x*-Karte ordnungsgemäß im PC-Steckplatz sitzt und die Konfiguration der KT*x*-Karte keine Konflikte mit anderen Adapterkarten im Host (wie z.B. anderen KT*x*-Karten, Speicherkarten und Video-Speicher) hervorruft.

Statuscodes für den RAM-Test sind Tabelle 2.B zu entnehmen.

Tabelle 2.B
Statuscodes für den RAM-Test

Statuscodes für den RAM-Test	Bedeutung des Codes
00h	Keine Fehler gefunden
02h	Störung, Z80 RAM 0
04h	Störung, Dual-Port-RAM
06h	Störung, RAM 0 und Dual-Port-RAM
80h	Störung, Z80 RAM 1
82h	Störung, RAM 0 und RAM 1
84h	Störung, RAM 1 und Dual-Port-RAM
86h	Störung, RAM 0, RAM 1 und Dual-Port-RAM

Statuscodes für den CTC-Test sind Tabelle 2.C zu entnehmen.

Tabelle 2.C
Statuscodes für den CTC-Test

Statuscodes für den CTC-Test	Bedeutung des Codes
00h	Keine Fehler gefunden
08h	Störung, CTC-Zeitwerkmodus
10h	Störung, CTC-Zählermodus
18h	Störung, CTC-Zeitwerk und -Zähler

Statuscodes für den SIO-Test entnehmen Sie Tabelle 2.D.

Tabelle 2.D
Statuscodes für den SIO-Test

Statuscodes für den SIO-Test	Bedeutung des Codes
00h	Keine Fehler gefunden
20h	Störung, SIO-Kanal: kein Interrupt
40h	Störung, SIO-Kanal A: Rückkopplungsstörung

Nachdem Sie das Diagnoseprogramm ausgeführt haben, müssen Sie den Host-Kompatibilitätstest durchführen. Anweisungen hierzu befinden sich im nächsten Abschnitt.

Ausführen des Host-Kompatibilitätstests

Allen-Bradley hat mehrere Maßnahmen ergriffen, um den ordnungsgemäßen Datentransfer zwischen einem IBM PC und der KT*x*-Karte über den ISA-Bus zu erleichtern. Wir haben einen Byte-Austauscher auf der anwendungsspezifischen integrierten Schaltung (ASIC) implementiert, der beide Byte-Pfade des ISA-Bus mit ungeradzahligen Bytedaten während eines ungeradzahligen Byte-Lesevorgangs des Dual-Ports steuert, d.h. SBHE=0, A0=1. M16 könnte somit bei ungeradzahligen Byte-Lesevorgängen in ‘aktuellen’ (ungefähr aus dem Jahre 1993 stammenden) Systemen problemlos übersehen werden. Umgekehrt steuern aktuelle Host-Systeme sowohl die höher- als auch die niederwertigen Datenpfade bei ungeradzahligen Byte-Schreibvorgängen, wobei der Host M16 nicht sieht.

Mit diesem Diagnosetest können Sie das System, in dem die KT*x*-Karte installiert ist, testen, um festzustellen, ob es als aktuelles System betrieben wird. Aktuelle Systeme steuern beide Byte-Pfade mit ungeradzahligen Bytedaten während eines ungeradzahligen Byte-Schreibvorgangs, wobei das System nicht sieht, ob die Slave-Karte M16 aktiviert hat.

Wichtig: Wird die Karte *nur* im 8-Bit-Modus ausgeführt, muß die M16-Diagnosefunktion nicht durchgeführt werden.

M16-Diagnosetest

Dieser Diagnosetest verwendet den erweiterten M16-Testmodus der KT*x*-ASIC. Dieser Testmodus ist nur dann betriebsfähig, wenn sich die KT*x*-Karte im erweiterten M16-Modus befindet. Der Testmodus schaltet den Betrieb der M16-Leitung aus, während die KT*x*-ASIC genauso betrieben wird, als ob M16 ordnungsgemäß aktiviert wäre. Es kann somit bestimmt werden, ob der höherwertige Byte-Pfad des PC bei einem ungeradzahligen Byte-Schreibvorgang gesteuert wird, wenn M16 nicht vom System gesehen wird. Wird der höherwertige Byte-Pfad nicht gesteuert, kann der PC nicht im erweiterten M16-Modus verwendet werden, und der Anwender muß den standardmäßigen M16- oder 8-Bit-Modus auswählen.

Wichtig: Der standardmäßige M16-Modus wird von keiner der aktuellen Anwendungsprogrammierschnittstellen (APIs) für die KT*x*-Karte verwendet. Dies bedeutet, daß keine aktuelle Host-Software die KT*x*-Karte vom erweiterten M16-Modus (Standardeinstellung) in den standardmäßigen M16-Modus umschaltet. Dieser Modus ist im Grunde ein Sicherheitsnetz für den Fall, daß ein Anwender angetroffen wird, der inkompatible Hardware besitzt oder nicht den 8-Bit-Modus verwendet.

Überprüfen des Modus

Bevor Sie das Diagnoseprogramm ausführen, stellen Sie sicher, daß die Karte im erweiterten M16-Modus betrieben wird.

1. Lesen Sie das Statusregister, Adresse 804, und überprüfen Sie, ob Bit 5 von Bits 0:7 auf 1 oder 0 gesetzt ist.

Zustand von Bit 5	Konfiguration der KTx-Karte	Erforderliche Maßnahmen
1	8-Bit-Modus 8/16-Bit-Betrieb ist mit dem 3-Stift-Kopfstück auf der KT*x*-Karte konfiguriert.	Informieren Sie den Anwender, daß seine KT*x*-Karte für den 8-Bit-Betrieb konfiguriert ist. Der Anwender muß die Karte für den 16-Bit-Betrieb konfigurieren, damit der Host-Kompatibilitätstest ausgeführt werden kann. **WICHTIG:** Lassen Sie den M16-Test aus, falls die Karte im 8-Bit-Modus belassen wird.
0	16-Bit-Modus	Fahren Sie mit Schritt 2 fort.

2. Lesen Sie das Kartensteuerungs-Leseregister, Adresse 80C, und überprüfen Sie, ob Bit 1 von Bits 0:7 auf 1 oder 0 gesetzt ist.

Zustand von Bit 1	Konfiguration der ASIC
1	Standardmäßiger M16-Betrieb; muß auf erweiterten M16-Modus umgeschaltet werden Die ASIC sollte sich nicht im Standardmodus befinden, da der M16-Standardmodus von den aktuellen APIs nicht implementiert wird (siehe Seite 2-7). Fahren Sie mit Schritt 3 fort.
0	Erweiterter M16-Betrieb Fahren Sie mit dem nächsten Abschnitt, “Deaktivieren des M16-Betriebs”, fort.

3. Schalten Sie die ASIC in den erweiterten M16-Modus um, indem Sie den Inhalt des Kartensteuerungs-Leseregisters sichern und in das Kartensteuerungs-Schreibregister, Adresse 80A, schreiben, wobei Bit 1 auf 0 gesetzt ist.

Die ASIC ist nun für den erweiterten M16-Betrieb konfiguriert.

Deaktivieren des M16-Betriebs

Schreiben Sie zum Deaktivieren des M16-Betriebs die Schlüsselbytes in die ASIC.

1. Überprüfen Sie die Funktionstüchtigkeit der Schlüsselbytes folgendermaßen:

- Schreiben Sie mit einem 0xBB in das Schreibregister für Schlüssel 0, Adresse 805
- Schreiben Sie mit einem 0xCC in das Schreibregister für Schlüssel 1, Adresse 807

2. Rufen Sie das Schlüssel-Leseregister, Adresse 800, mit einem 16-Bit-Lesevorgang auf.

Der zurückgeleitete Wert sollte 0xCCBB sein. Beachten Sie, daß das Schlüssel-Leseregister ein 16-Bit-Register ist und mit einem 16-Bit-Lesevorgang aufgerufen werden muß. Ein 8-Bit-Aufruf dieser Register leitet den Wert des aufgerufenen ISA-Bus bzw. 0xFF zurück.

Etablieren des Testmodus

1. Schreiben Sie mit einem 0xAA in Schlüssel 0, Adresse 805, und mit einem 0x55 in Schlüssel 1, Adresse 807.

2. Überprüfen Sie, ob der Schlüssel ordnungsgemäß geschrieben wurde, indem Sie einen 16-Bit-Lesevorgang für Adresse 800 durchführen.

Der zurückgeleitete Wert sollte 0xFFAA sein. Der Parameter 0xAA im Rückleitungswert ist korrekt, da das System einen 16-Bit-Lesevorgang einleitet und die Daten aus dem niederwertigen Byte-Pfad verwendet, wenn M16 nicht durch die ASIC aktiviert wird.

Das System kehrt dann mit einem 8-Bit-Lesevorgang des höherwertigen Bytes (es wird angenommen, die Karte wird im 8-Bit-Modus betrieben, da M16 nicht aktiviert war) zur Karte zurück, und die ASIC reagiert nicht auf den 8-Bit-Lesevorgang des höherwertigen Bytes des Schlüssels. Die zurückgeleiteten Daten für das ungeradzahlige Byte sind somit der aufgerufene ISA-Bus bzw. 0xFF.

Wird ein anderer Wert als 0xFF im höherwertigen Byte des Wortes zurückgeleitet, so wurde der Testmodus nicht ordnungsgemäß etabliert.

Ausführen des Diagnosetests

Überprüfen Sie, ob Daten ordnungsgemäß in den gesamten Dual-Port-Speicher geschrieben bzw. aus diesem gelesen werden können.

1. Füllen Sie den Dual-Port mit den folgenden Werten (ein Wert nach dem anderen), und überprüfen Sie, ob die korrekten Werte zurückgeleitet werden:

a. 0xFF

b. 0x00

c. 0x55

d. 0xAA

2. Beginnen Sie am Anfang des Dual-Ports (Basisadresse, Offset 0), und führen Sie 16-Bit-Schreibvorgänge des Wertes ‘count’ in den Speicher durch, wobei Sie ‘count’ und die Wortadresse erhöhen, bis der Dual-Port voll ist.
3. Überprüfen Sie, ob der Dual-Port Wortwerte zwischen 0x0000 und 0x03FF enthält.
4. Melden Sie dem Anwender, ob der Test erfolgreich war oder fehlgeschlagen ist.

Aktivieren des M16-Betriebs

Schreiben Sie die Schlüsselbytes zum Aktivieren des M16-Betriebs.

1. Überprüfen Sie die Funktionstüchtigkeit der Schlüsselbytes folgendermaßen:
   - Schreiben Sie mit einem 0x00H in das Schreibregister für Schlüssel 0, Adresse 805
   - Schreiben Sie mit einem 0x00H in das Schreibregister für Schlüssel 1, Adresse 807
2. Rufen Sie das Schlüssel-Leseregister, Adresse 800, mit einem 16-Bit-Lesevorgang auf.

   Der zurückgeleitete Wert sollte 0x0000 sein. Beachten Sie, daß das Schlüssel-Leseregister ein 16-Bit-Register ist und mit einem 16-Bit-Lesevorgang aufgerufen werden muß. Ein 8-Bit-Aufruf dieser Register leitet den Wert des aufgerufenen ISA-Bus bzw. 0xFF zurück.

Wichtig: Speichern Sie ggf. den Inhalt des Kartensteuerungs-Schreibregisters zurück. Dies sollte nicht erforderlich sein, da der M16-Standardmodus von den aktuellen APIs nicht implementiert wird.

Sie können nun den Protokollfile laden. Anweisungen hierzu finden Sie im nächsten Abschnitt.

Laden des Protokollfiles

Zum Herunterladen des Protokolls muß der Treiber:

- den Ladefile an den Dual-Port übertragen
- den Protokollfile laden

Übertragen des Ladefiles an den Dual-Port

Übertragen Sie zunächst den Ladebinärfile (LOADPCL.BIN) in den angegebenen Bereich des Dual-Port-RAM. Der Treiber verwendet das Ladeprogramm zum Herunterladen des Protokolls (der “weichen” Firmware) über die Dual-Port-Schnittstelle und in den RAM des Z80.

1. Löschen Sie den gesamten RAM-Speicher auf der KT*x*-Karte.

   a. Setzen Sie die KT*x*-Karte zurück, indem Sie 01h in Byte :803h schreiben.

   b. Kopieren Sie das Programm CLRRAM.BIN in den Dual-Port, beginnend mit :0000h.

   c. Schreiben Sie **01h** in Byte :0802h (zum Freigeben des Z80).

   d. Warten Sie zwei Sekunden lang, damit der Zyklus abgeschlossen werden kann.

   e. Setzen Sie die KTx-Karte zurück, indem Sie **01h** in Byte :0803h schreiben.

2. Übertragen Sie das Ladeprogramm (LOADPCL.BIN) an den Dual-Port, beginnend mit :0000h.

Laden Sie nun das Protokoll herunter.

Laden des Protokolls

Da das gesamte Protokoll zu lang ist, um vom 2K-Dual-Port-RAM in einer Operation verarbeitet zu werden, muß der Treiber den Protokollfile in mehreren, sich wiederholenden Abschnitten herunterladen.

Laden Sie das Protokoll anhand des nachstehenden Algorithmus in die KT*x*-Karte.

1. Laden Sie einen Block des Protokollcodes über den Dual-Port-RAM in den Z80-RAM. Beginnen Sie mit :0080h (gehen Sie nicht über :07FFh hinaus). Die maximale Byte-Größe beträgt :0780h (1920 dezimal).

2. Laden Sie die Blockgröße in die Speicherpositionen “Größenwort”. Das niederwertige Byte ist :0072h; das höherwertige Byte ist :0073h.

3. Geben Sie beim ersten Durchlauf den Z80 frei, indem Sie **01h** in Byte :0802h schreiben. Setzen Sie bei allen nachfolgenden Durchläufen das Handshake-Byte :0075h auf **00h** zurück, um dem Z80 der KT*x*-Karte zu signalisieren, daß der Treiber den nächsten Block in den Dual-Port geladen hat.

4. Warten Sie, bis das Ladeprogramm das Handshake-Byte :0075h auf **01h** zurücksetzt. Der Z80 lädt den Protokollcodeblock in seinen RAM-Speicher.

5. Überprüfen Sie Fehlerbyte :0074h.

Statuscode	Bedeutung des Codes
00h	Keine Fehler gefunden
01h	Block zu groß
02h	Z80-RAM zu voll für nächsten Block

6. Korrigieren Sie eventuell vorhandene Fehler, indem Sie das Problem beheben und anschließend den Z80 zurücksetzen und neu starten.

Hinweis: Möchten Sie wissen, wie viele Bytes Sie geladen hatten, berechnen Sie die Anzahl der bisher geladenen Bytes (Gesamte Wortzahl), indem Sie Byte :0070h (niederwertiges Byte) und :0071h (höherwertiges Byte) überprüfen.

7. Wiederholen Sie die Verfahrensweise für das Laden von Blöcken (Schritte 1–6), bis alle Blöcke geladen sind.

8. Setzen Sie die KTx-Karte zurück, indem Sie **01h** in Byte :0803h schreiben.

Ziehen Sie Tabelle 2.E zu Rate, um den Inhalt jeder Adresse zu sehen, nachdem Sie den Herunterladevorgang abgeschlossen haben.

Tabelle 2.E
Speicherbelegung des Ladeprogramms im Dual-Port

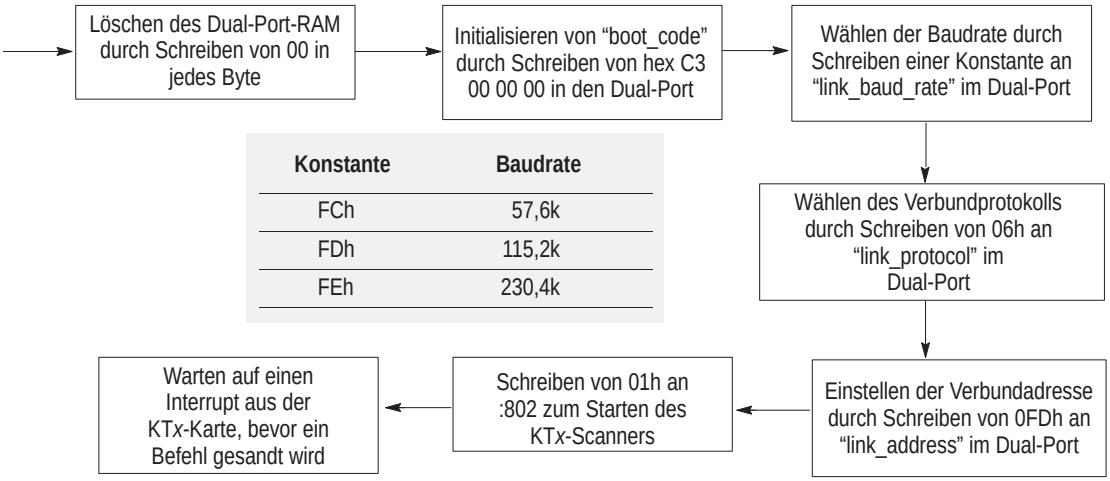
Dual-Port-Adresse des Hosts	Inhalt des Dual-Ports
:0000h – :006Fh	LOADPCL.BIN
:0070h	Gesamte Wortzahl (niederwertiges Byte)
:0071h	Gesamte Wortzahl (höherwertiges Byte)
:0072h	Größenwort (niederwertiges Byte)
:0073h	Größenwort (höherwertiges Byte)
:0074h	Fehlerstatus
:0075h	Handshake
:0076h – :007Fh	LOADPCL.BIN
:0080h – :07FFh	Ladeplatz

Initialisieren

Der Host und die KT*x*-Karte teilen sich Initialisierungsaufgaben.

Aufgabenbereich des Hosts

Während des Initialisierens sollte das Programm folgende Aufgaben ausführen:



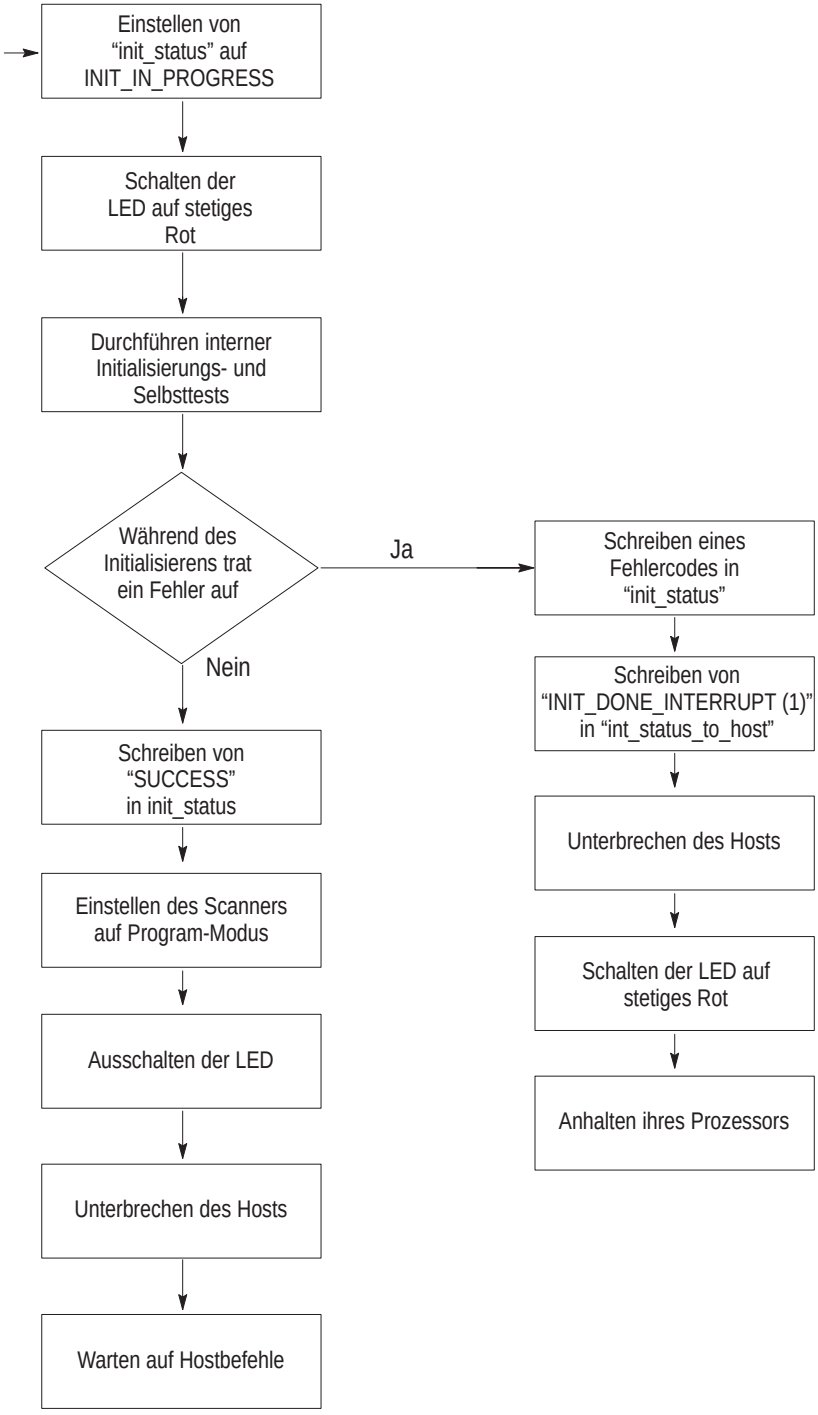
Konstante	Baudrate
FCh	57,6k
FDh	115,2k
FEh	230,4k

Beim Eintreten des Interrupts überprüfen Sie init_status auf ≠ init_in_progress; dies bedeutet SUCCESS oder ein Fehlercode.

Die Dual-Port-Speicherpositionen boot_code, link_baud_rate, link_address und link_protocol werden in Anhang A definiert.

Aufgabenbereich der KTx-Karte

Während des Initialisierens führt die Firmware des Scanners 1784-KT*x* die folgenden Aufgaben aus:



Abschalten

Der Host kann den Scanner jederzeit abschalten, indem er einen beliebigen Wert in das KT*x*-Deaktivierregister schreibt. Das Abschalten kann durch Lesen des KT*x*-Statusregisters (das die Karte im zurückgesetzten Zustand anzeigen sollte) überprüft werden.

Nächster Schritt

Kapitel 3 beschreibt die Programmierung: Layout des Dual-Ports, Handshaking und Interrupts.

Kapitel 3

Programmierüberblick

Kapitelinhalt

Zum Schreiben effektiver Programme müssen Sie die folgenden Konzepte verstehen:

- Layout des Dual-Ports
- Adapterstatustabelle
- E/A-Datentafeln

Sie müssen außerdem mit der Befehlsschnittstelle und der Bearbeitung von Host-Interrupts vertraut sein.

Layout des Dual-Ports

Der Dual-Port-RAM der KT*x*-Karte wird zur Kommunikation zwischen dem Host und dem Scanner 1784-KT*x* verwendet. Es folgt eine Kurzbeschreibung des Speicher-Layouts.

DP-Offset	Speicherbereich
:000h-017h	Allgemeiner Bereich des Dual-Ports
:018h-07bh	Reserviert
:07ch-07fh	Verschiedene Scannervariablen
:080h-1ffh	Adapterstatustabelle
:200h-2ffh	Befehlspuffer
:300h-3ffh	Bestätigungspuffer
:400h-5ffh	Eingangsdatentafel
:600h-7ffh	Ausgangsdatentafel

Eine komplette Auflistung befindet sich in Anhang A.

Adapterstatustabelle

Es gibt für jede der 128 möglichen Adapteradressen 3 lückenlose Datenbytes im Dual-Port:

- Adapter-Konfigurationsbyte
- Adapter-Störungsbyte
- Adapter-Neuversuchszählbyte

Kapitel 6 definiert den Inhalt der Statusbytes und erläutert die Adressierung von Statusbytes.

E/A-Datentafeln

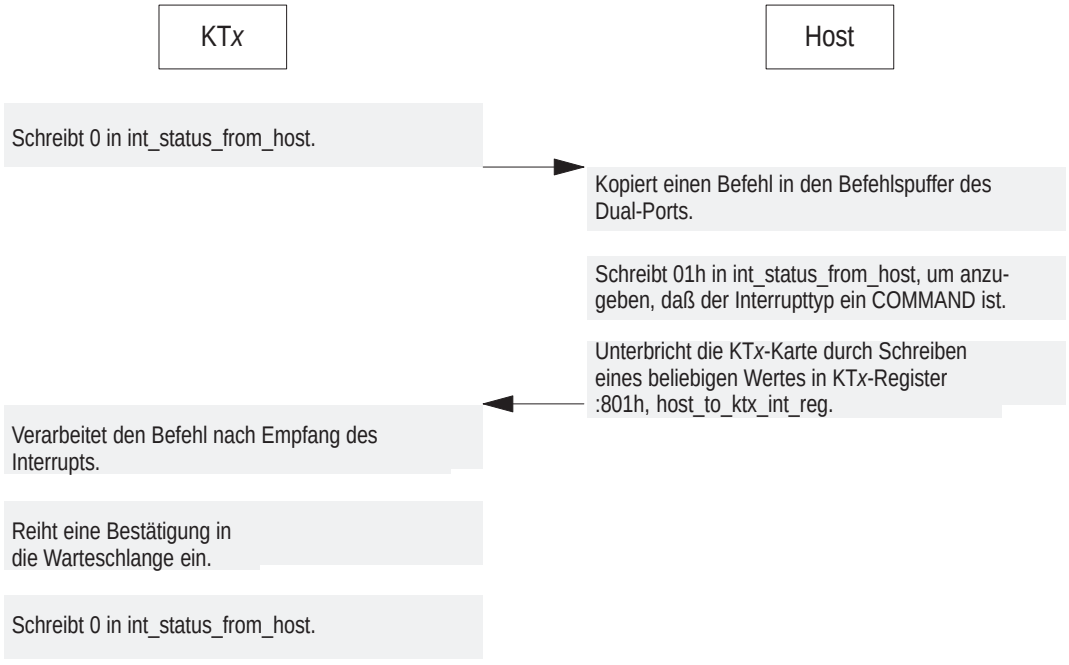
Datentafel	Inhalt
Eingang	Von den Adaptern in E/A-Antworten an den Scanner übertragene Eingangsdaten Die KT*x*-Karte aktualisiert die Tafel, wenn Daten im dezentralen E/A-Verbund empfangen werden. Der Host kann diese Daten jederzeit lesen, sollte diese jedoch nicht modifizieren.
Ausgang	Ausgangsdaten, die der Scanner in E/A-Befehlen an die Adapter überträgt Der Host ist für das Schreiben von Daten in die Ausgangsdatentafel verantwortlich und kann dies jederzeit tun; es ist kein Handshaking mit der KT*x*-Karte erforderlich.

Informationen zum Lesen der bzw. Schreiben in die E/A-Datentafeln sind Kapitel 7 zu entnehmen.

Befehlsschnittstelle

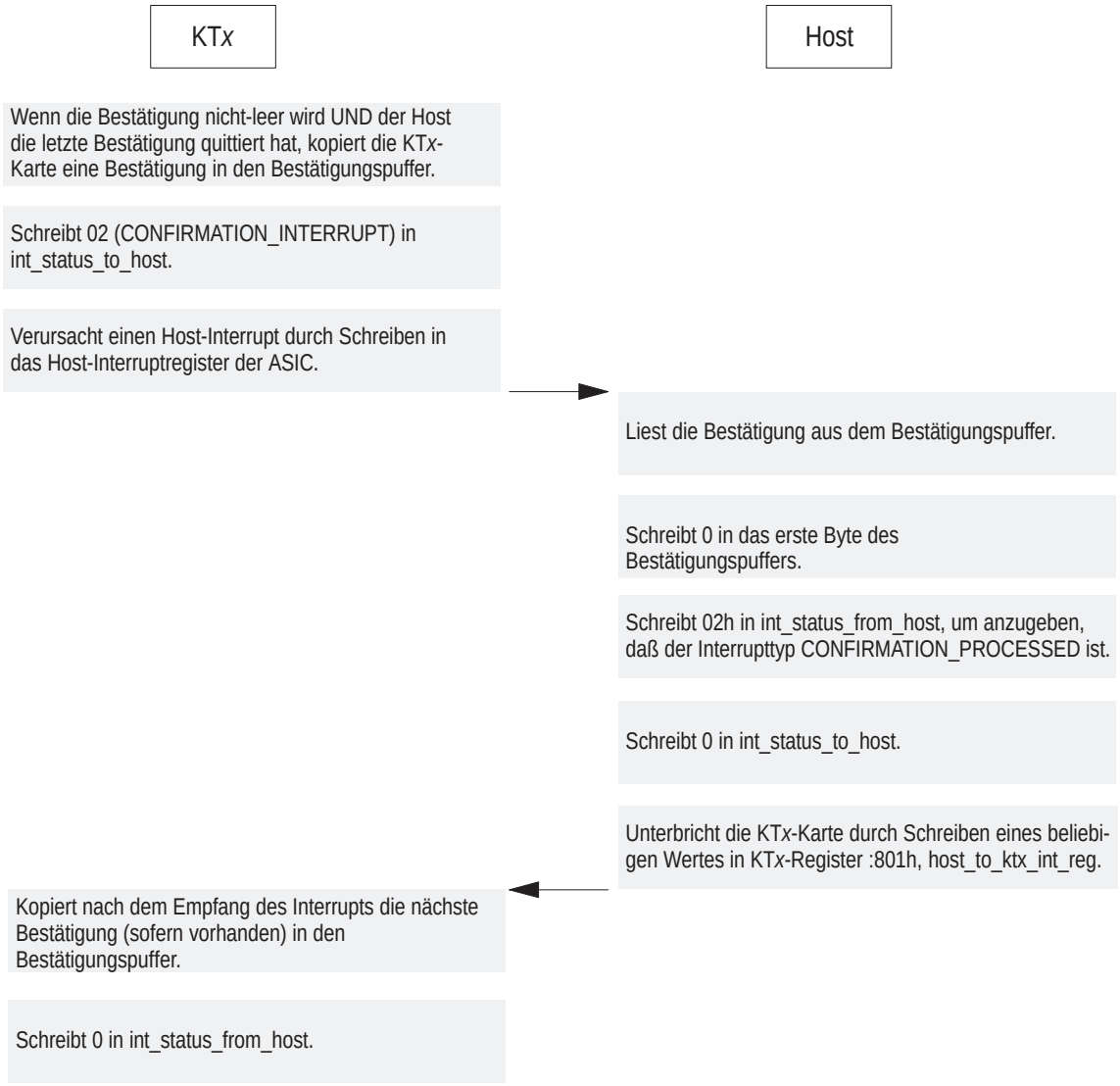
Der Host konfiguriert und steuert den Betrieb des Scanners, indem er Befehle über den Befehlspuffer im Dual-Port erteilt. Der Scanner informiert den Host mittels des Bestätigungspuffers über den Vervollständigungsstatus, gleichgültig ob die Operation erfolgreich war oder nicht. Ein gewisses Handshaking zwischen dem Host und dem Scanner ist erforderlich, um die Verwendung dieser Puffer zu koordinieren. Das Handshaking für beide Befehle und Bestätigungen wird nachstehend beschrieben. Adressen für Dual-Port-Variablen sind in Anhang A aufgeführt.

Das folgende Diagramm veranschaulicht das Handshaking für eine Befehlsfolge. Beachten Sie, daß der Host u.U. keine Befehlsfolge erteilt, bis die Variable int_status_from_host im Dual-Port gelöscht wurde.



Bestätigungs-Handshaking

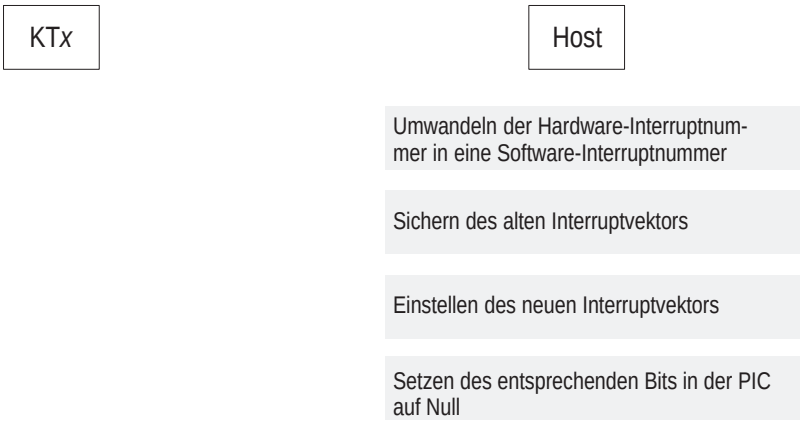
Jedesmal, wenn der Scanner einen Host-Befehl verarbeitet, setzt er eine Bestätigung (mit Vervollständigungsstatus des Befehls) in seine Bestätigungswarteschlange. Bestätigungen werden aus der Warteschlange entfernt und in den Bestätigungspuffer des Dual-Ports kopiert, wenn der Bestätigungspuffer vom Host verarbeitet wird. Das nachstehende Diagramm veranschaulicht das Handshaking für eine Bestätigungsfolge. Beachten Sie, daß der Host u.U. keine Bestätigungsfolge erteilt, bis die Variable int_status_from_host im Dual-Port gelöscht wurde.



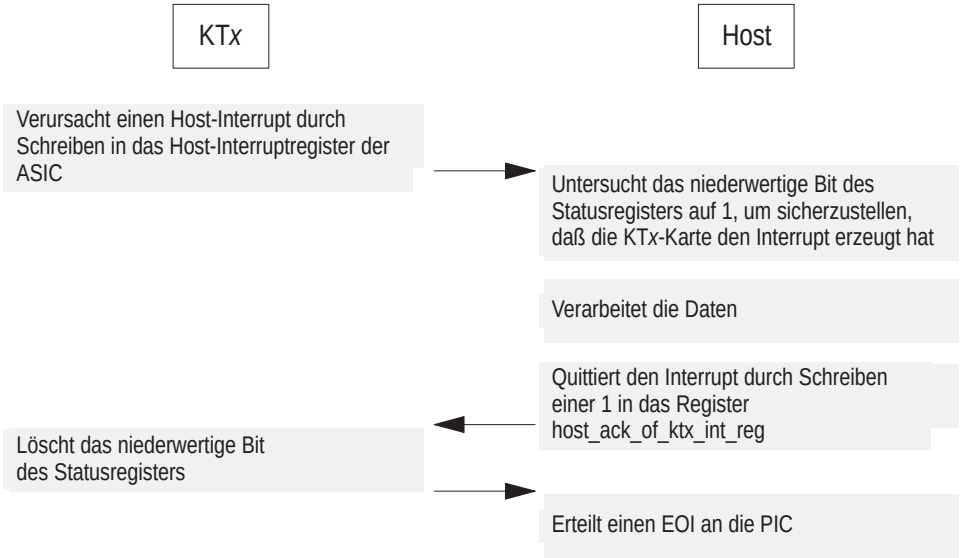
Bearbeitung von Host-Interrupts

Zum ordnungsgemäßen Bearbeiten von Interrupts aus der KT*x*-Karte muß der Host (in diesem Falle ein IBM PC-kompatibler Computer mit DOS) zuerst eine Interrupt-Bearbeitungsroutine (ISR) installieren und den entsprechenden Interrupt aktivieren.

Zum Installieren der Interrupt-Bearbeitungsroutine muß zunächst der Hardware-Interrupt (0–15) in eine Software-Interruptnummer (0–7,0x70–0x78) umgewandelt werden. Anschließend muß die Adresse der ISR in den entsprechenden Software-Interruptvektor geschrieben werden. Der Interrupt muß dann durch Setzen des entsprechenden Bits der speicherprogrammierbaren Interrupt-Steuerung (PIC) auf Null aktiviert werden. Das nachstehende Diagramm veranschaulicht die Schritte, die zum Installieren und Aktivieren von Interrupts in einem DOS-Host erforderlich sind.



Der Host sollte im Rahmen der Interrupt-Bearbeitungsroutine sicherstellen, daß der Interrupt von der KT*x*-Karte erzeugt wurde, die Daten verarbeiten, den Interrupt an die KT*x*-Karte quittieren und dann einen End-of-interrupt (EOI)-Befehl an die PIC erteilen. Das nachstehende Digramm veranschaulicht die von der ISR zu implementierenden Schritte.



Installieren und Aktivieren des Interrupts

Das Code-Fragment in Beispiel 3.A ist ein Beispiel für das Aktivieren von Interrupts und das Einstellen des Interruptvektors auf einen bestimmten Hardware-Interrupt bzw. eine bestimmte ISR unter DOS.

Zuerst wird die Hardware-Interruptnummer in eine Software-Interruptnummer umgewandelt, indem 8 und dann 60h hinzugefügt wird, falls die Interruptnummer größer als 7 ist. Dies ist erforderlich, um die PC-Abbildung von Hardware- auf Software-Interruptnummern umzuändern. Der alte Interruptvektor wird dann für den späteren Gebrauch gesichert, und die neue ISR-Adresse wird als neuer Vektor eingestellt. Anschließend wird das entsprechende Bit in der PIC auf 0 gesetzt, um den Interrupt zu aktivieren.

Beispiel 3.A
Installieren und Aktivieren von Interrupts

```
#define LowPICPort 0x20       /* Port address for Prog. Interrupt Controller for ints 0–7 */
#define HighPICPort 0xA0      /* Port address for Prog. Interrupt Controller for ints 8–15 */

void interrupt (*OldInterruptServiceRoutine)();      /* pointer to store old interrupt handler */

void InitInterrupt(unsigned char InterruptNumber, void interrupt (*InterruptServiceRoutine)())
{
unsigned char  PICMask,                     /* storage for prog. interrupt controller mask */
               PICPortBase,                 /* Port address of either first PIC or second PIC */
               SWInterruptNumber;           /* Software Interrupt number, converted from HW */

               /* Insure that InterruptNumber is valid one for KTX – leave in for debug only */

               assert((InterruptNumber > 2) && (InterruptNumber < 16));

               /* Set the base port of the Programmable Interrupt Controller depending on
                  whether the interrupt is on the first or second PIC */

               if (InterruptNumber < 8) PICPortBase = LowPICPort;
               else PICPortBase = HighPICPort;

               /* Convert from hardware interrupt numbers (0–7) to software interrupt numbers
                  0x8–0xf and hardware interrupt numbers (8–15) to software interrupt
                  numbers 0x70–0x78  */

               SWInterruptNumber = InterruptNumber + 0x8;
               if (SWInterruptNumber > 0xf) SWInterruptNumber += 0x60;

               /* Save the old interrupt handler vector, and install the new one */
               /* Need to disable interrupts when changing vectors */

               disable ();
               OldInterruptServiceRoutine = getvect(SWInterruptNumber);
               setvect(SWInterruptNumber, InterruptServiceRoutine);

               /* Enable the interrupt by setting the appropriate bit in the PICMask to 0 */

               PICMask = inportb(PICPortBase+1);
               delay(50);                   /* Delay for PIC to settle */
               outportb(PICPortBase+1, PICMask & ~(1 << (InterruptNumber % 8)));
               delay(50);                   /* Delay for PIC to settle */
               enable ();                   /* Re–enable interrupts */
}
```

Schreiben der Interrupt-Bearbeitungsroutine

Das Code-Fragment in Beispiel 3.B ist ein Beispiel für eine Interrupt-Bearbeitungsroutine der Karte 1784-KT*x*.

Wichtig: Dieses Routinebeispiel verarbeitet keine Daten; es soll lediglich als Gerüst dienen, auf dem noch weiter aufgebaut werden muß.

Zuerst wird das niederwertige Bit des Statusregisters überprüft, um festzustellen, ob die KT*x*-Karte einen Interrupt erzeugt hat. Hat die Karte einen Interrupt erzeugt, werden die Datenfelder überprüft, um festzustellen, welcher Interrupttyp erstellt wurde und welche Maßnahmen zu ergreifen sind. Der Host schreibt dann eine 1 in das Register host_ack_of_ktx_int_reg, um das niederwertige Bit des Statusregisters zu löschen. Vor dem Beenden der Routine muß ein End-of-interrupt-Befehl an die erste PIC sowie an die zweite PIC, falls sich der Interrupt auf der zweiten PIC befindet (größer als 7), erteilt werden.

Beispiel 3.B
Schreiben der Interrupt-Bearbeitungsroutine

```
#define LowPICPort 0x20       /* Port address for Prog. Interrupt Controller for ints 0–7 */
#define HighPICPort 0xA0      /* Port address for Prog. Interrupt Controller for ints 8–15 */
#define EOI 0x20              /* End Of Interrupt to PIC */

#define INT_PENDING_BIT 0x01  /* bit mask for interrupt pending bit in status register */

extern KTX_DUALPORT far *dp;                /* pointer to KTX's dualport */
extern unsigned int InterruptNumber;        /* Global interrupt number for testing for second PIC */

void interrupt InterruptServiceRoutine(void)
{

UBYTE   temp;

/*===========================================================================
**
**      Verify it was the KTX that caused the interrupt by reading the
**      status register.
**
===========================================================================*/
/**** Bit 0 of status reg == 1 when KTX ****/
/**** is the source of a host interrupt ****/

if (!(dp->status_reg & 1))
    return;

/**** Clear bit 0 of the status reg by ****/
/**** writing to acknowledge register. ****/

dp->host_ack_of_ktx_int_reg = ACKNOWLEDGE_KTX;


/*===========================================================================
**
**      Service the COS interrupt first, since it can happen most often...
**              1) make sure it's enabled
**              2) if so, see if we got a COS interrupt
**              3) if so, acknowledge it and check for COS overrun
**
===========================================================================*/
```

```
if (dp->config_data.enable_cos_detect &&
        (dp->COS_link_address != COS_ACKNOWLEDGE)) {

    temp = dp->COS_link_address ;
    dp->COS_link_address = COS_ACKNOWLEDGE;
    if (dp->operating_status.cos_overrun) {

        /**** When this bit is set, it indicates the host is unable
        ***** to keep up with COS interrupts, i.e., the system input
        ***** state is changing more rapidly than the host is able to
        ***** process.  COS interrupts should be enabled only in
        ***** systems that have infrequent state changes.
        ****/
    }
    /**** The application programmer should write a routine that
    ***** services the COS.  The handle_input_COS routine called
    ***** below is not one of the supplied examples.
    ****/

    handle_input_COS (temp);

}

/*=============================================================================
**
**      Service the end-of-scan-list interrupt next, since it is also
**      time-critical.
**              1) make sure it's enabled
**              2) if so, see if we got an end-of-scan-list interrupt
**              3) if so, acknowledge it
**
=============================================================================*/

if (dp->config_data.interrupt_after_each_scan
        && dp->running_status.end_of_scan_list) {

    /**** Acknowledge the end_of_scan_list interrupt ****/

    dp->running_status.end_of_scan_list = FALSE;

    /**** The application programmer may want to write a routine that
    ***** services the I/O image tables whenever a end_of_scan_list
    ***** interrupt occurs.  The designer should keep in mind that
    ***** links operating at a high baud rate and with short scan lists
    ***** are likely to cause frequent interrupts.  The host should be
    ***** capable of handling these interrupts in a timely fashion.
    ****/
}

/*=============================================================================
**
**      Service the regular interrupt next, since it is queued
**              1) see what type it is
**              2) service by type...
**
=============================================================================*/

/**** dp->int_status_to_host is the interrupt type ****/

switch (dp->int_status_to_host) {
```

```
        /**** Host ack'd last interrupt ****/

        case NO_KTX_INT_AVAILABLE :
                break;

        /**** Initialization is complete ****/

        case INIT_DONE_INTERRUPT :

                /**** Check completion status    ****/

                if (dp->init_status != SUCCESS)
                    unrecoverable_error(dp);

                /**** If good, clear interrupt type ****/

                else
                   dp->int_status_to_host = 0;
                break;


        /**** A confirmation is available ****/

        case CONFIRMATION_INTERRUPT :

                /**** The application programmer needs to write a routine
                ***** that handles confirmations.  The other examples
                ***** provided show how to do it when the application
                ***** waits for a confirmation before proceeding to the
                ***** next command.  An asynchronous version would need
                ***** to match up confirmations with pending commands
                ***** using the transaction number and acknowledge the
                ***** confirmation.
                ****/

                break;



        /**** A processing problem is an unrecoverable error ****/

        case PROCESSING_PROBLEM_INTERRUPT  :
                unrecoverable_error(dp);
                break;

        /**** Unknown interrupt type ****/

        default :
                break;
}

}
          /* Need to issue the End Of Interrupt to the PIC controller(s) */

          outportb(LowPICPort, EOI);
          if (InterruptNumber > 7) outportb(HighPICPort, EOI); /* if int > 7 then second PIC also */

}
```

Nächster Schritt Kapitel 4 beschreibt die fünf Scanner-Verwaltungsbefehle.

Kapitel 4

Erteilen von Scanner-Verwaltungsbefehlen

Kapitelinhalt

Dieses Kapitel beschreibt die fünf Scanner-Verwaltungsbefehle und zeigt die ordnungsgemäße Syntax für grundlegende Befehls- und Bestätigungsstrukturen. Nach dem Lesen dieses Kapitels sollten Sie folgende Aufgaben ausführen können:

- Erteilen einer der fünf Scanner-Verwaltungsbefehle
- Interpretieren von Bestätigungen

Grundlegende Befehls- und Bestätigungsstrukturen

Befehle aus dem Host an den Scanner 1784-KT*x* verwenden die folgende Struktur:

Befehlssyntax

host_command	(Byte)
transaction_number	(Byte)
command_length	(Byte)
Daten	
HINWEIS: Befehle beginnen stets beim ersten Byte des Befehlspuffers im Dual-Port-RAM.	

Bestätigungen aus dem Scanner 1784-KT*x* an den Host verwenden die folgende Struktur:

Bestätigungssyntax

host_command	(Byte)
transaction_number	(Byte)
confirmation_status	(Byte)
confirmation_length	(Byte)
Daten	
HINWEIS: Bestätigungen beginnen stets beim ersten Byte des Bestätigungspuffers im Dual-Port-RAM.	

Der Scanner 1784-KT*x* antwortet auf jeden Befehl mit einer Bestätigung. Der Scanner reiht Bestätigungen in der Reihenfolge ein, in der die Befehle empfangen werden (mit Ausnahme von Blocktransferbefehlen). Blocktransferbefehle werden erst nach Abschluß der Blocktransferfolge bestätigt. Informationen zu Blocktransfers sind in Kapitel 5 enthalten.

Der Scanner kopiert die Variablen host_command und transaction_number direkt in einen Eintrag in der Bestätigungswarteschlange, so daß der Host die Bestätigung auf den erteilen Befehl abstimmen kann. Die Variable confirmation_status ist ein 1-Byte-Code, der den endgültigen Abschlußstatus des Befehls zeigt. Die Variable confirmation_length gibt die Länge (in Byte) der nachfolgenden Daten (sofern vorhanden) an.

Host-Befehle

Fünf der sieben 1784-KT*x*-Hostbefehle sind Scanner-Verwaltungsbefehle:

- Set Mode
- Set Scan List
- Autoconfigure
- Set Fault Group
- Get Scan List

Diese Befehle werden im einzelnen beschrieben, einschließlich Befehlssyntax, Fehlercodes und Bestätigungsprozeß für den jeweiligen Befehl.

Die zwei verbleibenden Befehle, Host BT Write und Host BT Read, sind Blocktransferbefehle und werden in Kapitel 5 beschrieben.

Set Mode (Modus einstellen)

Verwenden Sie den Befehl Set Mode, um den Scanner 1784-KT*x* auf einen von drei Modi einzustellen: Program, Test oder Run.

Modus	Scannerbetrieb
Program	Aktualisiert die Eingangsdatentafel, sendet jedoch keine Ausgänge an die Adapter.
Test	Sendet E/A-Rücksetzbefehle (Ausgänge sind inaktiv) und Blocktransfers
Run	Sendet E/A-Befehle (Ausgänge sind aktiv) und Blocktransfers

Der Scanner befindet sich anfänglich im Program-Modus und kommuniziert nicht im dezentralen E/A-Verbund, bis er über eine Abfrageliste verfügt.

Befehlssyntax	
host_command	1
transaction_number	0 – 255
command_length	1
scanner_mode	Program (0), Test (1) oder Run (2)

Dieser Befehl reiht sofort eine Bestätigung in die Warteschlange ein. Die Bestätigung sieht folgendermaßen aus:

Bestätigung für den Befehl “Set Mode”	
host_command	1
transaction_number	0 – 255 (je nach Angaben des Hosts)
confirmation_status	(siehe Tabelle 4.A)
confirmation_length	0
Daten	–

Tabelle 4.A
Fehlerbedingungen für den Befehl “Set Mode”

Fehlermnemonik	Code	Beschreibung
SUCCESS	0	Moduswechsel war erfolgreich.
SC_BAD_REQUEST	21	Befehlslänge war ungleich 1.
SC_BAD_PARAM	22	Parameter war nicht Program, Test oder Run.
SC_CANNOT_CHANGE_MODE_DURING_SET_SCAN_LIST	23	Modus kann nicht gewechselt werden, während eine neue Abfrageliste eingestellt wird.
SC_CANNOT_CHANGE_MODE_DURING_AUTOCONFIGURE	24	Modus kann nicht während der automatischen Konfiguration gewechselt werden.

Die Routine in Beispiel 4.A stellt den Scanner-Modus ein.

Beispiel 4.A

```
#include ”ktx_dp.h”        /* 1784–KTX scanner dualport definition   *
/
#include ”ktx_err.h”       /* 1784–KTX scanner error codes           *
/
#include ”ktxconst.h”      /* 1784–KTX scanner constants             *
/

UBYTE     set_mode (KTX_DUALPORT far *dp, UBYTE mode, UBYTE trans_num,
                    USHORT timeout)
{
UBYTE     status;

/**** Initialize the command buffer ****/

dp–>cmd_buffer.host_command = SET_MODE;
dp–>cmd_buffer.transaction_num = trans_num;
dp–>cmd_buffer.command_length = 1;
dp–>cmd_buffer.cmd.set_mode.scanner_mode = mode;

/**** Send the command ****/

dp–>int_status_from_host = SCANNER_COMMAND_FROM_HOST;
dp–>host_to_ktx_int_reg = INTERRUPT_KTX;

/**** Get and acknowledge the confirmation  ****/

status = get_confirmation(dp, trans_num, timeout);

if ((status != CONFIRMATION_TIMED_OUT) && (status != TRANS_NUM_MIS-
MATCH))
    acknowledge_confirmation(dp);

return status;
}
```

Set Scan List (Abfrageliste einstellen)

Der Scanner liest und schreibt diskrete E/A-Punkte ständig aus bzw. in E/A-Adaptergeräte im dezentralen E/A-Netzwerk. Der Scanner muß so konfiguriert sein, daß er weiß, welche Adaptermodule sich im Netzwerk befinden. Verwenden Sie hierzu den Befehl Set Scan List, um alle Verbundadressen, die in der Liste enthalten sein sollen, anzugeben. Es können bis zu 64 Einträge enthalten sein, und eine Adapteradresse kann mehrmals erscheinen, falls der Anwender diese Adresse öfter abfragen möchte als andere Adressen in der Liste. Der Befehl Set Scan List ersetzt eine bestehende Abfrageliste und kann nur dann ausgeführt werden, wenn sich der Scanner im Program-Modus befindet.

Befehlssyntax

host_command	2
transaction_number	0 – 255
command_length	1+Anzahl der Einträge in der Abfrageliste
count	1 – 64
adapter_address [64]	Feld der Adapteradressen, die abgefragt werden sollen (1 bis 64 Einträge)

Dieser Befehl reiht sofort eine Bestätigung in die Warteschlange ein. Die Bestätigung sieht folgendermaßen aus:

Bestätigung für den Befehl “Set Scan List”

host_command	2
transaction_number	0 – 255 (je nach Angaben des Hosts)
confirmation_status	(siehe Tabelle 4.B)
confirmation_length	0 (wenn erfolgreich) oder 1 (wenn fehlgeschlagen)
Daten	Verwaiste Adapteradresse (falls zutreffend)

Tabelle 4.B
Fehlerbedingungen für den Befehl “Set Scan List Error”

Fehlermnemonik	Code	Beschreibung
SUCCESS	0	Abfrageliste war gültig.
SCANNER_NOT_PROGRAM	15	Scanner muß sich im Program-Modus befinden.
SCAN_LIST_TOO_LONG	16	Abfrageliste enthält mehr als 64 Einträge.
SET_SCAN_LIST_IN_PROGRESS	17	Ein vorhergehender Befehl “Set Scan List” wird gerade ausgeführt.
AUTOCONFIGURING_IN_PROG	18	Ein Befehl “Autoconfigure” wird gerade ausgeführt.
SCAN_LIST_CAUSES_FDG_ORPHAN	19	Die neue Abfrageliste würde eine verwaiste Adapteradresse hervorrufen. Die Adresse des Waisen wird im Datenbyte der Bestätigung zurückgeleitet. Es kommt zu einem FDG-Waisen, wenn ein Adapter in der vorhergehenden Abfrageliste zu einer störungsabhängigen Gruppe gehörte, die Adresse des Adapters jedoch nicht in der neuen Abfrageliste enthalten ist.
BAD_COMMAND_DATA_LENGTH	26	Befehlslänge ist ungültig.
INVALID_ADAPTER_ADDRESS	49	Adapteradresse ist größer als 127.
ADAPTER_SIZE_OVERLAP	51	Adaptergrößen überlappen sich.
ERROR_MORE_THAN_32_UNIQUE_DEVICES	53	Es können maximal 32 unverwechselbare Adapter in einer Abfrageliste angegeben werden.

Die Routine in Beispiel 4.B weist den Scanner an, die bestehende Abfrageliste durch die neue, im Befehl angegebene Abfrageliste zu ersetzen. Würde die neue Abfrageliste einen FDG-Waisen hervorrufen, so wird die Adresse des Waisen zusätzlich zum Statuscode in die aufrufende Adresse kopiert.

Beispiel 4.B

```
#include ”ktx_dp.h”    /* 1784–KTX scanner dualport definition */
#include ”ktx_err.h”   /* 1784–KTX scanner error codes   */
#include ”ktxconst.h” /* 1784–KTX scanner constants     */

UBYTE set_scan_list (KTX_DUALPORT far *dp, SCAN_LIST *sl,
                     UBYTE *orphan, UBYTE trans_num, USHORT timeout)
{
UBYTE      status;

/**** Initialize the command buffer ****/

dp–>cmd_buffer.host_command = SET_SCAN_LIST;
dp–>cmd_buffer.transaction_num = trans_num;
dp–>cmd_buffer.command_length = 1 + sl–>count;
dp–>cmd_buffer.cmd.set_scan_list = *sl;

/**** Send the command ****/

dp–>int_status_from_host = SCANNER_COMMAND_FROM_HOST;
dp–>host_to_ktx_int_reg = INTERRUPT_KTX;

/**** Get and process the confirmation  ****/

status = get_confirmation(dp, trans_num, timeout);

switch (status) {

    /* break if no confirmation was available */

    case CONFIRMATION_TIMED_OUT:
           break;

    /* break if the transaction numbers didn’t match */

    case TRANS_NUM_MISMATCH:
           break;

    /* If an adapter became a FDG orphan, copy            */
    /* orphan’s address to caller.                        */
    /* (fall through to acknowledge confirmation) */

    case SCAN_LIST_CAUSES_FDG_ORPHAN:
           *orphan = dp–>confirmation_buffer.conf.set_list.orphan;
           acknowledge_confirmation(dp);
           break;

    /* if a different error occurred or SUCCESS, acknowledge */
    /* the confirmation and return the status to the caller  */

    case SUCCESS:
    default:
           acknowledge_confirmation(dp);
           break;
}
return status;
}
```

Autoconfigure (Automatische Konfiguration)

Dieser Befehl weist den Scanner an, alle möglichen Verbundadressen im Verbund abzufragen, um zu bestimmen, welche Geräte im Verbund vorhanden sind. Eine neue Abfrageliste wird erzeugt, die aus den gefundenen Geräten in numerischer Reihenfolge besteht. Der Host kann eine Kopie dieser neuen Abfrageliste erhalten, indem er nach Abschluß der automatischen Konfiguration einen Befehl Get Scan List erteilt.

Befehlssyntax

host_command	3
transaction_number	0 – 255
command_length	0

Dieser Befehl reiht sofort eine Bestätigung in die Warteschlange ein. Die Bestätigung sieht folgendermaßen aus:

Bestätigung für den Befehl "Autoconfigure"

host_command	3
transaction_number	0 – 255 (je nach Angaben des Hosts)
confirmation_status	(siehe Tabelle 4.C)
confirmation_length	0
Daten	–

Tabelle 4.C
Fehlerbedingungen für den Befehl "Autoconfigure"

Fehlermnemonik	Code	Beschreibung
SUCCESS	0	Abfrageliste war gültig.
SCANNER_NOT_PROGRAM	15	Scanner muß sich im Program-Modus befinden.
SET_SCAN_LIST_IN_PROGRESS	17	Ein vorhergehender Befehl "Set Scan List" wird gerade ausgeführt.
SCANNER_ALREADY_AUTOCONFIGURING	25	Ein Befehl "Autoconfigure" wird gerade ausgeführt.
ADAPTER_SIZE_OVERLAP	51	Adaptergrößen überlappen sich.
TOO_MANY_ADAPTERS	130	Es können maximal 32 unverwechselbare Geräte im RIO-Verbund vorhanden sein.

Die Routine in Beispiel 4.C weist den Scanner an, eine automatische Konfiguration durchzuführen, und wartet dann auf eine Bestätigung. Ein Befehl Autoconfigure verlangt, daß der Scanner alle 128 möglichen Verbundadressen im dezentralen E/A-Verbund abfragt. Dieser Vorgang kann bei einer Übertragungsgeschwindigkeit von 57,6 kBaud bis zu 6 Sekunden dauern.

Beispiel 4.C

```
#include ”ktx_dp.h”        /* 1784–KTX scanner dualport definition   *
/
#include ”ktx_err.h”       /* 1784–KTX scanner error codes           *
/
#include ”ktxconst.h”      /* 1784–KTX scanner constants             *
/

UBYTE     autoconfigure (KTX_DUALPORT far *dp, UBYTE trans_num,
                         USHORT timeout)
{
UBYTE     status;

/**** Initialize the command buffer ****/

dp–>cmd_buffer.host_command = AUTOCONFIG;
dp–>cmd_buffer.transaction_num = trans_num;
dp–>cmd_buffer.command_length = 0;

/**** Send the command ****/

dp–>int_status_from_host = SCANNER_COMMAND_FROM_HOST;
dp–>host_to_ktx_int_reg = INTERRUPT_KTX;

/**** Get and acknowledge the confirmation  ****/

status = get_confirmation(dp, trans_num, timeout);

if ((status != CONFIRMATION_TIMED_OUT) && (status != TRANS_NUM_MIS-
MATCH))
    acknowledge_confirmation(dp);

return status;
}
```

Set Fault Group (Störungsgruppe einstellen)

Verwenden Sie den Befehl Set Fault Group, um den Scanner anzuweisen, eine Gruppe von Adaptern als Einheit zu behandeln. Tritt eine Störung in einem der Adapter auf, so weist der Scanner die anderen Adapter an, in den Störungsmodus zu schalten. Sie stellen eine Liste von 128 Byte bereit, ein Byte für jede mögliche physikalische Adapteradresse. Jedes Byte ist wie folgt definiert:

Bit	Definition
0 – 3	Störungsgruppennummer (0–15)
4	Bei aktiviertem Bit soll diese Adresse in der Störungsgruppe enthalten sein
5 – 7	Nicht belegt

Der Host kann somit bis zu 16 Störungsgruppen und die Mitgliedschaft eines Adapters in einer der Störungsgruppen definieren. Befindet sich der Adapter nicht in der Abfrageliste, verwirft der Scanner den Störungsgruppenbefehl und leitet eine Fehlerbestätigung zurück.

Befehlssyntax	
host_command	6
transaction_number	0 – 255
command_length	128
fault_group_data	128 Byte (siehe obige Definition)

Dieser Befehl reiht sofort eine Bestätigung in die Warteschlange ein. Die Bestätigung sieht folgendermaßen aus:

Bestätigung für den Befehl “Set Fault Group”	
host_command	6
transaction_number	0 – 255 (je nach Angaben des Hosts)
confirmation_status	(siehe Tabelle 4.D)
confirmation_length	0

Tabelle 4.D
Fehlerbedingungen für den Befehl “Set Fault Group”

Fehlermnemonik	Code	Beschreibung
SUCCESS	0	Befehl war erfolgreich.
SCANNER_NOT_PROGRAM	15	Scanner muß sich für diesen Befehl im Program-Modus befinden.
BAD_COMMAND_DATA_LENGTH	26	Befehlsdatenlänge ist ungleich 128.
ADDRESS_NOT_IN_SCAN_LIST	32	Adapteradresse muß sich in der Abfrageliste befinden, falls sie in eine Störungsgruppe aufgenommen werden soll.
CANNOT_SET_FG_WHILE_AUTOCONFIGURING	33	Auf den Abschluß der automatischen Konfiguration warten und dann den Befehl erneut erteilen.
CANNOT_SET_FG_WHILE_SETUP_SCAN_LIST	34	Auf den Abschluß der Abfragelistenerstellung warten und dann den Befehl erneut erteilen.

Die Routine in Beispiel 4.D weist den Scanner an, die bestehende Störungsgruppenliste durch die neue, im Befehl angegebene Liste zu ersetzen.

Beispiel 4.D

```
#include ”ktx_dp.h”        /* 1784–KTX scanner dualport definition   *
/
#include ”ktx_err.h”       /* 1784–KTX scanner error codes           *
/
#include ”ktxconst.h”      /* 1784–KTX scanner constants             *
/

UBYTE set_fault_dependent_group (KTX_DUALPORT far *dp,
SET_FAULT_GROUP_CMD                          *fdg, UBYTE trans_num,
USHORT timeout)
{
UBYTE          status;

/**** Initialize the command buffer ****/

dp–>cmd_buffer.host_command = SET_FAULT_GROUP;
dp–>cmd_buffer.transaction_num = trans_num;
dp–>cmd_buffer.command_length = sizeof(SET_FAULT_GROUP_CMD);
dp–>cmd_buffer.cmd.set_fault_group = *fdg;

/**** Send the command ****/

dp–>int_status_from_host = SCANNER_COMMAND_FROM_HOST;
dp–>host_to_ktx_int_reg = INTERRUPT_KTX;

/**** Get and process the confirmation  ****/

status = get_confirmation(dp, trans_num, timeout);

if ((status != CONFIRMATION_TIMED_OUT) && (status != TRANS_NUM_MIS-
MATCH))
    acknowledge_confirmation(dp);

return status;
}
```

Get Scan List (Abfrageliste erhalten)

Verwenden Sie den Befehl Get Scan List, um eine Kopie der aktuellen Abfrageliste vom Scanner zu erhalten. Dieser Befehl ist am nützlichsten nach Abschluß einer automatischen Konfiguration, da aus der Liste ersichtlich ist, welche Adapter im Verbund gefunden wurden.

Befehlssyntax

host_command	7
transaction_number	0 – 255
command_length	0

Dieser Befehl reiht sofort eine Bestätigung in die Warteschlange ein. Die Bestätigung sieht folgendermaßen aus:

Bestätigung für den Befehl "Get Scan List"

host_command	7
transaction_number	0 – 255 (je nach Angaben des Hosts)
confirmation_status	SUCCESS (ERFOLG) ①
confirmation_length	Anzahl der Einträge in der Abfrageliste + 2
scan_list_count	Anzahl der Einträge in der Abfrageliste
adapter_address [64]	Feld der Adapteradressen, die abgefragt werden sollen (1 bis 64 Einträge)

① SUCCESS ist der einzig mögliche Zustand.

Die Routine in Beispiel 4.E weist den Scanner an, die aktuelle Abfrageliste zurückzuleiten.

Beispiel 4.E

```
#include ”ktx_dp.h”    /* 1784–KTX scanner dualport definition   *
/
#include ”ktx_err.h”   /* 1784–KTX scanner error codes           *
/
#include ”ktxconst.h”  /* 1784–KTX scanner constants             *
/

UBYTE get_scan_list (KTX_DUALPORT far *dp, SCAN_LIST *sl,
                     UBYTE trans_num, USHORT timeout)
{
int                  i;
UBYTE                status;

/**** Initialize the command buffer ****/

dp–>cmd_buffer.host_command = GET_SCAN_LIST;
dp–>cmd_buffer.transaction_num = trans_num;
dp–>cmd_buffer.command_length = 0;

/**** Send the command ****/

dp–>int_status_from_host = SCANNER_COMMAND_FROM_HOST;
dp–>host_to_ktx_int_reg = INTERRUPT_KTX;

/**** Get and process the confirmation  ****/

status = get_confirmation(dp, trans_num, timeout);

switch (status) {

    /* break if no confirmation was available */

    case CONFIRMATION_TIMED_OUT:
         break;

    /* break if the transaction numbers didn’t match */

    case TRANS_NUM_MISMATCH:
         break;

    /* when successful, copy the scan list to the */
    /* caller and acknowledge the confirmation            */

    case SUCCESS:
         sl–>count = dp–>confirmation_buffer.conf.get_list.count;
         for (i=0; i<sl–>count; i++)
             sl–>scan_list[i] =
                 dp–>confirmation_buf-
fer.conf.get_list.scan_list[i];
         acknowledge_confirmation(dp);
         break;

    /* if an error occurred, acknowledge the confirmation */
    /* and return the status to the caller                */

    default:
         acknowledge_confirmation(dp);
         break;
}
return status;
}
```

Nächster Schritt

Kapitel 5 beschreibt die zwei Blocktransferbefehle: Host BT Write und Host BT Read.

Kapitel 5

Erteilen von Blocktransferbefehlen

Kapitelinhalt

Dieses Kapitel erläutert das Übertragen von Daten zwischen intelligenten E/A-Modulen (Blocktransfermodulen) und dem Datenbereich des Programms. Nach dem Lesen dieses Kapitels sollten Sie folgende Aufgaben ausführen können:

- Lesen eines Datenblocks aus einem intelligenten E/A-Modul
- Schreiben eines Datenblocks in ein intelligentes E/A-Modul
- Berechnen der Zeitspanne, die durch das Ausführen eines bestimmten Blocktransfers in Anspruch genommen wird

Zur knapperen Darstellung und besseren Lesbarkeit dieses Kapitels wird die Abkürzung BT für Blocktransfer verwendet.

Funktionsweise eines Blocktransfers

Es wird im nachstehenden die Folge von Aktionen beschrieben, die an der Ausführung eines BT beteiligt sind:

- Das Programm initialisiert den KT*x*-Befehlspuffer mit BT-Befehlsdaten.
- Das Programm unterbricht die KT*x*-Karte und informiert diese somit, daß es sich um einen Befehlsinterrupt handelt.
- Der Scanner antwortet nach einer Weile auf den Interrupt und überprüft, ob der BT-Befehl syntaktisch korrekt ist.
- Wenn der BT abgeschlossen ist oder wenn er nicht innerhalb von 4 Sekunden nach Einreihung in die Warteschlange abgeschlossen ist, unterbricht der Scanner den Host mit einer Bestätigung. Bei einem (erfolgreichen) BT-Lesevorgang leitet der Scanner zudem die von ihm gelesenen Daten zurück.

In den folgenden Abschnitten werden diese Schritte näher beschrieben.

Adapter Decide Length (ADL)

Bei den meisten Operationen befiehlt der Host dem BT-Modul, wie viele Daten in einem BT-Lese- bzw. -Schreibvorgang übertragen werden sollen. Es gibt Umstände (wie z.B. beim erstmaligen Konfigurieren eines BT-Moduls), unter denen der Host das BT-Modul die Anzahl der zu übertragenden Worte festlegen läßt. Dieser Modus wird als Adapter Decide Length (ADL) bezeichnet.

Um einen ADL-Blocktransfer durchzuführen, erteilt der Host einen Befehl Host BT Read oder Host BT Write an den Scanner, gibt jedoch bt_data_len mit Null an. Handelt es sich um einen Host BT Write, stellt der Host außerdem bis zu 64 zu übertragende Datenworte bereit. Befinden sich weniger Worte im Parameter bt_write_data des Befehlspuffers als vom Adapter angefordert, leitet der Scanner eine Bestätigung mit dem folgenden Fehlercode zurück:

MODULE_REQUESTED_TOO_MUCH_DATA

Um diesen Fehler zu vermeiden, empfehlen wir die Ablage von 64 Worten im Parameter bt_write_data des Befehlspuffers, damit ein ADL-BT diesen Fehler nicht verursachen kann.

Beispiel:

Der Host möchte den Adapter die Länge eines BT-Schreibvorgangs bestimmen lassen. Der Host legt 16 Datenworte in bt_write_data ab, stellt bt_data_len auf 0 für Adapter Decide Length und die Befehlslänge auf 35 (3 + (2 * 16)) ein. Der Scanner informiert den Adapter, daß er einen ADL-BT-Schreibvorgang durchführen möchte. Fordert der Adapter mehr als 16 Datenworte an, was der vom Host bereitgestellten Wortzahl entspricht, verwirft der Scanner die Anforderung des Adapters und leitet eine Bestätigung mit dem Fehlercode MODULE_REQUESTED_TOO_MUCH_DATA an den Host zurück.

Host BT Write

Die Host-Anwendung erteilt einen Befehl Host BT Write, wenn sie einen Blocktransfer an einen Adapter einleiten möchte. Um die Antwort nach Abschluß des Blocktransfers auf die Anforderung abzustimmen, muß der Host folgende Parameter bereitstellen:

- die *logische Rack*-Adresse
- die Steckplatzadresse des Blocktransfermoduls
- die an das Modul zu schreibenden Daten
- eine unverwechselbare Transaktionsnummer

Es können jeweils bis zu 64 Blocktransferbefehle in die Warteschlange eingereiht werden,wobei jedes Blocktransfermodul nur einen BT ausführen darf (bis zu 16 BTs je logisches Rack). Der Scanner stellt ein 4-Sekunden-Zeitwerk für jede BT-Anforderung ein. Läuft das Zeitwerk ab, bevor der BT-Austausch mit dem Adapter abgeschlossen ist, wird die BT-Anforderung abgebrochen und eine Bestätigung an den Host zurückgeleitet, die besagt, daß das BT-Zeitwerk abgelaufen ist.

Befehlssyntax

host_command	4
transaction_number	0 – 255
command_length	3 + 2 * Anzahl der Datenworte
module_slot_address	0 – 15
logical_rack_address	0 – 31
bt_data_len	Anzahl der Datenworte[1]
bt_write_data	1 – 64 Datenworte

[1] Sie können eine Zahl zwischen 0 und 63 angeben; geben Sie 0 an, bestimmt der Adapter die Datenlänge.

Dieser Befehl reiht eine Bestätigung in die Warteschlange ein, wenn der Blocktransferaustausch abgeschlossen ist, d.h., wenn er ein BT-Antwortpaket vom Adapter erhalten hat oder wenn das 4-Sekunden-Zeitwerk abgelaufen ist. Die Bestätigung sieht folgendermaßen aus:

Bestätigung des Befehls “Host BT Write”

host_command	4
transaction_number	0 – 255 (je nach Angaben des Hosts)
confirmation_status	(siehe Tabelle 5.A)
confirmation_length	0
Daten	–

Tabelle 5.A
Fehlerbedingungen für den Befehl “Host BT Write”

Fehlermnemonik	Code	Beschreibung
SUCCESS	0	Der BT-Schreibvorgang war erfolgreich.
BAD_COMMAND_DATA_LENGTH	26	Die Befehlsdatenlänge überschreitet den Höchstwert.
BT_BAD_ADDRESS	27	Die logische Rackadresse oder Steckplatzadresse liegt außerhalb des gültigen Bereichs.
BT_BAD_ADDRESS_NOT_IN_SCAN_LIST	28	Die BT-Adresse ist nicht in der Abfrageliste enthalten.
BT_BAD_DATA_LENGTH	29	Es wurden mehr als 63 Worte angegeben.
TOO_MANY_REQUESTS_FOR_MODULE	30	Dieses Modul führt bereits einen BT aus.
BT_QUEUE_FULL	31	Es stehen bereits 64 BT-Anforderungen an.
BT_REQUEST_TIMEOUT	36	Der BT-Austausch zwischen Scanner und Adapter wurde nicht innerhalb von 4 Sekunden abgeschlossen.
USER_MODULE_REQUEST_TYPE_MISMATCH	38	Der Host forderte einen Schreibvorgang an, das Modul antwortete mit einer Leseanforderung.
USER_MODULE_LENGTH_MISMATCH	39	Die von Scanner und Adapter angeforderten Längenwerte stimmen nicht überein.
MODULE_REQUESTED_TOO_MUCH_DATA	40	Das Modul forderte mehr Daten an als vom Host bereitgestellt.[1]
MODULE_IN_FAULT_GROUP_REQUESTED_BT	50	Ein BT-Schreibvorgang wurde für ein Modul angefordert, das zu einer Störungsgruppe gehört.

[1] Dieser Zustand gilt nur für “Adapter Decide Length”.

Die Routine in Beispiel 5.A weist den Scanner an, einen Host BT Write durchzuführen.

Beispiel 5.A

```
#include ”ktx_dp.h”    /* 1784–KTX scanner dualport definition */
#include ”ktx_err.h”   /* 1784–KTX scanner error codes   */
#include ”ktxconst.h” /* 1784–KTX scanner constants     */

UBYTE bt_write        (KTX_DUALPORT far *dp, UBYTE slot_number, UBYTE link_address, BT_BUFFER
*bt_buf,
            UBYTE trans_num, USHORT timeout)
{
UBYTE      status;
int        i,size;

/**** Initialize the command buffer ****/

dp–>cmd_buffer.host_command = BT_WRITE;
dp–>cmd_buffer.transaction_num = trans_num;

/**** Copy entire BT buffer if length is ****/
/**** set to ADAPTER_DECIDE_LENGTH       ****/

if (bt_buf–>count == ADAPTER_DECIDE_LENGTH)
    size = 64;
else
    size = bt_buf–>count;
dp–>cmd_buffer.command_length = 3 + (2*size);
dp–>cmd_buffer.cmd.bt_write.module_slot_address = slot_number;

/**** Convert link address to logical rack address ****/

dp–>cmd_buffer.cmd.bt_write.logical_rack_address = link_address >> 2;
dp–>cmd_buffer.cmd.bt_write.bt_data_length = bt_buf–>count;
for (i=0; i<size; i++)
    dp–>cmd_buffer.cmd.bt_write.bt_data[i] = bt_buf–>bt_data[i];

/**** Send the command ****/

dp–>int_status_from_host = SCANNER_COMMAND_FROM_HOST;
dp–>host_to_ktx_int_reg = INTERRUPT_KTX;

/**** Get and acknowledge the confirmation  ****/

status = get_confirmation(dp, trans_num, timeout);

if ((status != CONFIRMATION_TIMED_OUT) && (status != TRANS_NUM_MISMATCH))
    acknowledge_confirmation(dp);

return status;
}
```

Host BT Read

Die Host-Anwendung erteilt einen Befehl Host BT Read, wenn sie einen Blocktransfer aus einem Adapter einleiten möchte. Um die Antwort nach Abschluß des Blocktransfers auf die Anforderung abzustimmen, muß der Host folgende Parameter bereitstellen:

- die *logische Rack*-Adresse
- die Steckplatzadresse des Blocktransfermoduls
- die aus dem Modul zu lesende Datengröße
- eine unverwechselbare Transaktionsnummer

Es können jeweils bis zu 64 Blocktransferbefehle in die Warteschlange eingereiht werden,wobei jedes Blocktransfermodul nur einen BT ausführen darf (bis zu 16 BTs je logisches Rack). Der Scanner stellt ein 4-Sekunden-Zeitwerk für jede BT-Anforderung ein. Läuft das Zeitwerk ab, bevor der BT-Austausch mit dem Adapter abgeschlossen ist, wird die BT-Anforderung abgebrochen und eine Bestätigung an den Host zurückgeleitet, die besagt, daß das BT-Zeitwerk abgelaufen ist.

Befehlssyntax

host_command	5
transaction_number	0 – 255
command_length	3
module_slot_address	0 – 15
logical_rack_address	0 – 31
bt_data_len	Anzahl der Datenworte[1]

[1] Sie können eine Zahl zwischen 0 und 63 angeben; geben Sie 0 an, bestimmt der Adapter die Datenlänge.

Dieser Befehl reiht eine Bestätigung in die Warteschlange ein, wenn der Blocktransferaustausch abgeschlossen ist, d.h., wenn er ein BT-Antwortpaket vom Adapter erhalten hat oder wenn das 4-Sekunden-Zeitwerk abgelaufen ist. Die Bestätigung sieht folgendermaßen aus:

Bestätigung des Befehls “Host BT Read”

host_command	5
transaction_number	0 – 255 (je nach Angaben des Hosts)
confirmation_status	(siehe Tabelle 5.B)
confirmation_length	Anzahl der Datenbytes (sofern erfolgreich, ansonsten 0)
bt_read_data	1 – 64 Datenworte

Tabelle 5.B
Fehlerbedingungen für den Befehl “Host BT Read”

Fehlermnemonik	Code	Beschreibung
SUCCESS	0	Der BT-Lesevorgang war erfolgreich. Lesedaten sind in der Bestätigung enthalten.
BAD_COMMAND_DATA_LENGTH	26	Die Befehlsdatenlänge ist ungleich 3.
BT_BAD_ADDRESS	27	Die logische Rackadresse oder Steckplatzadresse liegt außerhalb des gültigen Bereichs.
BT_BAD_ADDRESS_NOT_IN_SCAN_LIST	28	Die BT-Adresse ist nicht in der Abfrageliste enthalten.
BT_BAD_DATA_LENGTH	29	Es wurden mehr als 63 Worte angegeben.
TOO_MANY_REQUESTS_FOR_MODULE	30	Dieses Modul führt bereits einen BT aus.
BT_QUEUE_FULL	31	Es stehen bereits 64 BT-Anforderungen an.
BT_REQUEST_TIMEOUT	36	Der BT-Austausch zwischen Scanner und Adapter wurde nicht innerhalb von 4 Sekunden abgeschlossen.
USER_MODULE_REQUEST_TYPE_MISMATCH	38	Der Host forderte einen Lesevorgang an, das Modul antwortete mit einer Schreibanforderung.
USER_MODULE_LENGTH_MISMATCH	39	Die von Host und Adapter angeforderten Längenwerte stimmen nicht überein.
BT_CHECKSUM_ERROR	41	Es wurde ein Prüfsummenfehler im Antwortpaket festgestellt. Die BT-Daten sind ungültig.

Die Routine in Beispiel 5.B weist den Scanner an, einen Host BT Read durchzuführen.

Beispiel 5.B

```
#include ”ktx_dp.h”    /* 1784–KTX scanner dualport definition */
#include ”ktx_err.h”   /* 1784–KTX scanner error codes              */
#include ”ktxconst.h” /* 1784–KTX scanner constants                */

UBYTE bt_read (KTX_DUALPORT far *dp, UBYTE slot_number, UBYTE link_address, BT_BUFFER *bt_buf,
                        UBYTE trans_num, USHORT timeout)
{
int        i;
UBYTE      status;

/**** Initialize the command buffer ****/

dp–>cmd_buffer.host_command = BT_READ;
dp–>cmd_buffer.transaction_num = trans_num;
dp–>cmd_buffer.command_length = 3;
dp–>cmd_buffer.cmd.bt_read.module_slot_address = slot_number;

/**** Convert the link address to logical rack address ****/

dp–>cmd_buffer.cmd.bt_read.logical_rack_address = link_address >> 2;
dp–>cmd_buffer.cmd.bt_read.bt_data_length = bt_buf–>count;

/**** Send the command ****/

dp–>int_status_from_host = SCANNER_COMMAND_FROM_HOST;
dp–>host_to_ktx_int_reg = INTERRUPT_KTX;

/**** Get and process the confirmation  ****/

status = get_confirmation(dp, trans_num, timeout);

switch (status) {

    /* break if no confirmation was available */

    case CONFIRMATION_TIMED_OUT:
           break;

    /* break if the transaction numbers didn’t match */

    case TRANS_NUM_MISMATCH:
           break;

    /* when successful, copy BT read data to caller  */
    /* and acknowledge the confirmation                      */

    case SUCCESS:
           bt_buf–>count = dp–>confirmation_buffer.conf_length/2;
           for (i=0; i < bt_buf–>count; i++)
               bt_buf–>bt_data[i] = dp–>confirmation_buffer.conf.bt_read.bt_data[i];
           acknowledge_confirmation(dp);
           break;

    /* if an error occurred, acknowledge the confirmation */
    /* and return the status to the caller                */

    default:
           acknowledge_confirmation(dp);
           break;

}
return status;
}
```

Ausführungszeit

Die Faktoren, die die für das Leeren der BT-Warteschlange des Scanners erforderliche Zeitspanne beeinträchtigen, sehen wie folgt aus. Sie werden im großen und ganzen in der Reihenfolge ihrer Bedeutung aufgeführt.

- die Anzahl der in die Warteschlange eingereihten BTs an denselben Adapter

  Hinweis: Ein einzelner Adapter, der mehr als acht Steckplätze aufweist und 1-Slot-Adressierung verwendet, wird vom Scanner als zwei verschiedene Adapter angesehen. In diesem Falle gehören die unteren acht Steckplätze zum ersten virtuellen Adapter.

- die Anzahl der in die Warteschlange eingereihten BTs an dasselbe Modul
- wie oft ein Adapter in der Abfrageliste des Scanners erscheint
- die Länge der Abfrageliste des Adapters
- die Anzahl der auszutauschenden BT-Datenworte
- die Baudrate

Zeitverhaltensformel

In der nachstehenden Formel wird davon ausgegangen, daß das Modul bereit ist, auf Anfrage einen Transfer durchzuführen, was häufig der Fall ist. Ist ein Modul nicht auf Anfrage zur Ausführung eines BT bereit, so muß die Zeit, die das Modul zum Fertigwerden für einen Transfer benötigt, zur Gesamtzeit hinzugerechnet werden.

T = (Effektive Anzahl der Abfragen) x (Länge der Abfrageliste) x (Zeit je Abfrage) + (Anzahl der BTs) x (Zeit je BT) + ((Anzahl der Worte) x (Zeit je Wort)),

wobei T die Zeit ist, die zum Ausführen aller in die Warteschlange eingereihten Blocktransfers erforderlich ist (unter Berücksichtigung der für die Formel geltenden Einschränkungen).

Effektive Anzahl der Abfragen: Befindet sich jeder Adapter nur einmal in der Abfrageliste des Scanners und besitzt kein Modul mehr als einen in die Warteschlange eingereihten BT, dann ist die effektive Anzahl der Abfragen die höchste Anzahl der in eine Warteschlange eingereihten BTs an einen beliebigen Adapter plus 1. Nähere Einzelheiten entnehmen Sie dem nachstehenden Abschnitt “Anzahl der Abfragen”.

Länge der Abfrageliste: Die Anzahl der Adapter in der Abfrageliste.

Zeit je Abfrage: Die zur Abfrage eines Adapters erforderliche Zeit hängt von der Baudrate ab. Bei 57,6 kBaud beträgt diese Zeit 11 ms, bei 115,2 kBaud 7 ms und bei 230,4 kBaud 4 ms.

Anzahl der BTs: Die Gesamtzahl der in die Warteschlange des Scanners eingereihten BTs.

Zeit je BT: Bei 57,6 kBaud oder 115,2 kBaud beträgt diese Zeit 5,0 ms.

Anzahl der Worte: Die Gesamtzahl der auszutauschenden BT-Datenworte, einschließlich aller in der Warteschlange befindlichen BTs.

Zeit je Wort: Bei 57,6 kBaud beträgt diese Zeit 0,3 ms und bei 115,2 kBaud 0,2 ms.

Anzahl der Abfragen

Die Gesamtzahl der Abfragen, die zum Leeren der BT-Warteschlange des Scanners erforderlich ist, entspricht der höchsten (effektiven) Anzahl der Abfragen, die zum Bearbeiten aller in die Warteschlange eingereihten BTs an einen einzelnen Adapter benötigt werden. Bestimmen Sie also die effektive Anzahl der Abfragen, die jeder einzelne Adapter benötigt, und verwenden Sie dann die höchste Zahl.

Es folgt eine Verfahrensweise zum Bestimmen der Anzahl der Abfragen, die ein einzelner Adapter benötigt. Beachten Sie, daß ein einzelner Adapter, der mehr als acht Steckplätze aufweist und 1-Slot-Adressierung verwendet, vom Scanner als zwei verschiedene Adapter angesehen wird.

Die Anzahl der Abfragen entspricht der Gesamtzahl der BTs plus 1. Sind z.B. 4 BTs an 4 Module in eine Warteschlange eingereiht, dann ist die Anzahl der Abfragen, die zum Bearbeiten aller in die Warteschlange eingereihten BTs an diesen Adapter erforderlich ist, 5 (4 BTs + 1 = 5 Abfragen).

Sie wissen jetzt die Anzahl der Abfragen, die zum Bearbeiten jedes einzelnen Adapters erforderlich ist. Um die zum Leeren der Scanner-Warteschlange erforderliche Zeit zu berechnen, benötigen Sie jedoch die effektive Anzahl der Abfragen. Die effektive Anzahl der Abfragen entspricht der Anzahl der Abfragen, dividiert durch die Häufigkeit des Auftretens des Adapters in der Abfrageliste des Scanners. Beträgt die Anzahl der zum Bearbeiten der BTs eines Adapters erforderlichen Abfragen z.B. 11 und der Adapter erscheint viermal in der Abfrageliste, dann entspricht die effektive Anzahl der zum Bearbeiten dieses Adapters erforderlichen Abfragen 2,75 (11 Abfragen / 4 Einträge = 2,75 effektive Abfragen).

Die effektive Anzahl der Abfragen, die zum Bestimmen der Gesamtzeit für das Leeren der Scanner-Warteschlange verwendet werden sollte, ist die höchste effektive Anzahl der Abfragen für einen einzelnen Adapter.

Unaufgeforderter Blocktransfer

Der Scanner setzt zu Beginn eines BT-Lese- bzw. -Schreibvorgangs ein Modulsteuerbyte (MCB) in die Ausgangsdatentafelpositionen, die der diskreten Adresse der BT-Module entsprechen.

Das Programm sollte niemals in die Ausgangsdatentafelbytes schreiben, die intelligenten E/A-Modulen entsprechen. Schreibt das Programm ein gültiges MCB in die Ausgangsdatentafel, dann geht das intelligente E/A-Modul davon aus, daß der Scanner versucht, einen BT auszuführen und antwortet dementsprechend. Der Scanner weiß jedoch nichts von einem BT, sendet dem Modul deshalb die Meldung “Nein, danke!” und setzt das Bit für unaufgeforderte BTs in operating_status. Dies weist darauf hin, daß ein Modul versucht hat, einen unaufgeforderten BT zu starten, über den der Scanner selbst nicht informiert war.

Dieses Betriebsstatusbit bleibt gesetzt, bis es vom Programm zurückgesetzt wird. Das Programm sollte dieses Betriebsbit einmal je Programmabfrage überprüfen und dieses dann ggf. zurücksetzen.

Nächster Schritt

Kapitel 6 beschreibt die Anzeige des Scannerstatus.

Kapitel 6

Anzeige des KT*x*-Scannerstatus

Kapitelinhalt

Nach dem Lesen dieses Kapitels sollten Sie in der Lage sein, einen Code zu schreiben, der:

- den aktuellen Betriebsstatus des Scanners überprüft
- den aktuellen Betriebsstatus eines beliebigen Adapters im dezentralen E/A-Verbund überprüft
- sicherstellt, daß der Scanner immer noch aktiv ist
- den Scanner informiert, daß das Programm immer noch aktiv ist

Sie sollten außerdem in der Lage sein:

- die Ursache eines nicht behebbaren Fehlers zu diagnostizieren
- LEDs (Leuchtdioden) zu interpretieren

Der Scanner verwaltet Statusinformationen im Dual-Port, die der Host zum Überwachen des Scannerzustands und des Adapterbetriebs verwenden kann. Der Scanner aktualisiert diese Informationen, wenn sich bestimmte Zustände ändern. Das Hardware-Register auf der KT*x*-Karte enthält Daten über den aktuellen Hardware-Status.

Das Wort “Operating Status”

Das Wort operating_status im Dual-Port bietet dem Host eine Zusammenfassung des Scannerbetriebs. Durch periodisches Abfragen des Wortes operating_status kann der Host bestimmen, ob irgendwelche Fehler im dezentralen E/A-Verbund oder in der KT*x*/Host-Schnittstelle aufgetreten sind. Die folgende Tabelle definiert das Wort “Operating Status”.

Bitnr.	Mnemonik	Beschreibung
0	program_mode	Bei gesetztem Bit befindet sich der Scanner im Program-Modus.
1	test_mode	Bei gesetztem Bit befindet sich der Scanner im Test-Modus.
2	run_mode	Bei gesetztem Bit befindet sich der Scanner im Run-Modus.
3	debug_mode	Für Allen-Bradley reserviert.
4	unsolicited_bt	Unaufgeforderte BT-Anforderung empfangen.
5	bt_pending	BT im Gange.
6	fault_exists	Mindestens ein Adapter weist eine Störung auf.
7	fault_changed	Ein gestörter Adapter schaltete wieder in den Online-Betrieb um.
8	unsolicited_bt_read_reply	Ein Adapter antwortete auf einen angeblichen BT-Lesebefehl, es handelte sich jedoch um einen anderen Befehlstyp.
9	unsolicited_bt_write_reply	Ein Adapter antwortete auf einen angeblichen BT-Schreibbefehl, es handelte sich jedoch um einen anderen Befehlstyp.
10	confirmation_queue_full	Bestätigungswarteschlange ist voll/über-gelaufen.
11	unknown_interrupt_type	Host sandte unbekannten Interrupttyp.
12	host_dead	Scanner erklärte Host für “tot”.
13	cos_overrun	Host bearbeitet keine COS-Interrupts.
14–15	undefined	

Adapterstatustabelle

Es gibt für jede der 128 möglichen Adapteradressen 3 lückenlose Datenbytes im Dual-Port:

- Adapter-Konfigurationsbyte
- Adapter-Störungsbyte
- Adapter-Neuversuchszählbyte

Es sei noch einmal darauf hingewiesen, daß jedes logische Rack in Viertel, sogenannte viertel Racks, unterteilt werden kann.

Adapter-Konfigurationsbyte

Der Host verwendet das Adapter-Konfigurationsbyte, um:

- zu überprüfen, ob der Adapter immer noch in der Abfrageliste enthalten ist
- zu überprüfen, ob der Adapter auf Scannerbefehle antwortet
- die physikalische Größe des Adapterracks zu bestimmen
- zwischen Netzknotenadaptern und Standardadaptern zu unterscheiden

Die nachstehende Tabelle definiert die Bits des Adapter-Konfigurationsbytes.

Bitnr.	Mnemonik	Beschreibung
0	in_scan	Im gesetzten Zustand zeigt dieses Bit an, daß sich der Adapter in der aktuellen Abfrageliste befindet.
1	exists	Im gesetzten Zustand hat der Adapter auf Scannerbefehle geantwortet.
2	known	Für interne Scannerzwecke reserviert.
3–4	size	Diese Bits geben die physikalische Gerätegröße des Adapterracks an:
		Größenbits → **Physikalische Gerätegröße**
		0 → viertel Rack
		1 → halbes Rack
		2 → dreiviertel Rack
		3 → volles Rack
5	node_adapter	Im gesetzten Zustand gibt dieses Bit an, daß der Adapter ein Netzknotenadaptergerät ist.
6–7	undefined	Für spätere Zwecke reserviert.

Adapter-Störungsbyte

Der Host verwendet das Adapter-Störungsbyte, um den aktuellen Zustand eines Adapters und sein Verhältnis zu einer störungsabhängigen Gruppe zu bestimmen. Die folgende Tabelle definiert die Bits des Adapter-Störungsbytes.

Bitnr.	Mnemonik	Beschreibung
0–3	fault_group_number	Die Störungsgruppe wird durch eine Zahl zwischen 0 und 15 gekennzeichnet.
4	in_fault_group	Im gesetzten Zustand gehört der Adapter zur Störungsgruppe.
5	adapter_online	Im gesetzten Zustand befindet sich der Adapter online, d.h. er ist im Verbund aktiv. Im rückgesetzten Zustand antwortet der Adapter nicht auf Scannerbefehle.
6	adapter_group_faulted	Im gesetzten Zustand weist mindestens ein Adapter in der Störungsgruppe einen Fehler auf.
7	undefined	Für spätere Zwecke reserviert.

Adapter-Neuversuchszählbyte

Wenn ein Adapter nicht auf einen Scannerbefehl antwortet oder wenn ein Kommunikationsfehler in der Antwort des Adapters festgestellt wird, versucht der Scanner, den Befehl erneut an den Adapter zu senden. Der Scanner protokolliert die Anzahl der Neuversuche für jeden Adapter in der Abfrageliste, indem er das Neuversuchszählbyte des Adapters inkrementiert. Ein großer Wert in diesem Byte weist auf eine unzuverlässige Kommunikation des Adapters mit dem Scanner hin.

Der Scanner versucht zweimal, einen diskreten E/A-Befehl erneut an einen Adapter zu senden. Reagiert der Adapter beim letzten Neuversuch nicht mit einer gültigen Antwort, so betrachtet der Scanner den Adapter als fehlerhaft und setzt das Bit adapter_online im Byte adapter_fault_byte zurück. Der Scanner unternimmt im Falle eines BT-Befehls an einen Adapter nur einen Neuversuch. Reagiert der Adapter nicht mit einer gültigen BT-Antwort auf den Neuversuch, informiert der Scanner den Host mittels einer Bestätigung, die besagt, daß der BT fehlgeschlagen ist. Im Falle eines fehlgeschlagenen Blocktransfers betrachtet der Scanner den Adapter jedoch nicht als fehlerhaft und ändert das Bit adapter_online nicht.

Indizieren in die Adapterstatustabelle

In Beispiel 6.A (dem Headerfile-Beispiel ktx_dp.h entnommen) ist die Adapterstatustabelle als Feld mit 128 Mitgliedern festgelegt. Jedes Mitglied besteht aus einem Adapter-Konfigurationsbyte, einem Adapter-Störungsbyte und einem Adapter-Neuversuchszählbyte. Verwenden Sie einfach die Adapteradresse zum Indizieren in das Feld. Das Code-Fragment zeigt, wie die Größe des Adapters aus der Adapterstatustabelle gelesen wird, wenn die Adresse des Adapters 12 ist.

```
size = dp->adapter_status_table[12].adapter_config_info.size;
```

Beispiel 6.A
Festlegung der Adapterstatustabelle

```
#define     UBYTE                 unsigned char
#define     UBYTE                 unsigned char

typedef struct
{
            UBYTE                 in_scan:1;
            UBYTE                 exists:1;
            UBYTE                 known:1;
            UBYTE                 size:2;
            UBYTE                 node_adapter:1;
            UBYTE                 undefined:2;
} ADAPTER_CONFIG_INFO;

typedef struct
{
            UBYTE                 fault_group_number:4;
            UBYTE                 in_fault_group:1;
            UBYTE                 adapter_online:1;
            UBYTE                 adapter_group_faulted:1;
            UBYTE                 undefined:1;
} ADAPTER_FAULT_INFO;

typedef struct
{
            ADAPTER_CONFIG_INFO   adapter_config_info;
            ADAPTER_FAULT_INFO    adapter_fault_info;
            UBYTE                 adapter_retry_count;
} ADAPTER_STATUS;

typedef struct
{
            UBYTE                 boot_code[4];
            UBYTE                 link_state;
            UBYTE                 link_address;
            UBYTE                 link_protocol;
            UBYTE                 link_baud_rate;
            UBYTE                 reserved1;
            CONFIG_DATA           config_data;
            UBYTE                 init_status;
            RUNNING_STATUS        running_status;
            UBYTE                 int_status_to_host;
            UBYTE                 int_status_from_host;
            UBYTE                 host_dead_counter;
            UBYTE                 reset_diags_counters;
            UBYTE                 reserved2[2];
            UBYTE                 alive_flag;
            UBYTE                 duplicate_node;
            UBYTE                 off_ktx;
            UBYTE                 stopped_flag;
            UBYTE                 module_state;
            UBYTE                 COS_link_address;
            UBYTE                 reserved3[100];
            UBYTE                 num_adapter_addresses;
            UBYTE                 num_faulted_adapters;
            OPERATING_STATUS      operating_status;
            /**
            ***  Declare adapter status table as an array with
128
***  members, where each member has a configuration
            ***  byte, fault byte, and retry count byte.
            **/
            ADAPTER_STATUS        adapter_status_table[128];
            COMMAND               cmd_buffer;
            CONFIRMATION          confirmation_buffer;
            UWORD                 input_image_table[256];
            UWORD                 output_image_table[256];
} KTX_DUALPORT;
```

Hardware-Statusregister der KT*x*-Karte

Das Hardware-Statusregister der KT*x*-Karte enthält den aktuellen Status der KT*x*-Hardware, einschließlich des Z84-Betriebszustands, sowie ein Bit, um anzugeben, daß der Scanner einen Host-Interrupt eingeleitet hat. Der Host kann diese Informationen jederzeit lesen und sollte dies tun, wenn ein nicht behebbarer Fehler aufgetreten ist.

Eine vollständige Beschreibung dieses Registers und seines Inhalts ist Kapitel 2, “Ein- und Abschalten”, sowie Anhang C zu entnehmen.

Verwendung der LEDs als Statusanzeigen

Die folgende Tabelle zeigt die Bedeutungen der LED-Zustände.

LED-Zustand	Beschreibung
Aus	Keine Adapter gefunden oder die KTx-Karte ist zurückgesetzt.
Stetiges grün	Es befinden sich keine fehlerhaften Adapter im dezentralen E/A-Verbund.
Blinkendes grün	Mindestens ein Adapter im dezentralen E/A-Verbund weist einen Fehler auf.
Stetiges Rot	Initialisierung im Gange ODER es ist zu einem nicht behebbaren Fehler gekommen
Blinkendes Rot	Alle Adapter im dezentralen E/A-Verbund sind fehlerhaft.

Bearbeitung von Ausnahmen/Fehlern

Der Scanner sucht ständig nach Fehlern in der KT*x*/Host-Schnittstelle, im dezentralen E/A-Verbund sowie in seiner eigenen Hardware und Firmware. Die meisten Fehler sind behebbare Fehler, d.h. der Scanner erkennt und berichtet den Fehler und setzt die Verarbeitung fort. Andere Fehler können nicht behoben werden und erfordern das Abschalten des Scanners. Wenn ein nicht behebbarer Fehler eintritt, aktualisiert der Scanner den Status im Dual-Port unter Angabe der Ursache des Fehlers, unterbricht den Host und hält dessen Prozessor an. Zu nicht behebbaren Fehlern gehören:

- Initialisierung, Fehler
- Programm-CRC, Fehler
- Dual-Port-RAM, Fehler
- Programm-RAM, Fehler
- Doppelter Netzknoten, Fehler
- Bestätigungswarteschlange, Überlauf
- Host “tot”

Initialisierung, Fehler

Während der Initialisierung überprüft der Scanner den Betrieb der KT*x*-Hardware und -Firmware, die Berechtigung des Hosts, das Scannerprotokoll auszuführen, und die ordnungsgemäße Initialisierung bestimmter Dual-Port-Variablen durch den Host. Schlägt einer dieser Tests fehl, schreibt der Scanner den Fehlercode in init_status des Dual-Ports, unterbricht den Host und hält dessen Prozessor an.

Programm-CRC, Fehler

Der Binärcode des Scanners wird im Programm-RAM, nicht im EPROM ausgeführt. Da der RAM-Speicher überschrieben werden kann, überprüft der Scanner die Integrität des Programmbinärcodes durch Ausführen eines CRC-16-Tests. Wird der Binärcode des Scanners aus irgendeinem Grund verfälscht, schreibt der Scanner den Fehlercode in module_state des Dual-Ports, unterbricht den Host und hält dessen Prozessor an.

Dual-Port-RAM, Fehler

Die Integrität der Daten im Dual-Port-RAM wird durch die Erkennung von Paritätsfehlern in der Dual-Port-Hardware geschützt. Wird ein Paritätsfehler im Dual-Port-RAM erkannt, unterbricht der Scanner den Host und hält dessen Prozessor an.

Programm-RAM, Fehler

Der Scanner überprüft die Integrität des für die Speicherung seiner internen Variablen verwendeten RAM-Speichers, indem er einen Teil des RAM-Speichers nach der Übertragung jedes dezentralen E/A-Pakets testet. Wird ein Fehler im Programm-RAM erkannt, schreibt der Scanner den Fehlercode in module_state, unterbricht den Host und hält dessen Prozessor an.

Doppelter Netzknoten, Fehler

Es wird genau ein Scanner zum Betreiben eines dezentralen E/A-Verbunds benötigt. Erkennt der KT*x*-Scanner die Anwesenheit eines anderen Scanners im Verbund, schreibt er TRUE (0x01) in duplicate_node des Dual-Ports, unterbricht den Host und hält dessen Prozessor an.

Bestätigungswarteschlange, Überlauf

Der Scanner verwaltet eine Warteschlange mit Bestätigungen von Host-Befehlen. Die Warteschlange wird geleert, während der Host Bestätigungen verarbeitet und quittiert. Erteilt der Host dem Scanner weiterhin Befehle, ohne die Bestätigungen zu verarbeiten, wird die Warteschlange mit der Zeit voll und läuft über. Der Scanner geht in diesem Fall davon aus, daß die Host-Anwendung außer Kontrolle geraten ist (ein nicht behebbarer Fehler). Der Scanner setzt das Bit confirmation_queue_full im Wort operating_status, unterbricht den Host und hält dessen Prozessor an.

Host “tot”

Der Host kann das Watchdog-Zeitwerk des Scanners aktivieren (siehe nachstehenden Abschnitt). Führt der Host das Watchdog-Handshaking mit dem Scanner nicht zeitgemäß durch, erklärt der Scanner den Host für tot. Der Scanner setzt daraufhin das Bit host_dead im Wort operating_status des Dual-Ports, unterbricht den Host und hält dessen Prozessor an.

Scanner -Watchdog

Angenommen, das Programm stürzt aufgrund logischer Fehler oder durch Eingreifen des Bedieners ab oder es geht aufgrund logischer Fehler in eine Endlosschleife über. In diesen Fällen sendet das Programm keine bedeutungsvollen Informationen mehr an den Scanner. Die Anwendung kann den Scanner so konfigurieren, daß er diese Situationen erkennt und sich beim Erkennen dieses Zustands herabgesetzt abschaltet. Dies wird durch Aktivieren des Scanner-Watchdog erzielt.

Ist der Scanner-Watchdog aktiviert, müssen der Host und der Scanner periodisches Handshaking über den Dual-Port durchführen, um ihren normalen Betrieb anzugeben. Genauso wie der Scanner einen ‘toten’ Host erkennen kann, ist der Host in der Lage, einen ‘toten’ Scanner zu erkennen. Die Schritte zur Konfiguration des Scanner-Watchdogs werden nachstehend beschrieben und durch Codebeispiele ergänzt.

Der Host aktiviert und setzt den Zeitraum für den Scanner-Watchdog, indem er einen Wert ungleich 0 in host_dead_counter des Dual-Ports schreibt. Das Schreiben einer Null in diese Speicherposition deaktiviert das Watchdog-Zeitwerk. Der Zeitraum wird anhand der nachstehenden Formel durch den in host_dead_counter geschriebenen Wert bestimmt:

*Watchdog-Zeitraum = 20 ms * Wert in* host_dead_counter

Nach dem Aktivieren des Scanner-Watchdog muß der Host Handshaking mit dem Scanner durchführen, bevor das Watchdog-Zeitwerk abläuft. Das Handshaking bewirkt ein Rücksetzen des Watchdog-Zeitwerks und den Beginn einer neuen Zeitmessung. Das Handshaking des Hosts mit dem Scanner besteht aus dem Schreiben von 00h in alive_flag des Dual-Ports. Der Scanner leitet das Handshaking durch ein alle 20 ms erfolgendes Schreiben von 02h in alive_flag des Dual-Ports zurück. Der Host kann den Inhalt von alive_flag vor dem Schreiben in alive_flag auf 02h untersuchen und überprüfen, ob der Scanner den Wert von 00h (Handshaking des Hosts) auf 02h (Handshaking des Scanners) geändert hat.

Beispiel 6.A und Beispiel 6.B zeigen das Aktivieren des Watchdog und das Schreiben einer Interrupt-Bearbeitungsroutine, die das Watchdog-Handshaking in regelmäßigen Abständen durchführt. Die Interrupt-Bearbeitungsroutine wird durch den Uhr-Interrupt des PC aktiviert.

In Beispiel 6.A konfiguriert der Host den Scanner-Watchdog für 1000 ms, indem er 50 in host_dead_counter schreibt. Der Host führt alle 500 ms ein Watchdog-Handshaking mit dem Scanner durch. Dieser große Fehlerspielraum wurde gewählt, um zu veranschaulichen, daß der Programmierer berücksichtigen muß, daß MS-DOS keine zuverlässige Echtzeit-Antwort bietet.

Der Uhr-Interrupt des Personal Computer tritt ungefähr in Intervallen von 50 ms auf. Der Uhr-Interrupt des Scanners tritt ungefähr in Intervallen von 20 ms auf. Stellen Sie den Wert host_dead_counter auf (wdg_preset * 5) ein. In Beispiel 6.A löscht der PC alive_flag, nachdem die PC-Uhr zehnmal (~500 ms) getickt hat. Der Scanner prüft alive_flag, nachdem die KT*x*-Uhr fünfzigmal (1000 ms) getickt hat. Dieser Fehlerspielraum ist absichtlich hoch, da PCs beim Ausführen von Zeitmeßaufgaben erfahrungsgemäß nicht sehr genau sind.

Beispiel 6.A
Initialisierung

```
/**** ENABLE WATCHDOG TIMER OPERATION
*****
*****                            ****/

ktx_alive = TRUE;                    /* local variable            */
oldhandler = getvect(CLOCK_INTR);    /* save old vector           */
wdg_accum = 0;                       /* clear current wgd count   */
wdg_preset = 10;                     /* Host counts 10 interrupts */
dp->host_dead_counter =
          wdg_preset * WDG_SCALER;   /* KTX counts 50             */
setvect(CLOCK_INTR, KTX_watchdog);   /* install new interrupt     */
                                     /* service routine           */
```

Die Routine in Beispiel 6.B führt das Watchdog-Handshaking mit dem Scanner durch (sofern aktiviert). Ist der Scanner tot, setzt die Routine den Uhr-Interruptverwalter zurück, druckt eine Meldung und beendet den Vorgang.

Beispiel 6.B
Interrupt-Bearbeitungsroutine

```
void      interrupt      KTX_watchdog()
{

/**** If accumulated == preset, check alive_flag */

if (++wdg_accum == wdg_preset) {
    if (dp->alive_flag != KTX_ALIVE) {

          /**** Setting this to FALSE will allow main routine ****/
          /**** to drop out of its loop and cleanup the       ****/
          /**** interrupt vectors before exiting   ****/
          ktx_alive = FALSE;
   } else {
          /**** Let KTX know that the host is still alive ****/
          dp->alive_flag = HOST_ALIVE;
          wdg_accum = 0;
   }

}
/**** Call the old routine ****/
oldhandler();
}
```

Nächster Schritt

Kapitel 7 erläutert das Aufrufen der E/A-Datentafeln des Scanners zum Lesen und Modifizieren diskreter E/A.

Kapitel 7

Erläuterung diskreter E/A

Kapitelinhalt

Dieses Kapitel erläutert das Aufrufen der E/A-Datentafeln des Scanners zum Lesen und Modifizieren diskreter E/A. Nach dem Lesen dieses Kapitels sollten Sie folgende Aufgaben ausführen können:

- Prüfen eines diskreten Eingangs
- Setzen bzw. Rücksetzen eines diskreten Ausgangs
- Prüfen eines zuvor gesetzten bzw. rückgesetzten Ausgangs

Sie sollten durch Indizieren in die E/A-Datentafeln in der Lage sein, diese Aufgaben *direkt* auszuführen.

Sie sollten außerdem folgende Aufgaben ausführen können:

- Durchführen der Funktion “Erkennung von Zustandswechseln bei Eingängen”
- Unterbrechen am Ende der Abfrageliste

Direktzugriff auf die Datentafeln

Die Ausgangs- und Eingangsdatentafeln bestehen aus zwei Integerfeldern ohne Vorzeichen mit je 256 Worten und werden mit output_image_table bzw. input_image_table bezeichnet. Sie können beide Tafeln mittels C-Zuordnungsanweisungen direkt lesen und auf die gleiche Weise direkt in die Ausgangsdatentafel schreiben.

ACHTUNG: Stellen Sie sicher, daß Sie nicht in Ausgangsdatentafelbytes schreiben, die intelligenten E/A (Blocktransfer)-Modulen entsprechen. Schreiben Sie in diese Bytes, fordern Sie aus Versehen einen Blocktransfer an. Nähere Einzelheiten hierzu finden Sie in Kapitel 5, “Erteilen von Blocktransferbefehlen”.

Zugriff nach Wort

Die Datentafeln sind Worttafeln, wobei jedes 16-Bit-Wort den 16 Anschlüssen einer Modulgruppe entspricht. Anschluß 17 oktal (15 dezimal) entspricht Bit 017 (15 dezimal) des Wortes u.s.w. bis hin zu Anschluß 0 und Bit 0. (16 Bits mal 256 Worte ergibt 4096 Ausgangsanschlüsse und 4096 Eingangsanschlüsse. Obwohl möglich, ist es höchst unwahrscheinlich, daß eine bestimmte Anwendung alle E/A-Punkte benutzt.)

Um eine bestimmte Modulgruppe in der Ausgangs- bzw. Eingangsdatentafel zu adressieren, beträgt der Index das Vierfache der Verbundadresse plus die Gruppe. Möchten Sie z.B. den 16-Bit-Inhalt von Verbundadresse 2, Gruppe 1 aus der Eingangsdatentafel zurückleiten, könnten Sie folgenden Code verwenden:

```
Wert = input_image_table[4*2+1];
```

Einzelanschluß

Zum Zugreifen auf einen einzelnen Eingangsanschluß

müssen Sie den Wortwert verschieben und ein auf 1 gesetztes Bit mittels einer AND-Operation verknüpfen, um die anderen Bits wegzumaskieren. Anschlußnummern sollten im Dezimalformat angegeben werden, bei Angabe der Anschlußnummern im Oktalformat muß eine führende Null verwendet werden.

Beispiel: Um den Wert von Anschluß 12 oktal (10 dezimal) aus Verbundadresse 1, Gruppe 5 zu erhalten, kann eine der folgenden Anweisungen verwendet werden:

```
Bit = input_image_table[4*1+5]>>012 & 1;  oder
Bit = (input_image_table[4*1+5]>>10) & 1;
```

Die erste derart kodierte Anweisung hängt von der Bedienerpriorität der C-Sprache ab: Verschiebe-Operationen werden vor AND-Operationen durchgeführt. Sie können auch Klammern hinzufügen, um die Reihenfolge der Operationen zu betonen (siehe zweite Anweisung).

Verwenden Sie einfach einen Anschlußwert in einem **if**-Test, können Sie eine Verschiebe-Operation durchführen (siehe oben) oder eine Bitmaske benutzen:

```
if ( input_image_table[4*1+5] & (1<<012) ) . . .
```

Dies sieht nicht viel besser aus als die vorhergehende Schreibweise, definieren Sie jedoch symbolische Konstanten (was zu empfehlen ist), wird der Code klarer:

```
#define SENS_RACK 1
#define SENS_GRP  5
#define SENS_BIT  012
#define SENS_MASK (1<<SENS_BIT)
      . . .
if ( input_image_table [8*SENS_RACK+SENS_GRP] & SENS_MASK ) . . .
```

Zum Setzen eines einzelnen Ausgangsanschlusses

verknüpfen Sie einfach mittels einer OR-Operation ein auf 1 gesetztes Bit in der entsprechenden Position innerhalb des Ausgangsdatentafelwortes. Es folgt ein Beispiel für Anschluß 3 von Verbundadresse 0, Gruppe 7:

```
input_image_table[4*0+7] |=1<<3;
```

Wenn der Scanner dieses Rack das nächste Mal abfragt, wird der neue Wert an die Ausgangsdatentafel des Scanners übertragen.

Das Rücksetzen (Löschen) eines Anschlusses ist etwas komplizierter

Sie müssen eine Maske, die überall – außer in der Bitposition des zu löschenden Anschlusses – auf 1 gesetzte Bits enthält, mittels einer AND-Operation verknüpfen. Der C-Operator ~ ist für diese Operationsart vorgesehen:

```
input_image_table[4*0+7] &= ~(1<<3);
```

Programmierhinweis

Sie sollten Konstanten für die Adressen Ihrer E/A-Module definieren. Dies erleichtert das Verständnis der Programme und hilft bei der Entstörung. Beispiel:

```
#define ALARM_ADDRESS 5
#define ALARM_GRP  0
#define ALARM_TERM 017
        . . .
output_image_table [4*ALARM_ADDRESS+ALARM_GRP]
|= 1<<ALARM_TERM;
```

In diesem Beispiel haben wir die Verbundadresse und die Gruppe des Moduls, das einen Alarm steuert, und den spezifischen Anschluß (beachten Sie die führende Null bei Verwendung des Oktalformates) definiert. Um einen Ausgangsanschluß einzuschalten, haben wir ein auf 1 gesetztes Bit links der Anschlußnummer verschoben und dieses mit einer OR-Operation in die Ausgangsdatentafel verknüpft.

Fragen zur Effizienz? Die meisten C-Kompilierer führen alle Rechenvorgänge mit Konstanten zur Kompilierzeit durch. Zur Laufzeit gäbe es also keinen Unterschied in bezug auf die Ausführgeschwindigkeit zwischen der obigen Anweisung und

```
output_image_table[20] |= 0x8000;
```

Die erste Methode ist jedoch auf jeden Fall leichter zu verstehen und zu verwalten.

Zeitverhalten diskreter E/A

Vergessen Sie nicht, daß das Aktualisieren der Ausgangsdatentafel keine unmittelbare Auswirkung auf die Außenwelt hat. (Obwohl sich dieser Abschnitt mit Ausgängen befaßt, gilt das gleiche Zeitverhalten für Eingänge; ein neuer Eingangswert aus dem Eingangsmodul braucht genauso lange auf dem Weg zum Programm wie ein Ausgangswert, der den entgegengesetzten Weg einschlägt.)

Das Programm aktualisiert die Ausgangsdatentafel. Nach einer Weile wendet sich der Scanner an den Adapter, dessen Daten sich geändert haben, und sendet die neuen Informationen an den Adapter.

Diese Folge von Ereignissen kann bis zu einem ganzen Durchlauf durch die Abfrageliste in Anspruch nehmen. Dieser ungünstige Fall tritt ein, wenn Sie einen Ausgang aktualisieren, nachdem der Scanner den Adapter gerade aufgefrischt hat. Es kann also einen Zyklus lang dauern, bis der Scanner den Adapter bearbeitet und den geänderten Ausgang sendet.

Wie lang ist ein Zyklus?

Multiplizieren Sie 11 Millisekunden (7 ms bei 115,2 kBaud) mit der Anzahl der Adapter in der Abfrageliste. Das Programm weiß nicht, an welcher Stelle in der Abfrageliste sich der Scanner befindet; ist das Zeitverhalten also kritisch, müssen Sie davon ausgehen, daß Ausgänge erst nach einem ganzen Durchlauf durch die Abfrageliste an die Außenwelt gelangen könnten, falls jeder Adapter nur einmal in der Liste vorkommt. Sie können diese Situation verbessern, indem Sie einen kritischen Adapter zweimal in die Abfrageliste setzen (siehe Befehl “Set Scan List” in Kapitel 4, “Erteilen von Scanner-Verwaltungsbefehlen”).

Die Routine in Beispiel 7.A schreibt Daten in die KT*x*-Ausgangsdatentafel des Dual-Ports. Versuche, über die logische Größe des Gerätes hinaus zu schreiben, werden verworfen; die aufrufende Adresse kann z.B. nicht einen Schreibvorgang für ein volles Datenrack an ein Gerät, das ein halbes Rack groß ist, anfordern.

Beispiel 7.A wurde z.T. entworfen, um zu demonstrieren, wie die Daten in der Adapterstatustabelle aufgerufen und verwendet werden können. Aus Leistungsgründen führen manche Anwendungen u.U. eine Blockübertragung von Daten zum Aktualisieren mehrerer Adapter durch. Das Überprüfen der Grenzen in diesem Beispiel bietet Ihnen einen gewissen Sicherheitsgrad bei der Adressierung der Ausgangsdatentafel, da ein versehentliches Überschreiben des Ausgangsabbilds eines anderen Adapters verhindert wird.

Beispiel 7.A

```
#include ”ktx_dp.h”    /* 1784–KTX Scanner Dualport definition */
#include ”ktx_err.h”   /* 1784–KTX Scanner error codes */
#include ”ktxconst.h”  /* 1784–KTX Scanner constants */

UBYTE put_output_data (KTX_DUALPORT far *dp,
                       UBYTE link_address, IO_BUFFER *io)
{
int         i, max_xfer_size;
UBYTE       *q;
UBYTE       status = SUCCESS;
ADAPTER_CONFIG_INFO far *p;

/**** Point to the adapter status table ****/

p = &dp–>adapter_status_table[link_address].adapter_config_info;

/**** Validate the input parameters ****/

if (link_address > 127)
    return INVALID_ADAPTER_ADDRESS;
max_xfer_size = 2 * (p–>size + 1);
if (io–>count > max_xfer_size)
    return ADAPTER_SIZE_OVERLAP;

/**** Return status of adapter to caller ****/

if (!p–>in_scan)
    status = ADDRESS_NOT_IN_SCAN_LIST;
else if (!p–>exists)
    status = ADAPTER_NONEXISTENT;

/**** Transfer the data  ****/

for (i=0; i<io–>count; i++)
    dp–>output_image_table[(link_address * 2) + i] = io–>data[i];

return status;
}
```

Die Routine in Beispiel 7.B liest Daten aus den E/A-Datentafeln der KT*x*-Karte im Dual-Port und kopiert die Ergebnisse in einen von der aufrufenden Adresse bereitgestellten Puffer. Sie berechnet die Anzahl der zu kopierenden Bytes durch Lesen der Gerätegröße aus der KT*x*-Statustabelle. Ist das Gerät fehlerhaft oder dem KT*x*-Scanner nicht bekannt, leitet der Scanner einen Fehler im Statusbyte zurück.

Beispiel 7.B

```
#include ”ktx_dp.h”    /* 1784–KTX Scanner Dualport definition */
#include ”ktx_err.h”   /* 1784–KTX Scanner error codes */
#include ”ktxconst.h”  /* 1784–KTX Scanner constants */

UBYTE get_IO_data (KTX_DUALPORT far *dp,
                   UBYTE link_address, IO_BUFFER *io,
                   TABLE_NAME io_table)
{
int         i, xfer_size;
UWORD       far *q;
UBYTE       status = SUCCESS;
ADAPTER_CONFIG_INFO far *p;

/**** Validate the input parameters ****/

if (link_address > 127)
    return INVALID_ADAPTER_ADDRESS;
if ((io_table != INPUT_TABLE) && (io_table != OUTPUT_TABLE))
    return UNKNOWN_IO_TABLE;

/**** Point to the adapter status table ****/

p = &dp–>adapter_status_table[link_address].adapter_config_info;

/**** Return status of adapter to caller ****/

if (!p–>in_scan)
    status = ADDRESS_NOT_IN_SCAN_LIST;
else if (!p–>exists)
    status = ADAPTER_NONEXISTENT;

/**** Create pointer to I/O image table and get xfer size ****/

xfer_size = 2 * (p–>size + 1);
if (io_table == INPUT_TABLE)
    q = &dp–>input_image_table[link_address * 2];
else
    q = &dp–>output_image_table[link_address * 2];

/**** Transfer the data  ****/

io–>count = xfer_size;
for (i=0; i<xfer_size; i++, q++)
    io–>data[i] = *q;

return status;
}
```

Erkennung von Zustandswechseln bei Eingängen (COS)

Diese über die Software aktivierbare/deaktivierbare Funktion bewirkt, daß der Scanner den Host unterbricht, wenn eine Änderung der Eingangsdaten aus einem Adapter festgestellt wird. Diese Funktion sollte nur dann aktiviert werden, wenn die Systemeingänge in einem bestimmten Zustand verbleiben sollen, wobei eine Änderung auf eine Ausnahmebedingung hinweist. Bei einer Anwendung mit gleichbleibenden Zuständen bedeuten *alle* Änderungen der Eingangsdaten eine Ausnahme, was bewirkt, daß der Scanner den Host unterbricht.

Aktivieren/Deaktivieren der COS-Erkennung

Aufgabenstellung	Host-Aktion
Aktivieren der COS-Erkennung	Der Host muß das Bit enable_cos_detect (Bit 2) in der Variablen config_data des Dual-Ports setzen (siehe Anhang A)
Deaktivieren der COS-Erkennung	Der Host setzt das Bit enable_cos_detect in der Variablen config_data zurück

Der Host kann dieses Bit so oft wie nötig ohne Handshaking mit dem Scanner setzen/rücksetzen.

Host/KT*x*-Handshaking

Ist die COS-Erkennung aktiviert, sollte der Host jedesmal, wenn er einen Interrupt vom KT*x*-Scanner erhält, zwei Variablen im Dual-Port auf den Interrupttyp untersuchen. Der Host sollte zuerst cos_link_address überprüfen. Ist der Wert in cos_link_address nicht FFh, trat ein COS-Interrupt auf, und der Wert in cos_link_address ist die Adapteradresse, in der es zu einer Änderung des Eingangszustands kam. Der Host sollte in diesem Falle FFh in cos_link_address schreiben, um den Empfang des COS-Interrupts zu quittieren. Quittiert der Host diesen Interrupt nicht vor dem nächsten COS-Interrupt, setzt der Scanner das Bit cos_overrun (Bit 13) in operating_status. Der Host sollte dieses Bit bei jedem Empfang eines COS-Interrupts überprüfen, um zu verizifizieren, daß er mit dem Scanner Schritt hält.

Nach dem Untersuchen auf einen COS-Interrupt sollte der Host int_status_to_host auf einen Wert ungleich 0 überprüfen. Ist ein solcher Wert vorhanden, erhielt der Host eine andere Art von Interrupt (CONFIRMATION, PROCESSING_PROBLEM, etc.) und sollte diesen Interrupt bearbeiten.

Ausnahmen

Bei der automatischen Konfiguration bzw. bei der Konfiguration einer neuen Abfrageliste aktiviert der Scanner die COS-Erkennung nicht – unabhängig vom Zustand des Bits enable_cos_detect. Wenn die automatische Konfiguration bzw. die Konfiguration der neuen Abfrageliste abgeschlossen ist, wird der normale Betrieb der COS-Erkennung wiederaufgenommen (je nach Zustand des Bits enable_cos_detect).

Zeitliche Einschränkungen

Die COS-Erkennung erfordert Echtzeit-Antworten vom Host. Bei einer Kommunikationsrate von 230 kBaud ist es möglich, daß in einem dynamischen System nur 1,2 ms zwischen COS-Interrupts liegen. Diese Interrupts werden nicht in eine Warteschlange eingereiht, so daß der Host einen COS-Interrupt verpassen könnte, falls er die zeitlichen Anforderungen nicht erfüllen kann. Die Scanner-Firmware kann erkennen, ob der Host mit COS-Interrupts Schritt hält und setzt/löscht das Bit cos_overrun (Bit 13) in operating_status des Dual-Ports (siehe Anhang A).

Interrupt am Ende der Abfrageliste

Der Host kann den Scanner so konfigurieren, daß er den Host am Ende jedes Durchlaufs durch die Abfrageliste unterbricht. Der Host aktiviert diesen Modus durch Setzen des Bits interrupt_after_each_scan (Bit 0) in config_data des Dual-Ports. Wenn der Scanner das Ende der Abfrageliste erreicht, setzt er – falls dieser Modus aktiviert ist – das Bit end_of_scan_list (Bit 0) in running_status des Dual-Ports und unterbricht den Host. Der Host muß das Bit end_of_scan_list zurücksetzen.

Nächster Schritt

Anhang	Inhalt
A	Layout des Dual-Ports
B	Programmbeispiele
C	KTx-Hardwareregister
D	Terminologie

Anhang A

Layout des Dual-Ports

Inhalt des Anhangs

Die folgende Beschreibung des Dual-Ports enthält Adressen, Mnemonik, Größe, Lese-/Schreibzugriff für Host/KT*x* sowie eine Erläuterung der Dual-Port-Variablen. Adressen :000-016H gelten für alle KT*x*-Protokolle und treffen nicht unbedingt auf das Scanner-Protokoll zu. In diesem Fall ist das Beschreibungsfeld leer.

Adresse	Mnemonik	Größe	Host	KT*x*	Beschreibung
:000H	boot_code	4	R/W	R	Muß vom Host auf hex C3 00 00 00 initialisiert werden.
:004	link_state	1			
:005	link_address	1			
:006	link_protocol	1	R/W	R	Muß vom Host auf 06h initialisiert werden.
:007	link_baud_rate	1	R/W	R	Muß vom Host auf FC (57,6K), FD (115,2K) oder FE (230,4K) initialisiert werden
:008	reserved1	1			
:009	config_data	1	R/W	R	Ermöglicht die Konfiguration des Scannerbetriebs durch den Host. Bit 0: im gesetzten Zustand unterbricht der Scanner den Host am Ende jedes Durchlaufs durch die Abfrageliste. Bit 1: reserviert. Bit 2: im gesetzten Zustand unterbricht der Scanner den Host, wenn sich die Eingangsdatentafel ändert. Bits 3–7: nicht belegt.
:00A	init_status	1	R	W	Während die Initialisierung im Gange ist, wird dieses Byte auf 1 gesetzt. Schlägt die Initialisierung fehl, wird hier ein Fehlercode geschrieben. Andernfalls wird 0 geschrieben, um eine erfolgreiche Initialisierung anzuzeigen.
:00B	running_status	1	R/W	W	Der Host löscht die folgenden Bits: Bit 0: im gesetzten Zustand befindet sich der Scanner am Ende der Abfrageliste. Bit 1: im gesetzten Zustand antwortete der Adapter einer falschen Adresse. Bits 2–7: nicht belegt.
:00C	int_status_to_host	1	R	W	Wird vom KT*x*-Scanner verwendet, um dem Host mitzuteilen, welchen Interrupttyp der KT*x*-Scanner gerade an den Host gesandt hat.
:00D	int_status_from_host	1	R/W	R/W	Wird vom KT*x*-Scanner und vom Host für das Handshaking verwendet. Der Host kann den KT*x*-Scanner mit einem von zwei Interrupttypen unterbrechen: COMMAND (01) oder CONFIRMATION_PROCESSED (02). Der KT*x*-Scanner schreibt INTERRUPT_PROCESSED (0), wenn er die Verarbeitung des Interrupts vom Host abgeschlossen hat.

Adresse	Mnemonik	Größe	Host	KT*x*	Beschreibung
:00E	host_dead_counter	1			Ist dieser Wert 0, so ist dieser Zähler deaktiviert. Ein Wert ungleich 0 definiert das Timeout-Intervall: 20 ms * Wert Der Host muß die Variable alive_flag auf 0 setzen, bevor das Intervall abläuft, oder der Scanner erklärt den Host für tot.
:00F	reserved2	3			
:012	alive_flag	1			Siehe host_dead_counter.
:013	duplicate_node	1	R	W	Wird bei Erkennen eines doppelten Scanners auf 1 gesetzt.
:014	off_ktx	1			
:015	stopped_flag	1			
:016	module_state	1	R	W	Liegt ein Laufzeitfehler (CRC- oder RAM-Test) vor, wird der Fehlercode in diese Variable geschrieben.
:017	cos_link_address	1	R/W	R/W	Aktiviert der Host die COS-Erkennung, wird die Verbundadresse des Adapters, in dem es zu einem Zustandswechsel kam, vom Scanner in diese Variable geschrieben. Der Host muß dieses Bit durch Schreiben von 0FFh rücksetzen.
:01B	nicht belegt	65H			
:07C	num_adapter_addresses	1	R	R/W	Anzahl der Adapter im Verbund
:07D	num_faulted_adapters	1	R	R/W	Anzahl der Adapter im Verbund, die derzeit gestört sind
:07E	operating_status	2	R	R/W	Dieses Byte wird vom KT*x*-Scanner verwendet, um dem Host den Zustand des Moduls durch Setzen/Rücksetzen der Bits anzuzeigen. Die Bits sind (im gesetzten Zustand) folgendermaßen definiert: Bit 0: Program-Modus Bit 1: Test-Modus Bit 2: Run-Modus Bit 3: Debug-Modus Bit 4: Freilaufender BT im Verbund empfangen Bit 5: BT anstehend Bit 6: Störung vorhanden Bit 7: Störung hat sich geändert Bit 8: Antwort auf einen freilaufenden BT-Lesevorgang Bit 9: Antwort auf einen freilaufenden BT-Schreibvorgang Bit 10: Bestätigungswarteschlange ist voll Bit 11: Unbekannter Host-Interrupttyp Bit 12: Scanner erklärte Host für tot Bit 13: Zustand der Eingangsdatentafel hat sich geändert Bits 14–15: Nicht belegt

Adresse	Mnemonik	Größe	Host	KT*x*	Beschreibung
:080	adapter_status_table	180H	R	R/W	Adapter-Konfigurationsbyte: Bit 0: in_scan Bit 1: existiert Bit 2: bekannt Bits 3–4: Größe 00 = 1/4 Rack, 01 = 1/2 Rack, 10 = 3/4 Rack, 11 = volles Rack Bit 5: node_adapter Bits 6–7: nicht definiert Adapter-Störungsbyte: Bits 0–3: fault_group_number Bit 4: in_fault_group Bit 5: adapter_online Bit 6: adapter_group_faulted Bit 7: nicht definiert Adapter-Neuversuchszählbyte Bits 0–7: Kumulativzählwert der Neuversuche an diesen Adapter Weitere Informationen finden Sie in Kapitel 6.
:200	host2ktx_msg_buf	100H	R/W	R/W	Host-an-KT*x*-Befehlspuffer
:300	ktx2host_msg_buf	100H	R/W	R/W	KT*x*-an-Host-Bestätigungspuffer
:400	input_image_table	200H	R	R/W	Eingangsdatentafel
:600	output_image_table	200H	R/W	R/W	Ausgangsdatentafel

Anhang **B**

Programmierbeispiele

Inhalt des Anhangs

Dieser Anhang enthält die folgenden Programmierbeispiele in ihrer Gesamtheit:

Hinweise zu den Beispielen

Die Beispiele in diesem Handbuch sollen den Anwendungsprogrammierer in die Host/KT*x*-Schnittstelle und -Befehle einführen. Die Beispiele demonstrieren alle Host-an-KT*x*-Befehle sowie andere häufig auszuführende Aufgaben, wie z.B.:

- Lesen der und Schreiben in die E/A-Datentafeln
- Lesen der Statustabelle
- Watchdog-Zeitwerk
- Ein- und Abschalten

Im Gegensatz zu einem typischen Anwendungsprogramm sollen diese Programmbeispiele nicht Nutzen aus der Fähigkeit der KT*x*-Karte, den Host zu unterbrechen, wenn ein Host/KT*x*-Befehl nahezu abgeschlossen ist, ziehen. Einigen Befehlen, wie z.B. Blocktransfers, gibt die KT*x*-Karte 4 Sekunden zur Ausführung ihrer Aufgaben, bevor sie einen Timeout-Fehler erklärt. Bei den meisten Anwendungen wäre es unpraktisch, die Verarbeitung so lange zu unterbrechen.

Ein Quellcode für eine Interrupt-Bearbeitungsroutine ist enthalten, um zu demonstrieren, wie eine Anwendung KT*x*-Interrupts bearbeiten würde. Die Interrupt-Bearbeitungsroutine wurde nicht mit dem Rest des Quellcode-Beispiels integriert. Diese Aufgabe ist dem Anwendungsprogrammierer überlassen.

File ktx_dp.h

```
/*************************************************************************
*  KTX_DP.H                                                              *
*                                                                        *
*  Description:                                                          *
*                                                                        *
*   This file contains defintions for all of the common structures       *
*   and type definitions used by the example 1784-KTX Scanner software.  *
*                                                                        *
*  History:                                                              *
*                                                                        *
*  02/22/93  MJG  Original creation.                                     *
*                                                                        *
*************************************************************************/

#ifndef      KTX_DP_H     /* Prevent multiple inclusions */
#define KTX_DP_H 1

/**** atomic data types ****/

#define      UBYTE                    unsigned char
#define      BYTE                     char

#define      UWORD                    unsigned short
#define      WORD                     short

#define      USHORT                   unsigned short
#define      SHORT                    short

#define      UINT                     unsigned int

#define      ULONG                    unsigned long
#define      LONG                     long

#define      BOOL                     int
#define      METACHAR                 short

/*=========================================================================

 CONFIG_DATA - defines the bits associated with the config byte.
                         The host can use this byte to control the behavior of
                         the scanner firmware.  The bits are active-high.

=========================================================================*/

typedef struct
{
              UBYTE        interrupt_after_each_scan:1;
              UBYTE        debug_flag:1;
              UBYTE        enable_cos_detect:1;
              UBYTE        undefined:5;
} CONFIG_DATA;

/*=========================================================================

 RUNNING_STATUS - defines the bits associated with the running status byte.
                         Used by the scanner firmware to notify the host of running
                         status.
=========================================================================*/

typedef struct
{
              UBYTE        end_of_scan_list:1;
              UBYTE        adapter_addr_flt:1;
              UBYTE        undefined:6;
} RUNNING_STATUS;

/*=========================================================================

  OPERATING_STATUS  - this word is part of the dual port and contains boolean
                         status information about the operation of the scanner firmware.

=========================================================================*/

typedef struct
{
              UWORD                program_mode:1;
              UWORD                test_mode:1;
              UWORD                run_mode:1;
              UWORD                debug_flag:1;
              UWORD                unsolicited_bt:1;
              UWORD                bt_pending:1;
              UWORD                fault_exists:1;
              UWORD                fault_changed:1;
              UWORD                unsolicited_bt_read_reply:1;
              UWORD                unsolicited_bt_write_reply:1;
```

```
            UWORD               confirmation_queue_full:1;
            UWORD               unknown_interrupt_type:1;
            UWORD               host_dead:1;
            UWORD               cos_overrun:1;
            UWORD               undefined:2;
} OPERATING_STATUS;

/*============================================================================

  ADAPTER_CONFIG_INFO  – this structure contains information about each
                         adapter on the Remote I/O link.  It is updated
                         during an autoconfiguration and when the host
                         changes the scan list.

============================================================================*/

typedef struct
{
            UBYTE             in_scan:1;
            UBYTE             exists:1;
            UBYTE             known:1;
            UBYTE             size:2;
            UBYTE             node_adapter:1;
            UBYTE             undefined:2;
} ADAPTER_CONFIG_INFO;

/*============================================================================

  ADAPTER_FAULT_INFO   – this structure contains fault configuration
                         and operating information and for an adapter.

============================================================================*/

typedef struct
{
            UBYTE              fault_group_number:4;
            UBYTE              in_fault_group:1;
            UBYTE              adapter_online:1;
            UBYTE              adapter_group_faulted:1;
            UBYTE              undefined:1;
} ADAPTER_FAULT_INFO;

/*============================================================================

  ADAPTER_STATUS   – this structure contains all the configuration
                         and status information for an adapter.

============================================================================*/

typedef struct
{
            ADAPTER_CONFIG_INFO                adapter_config_info;
            ADAPTER_FAULT_INFO                 adapter_fault_info;
            UBYTE                              adapter_retry_count;
} ADAPTER_STATUS;

/*============================================================================

                         COMMAND DATA STRUCTURES

============================================================================*/

typedef struct {
            UBYTE       scanner_mode;
} SET_MODE_CMD;

typedef struct {
            UBYTE       count;
            UBYTE       scan_list[64];
} SET_SCAN_LIST_CMD;

typedef SET_SCAN_LIST_CMD SCAN_LIST;

typedef struct {
            UBYTE       unused;
} AUTOCONFIGURE_CMD;

typedef struct {
            UBYTE       module_slot_address;
            UBYTE       logical_rack_address;
            UBYTE       bt_data_length;
            UWORD       bt_data[64];
} BT_WRITE_CMD;

typedef struct {
            UBYTE       module_slot_address;
            UBYTE       logical_rack_address;
```

```
            UBYTE       bt_data_length;
} BT_READ_CMD;

typedef struct {
            UBYTE       fg_number:4;
            UBYTE       in_fault_group:1;
            UBYTE       unused:3;
} FAULT_GROUP_BYTE;

typedef struct {
            FAULT_GROUP_BYTE         fault_group_data[128];
} SET_FAULT_GROUP_CMD;

typedef struct {
            UBYTE       unused;
} GET_SCAN_LIST_CMD;

typedef struct {
            UBYTE       host_command;
            UBYTE       transaction_num;
            UBYTE       command_length;
            union       {
                        SET_MODE_CMD             set_mode;
                        SET_SCAN_LIST_CMD        set_scan_list;
                        AUTOCONFIGURE_CMD        autoconfig;
                        BT_WRITE_CMD             bt_write;
                        BT_READ_CMD              bt_read;
                        SET_FAULT_GROUP_CMD      set_fault_group;
                        GET_SCAN_LIST_CMD        get_scan_list;
                        UBYTE                    padding[253];
                        } cmd;
} COMMAND;

/*============================================================================

            CONFIRMATION DATA STRUCTURES

============================================================================*/

typedef struct          {
            UBYTE       count;
            UBYTE       scan_list[64];
} GET_LIST_CONF;

typedef struct          {
            UBYTE       orphan;
} SET_LIST_CONF;

typedef     struct      {
            UWORD       bt_data[64];
} BT_READ_CONF;

typedef struct
{
            UBYTE       host_command;
            UBYTE       transaction_num;
            UBYTE       conf_status;
            UBYTE       conf_length;
            union       {
                        GET_LIST_CONF            get_list;
                        SET_LIST_CONF            set_list;
                        BT_READ_CONF             bt_read;
                        UBYTE                    dummy[252];
            } conf;
} CONFIRMATION;

/*============================================================================

  KTX_DUALPORT   – defines the structure of the dual port memory (shared by
                        both the PC Host and the KTX Z84 microprocessor).

============================================================================*/

typedef struct
{
            UBYTE                    boot_code[4];
            UBYTE                    link_state;
            UBYTE                    link_address;
            UBYTE                    link_protocol;
            UBYTE                    link_baud_rate;
            UBYTE                    reserved1;
            CONFIG_DATA              config_data;
            UBYTE                    init_status;
            RUNNING_STATUS           running_status;
            UBYTE                    int_status_to_host;
            UBYTE                    int_status_from_host;
```

```
            UBYTE                   host_dead_counter;
            UBYTE                   reset_diags_counters;
            UBYTE                   reserved2[2];
            UBYTE                   alive_flag;
            UBYTE                   duplicate_node;
            UBYTE                   off_ktx;
            UBYTE                   stopped_flag;
            UBYTE                   module_state;
            UBYTE                   COS_link_address;
            UBYTE                   reserved3[100];
            UBYTE                   num_adapter_addresses;
            UBYTE                   num_faulted_adapters;
            OPERATING_STATUS        operating_status;
            ADAPTER_STATUS          adapter_status_table[128];
            COMMAND                 cmd_buffer;
            CONFIRMATION            confirmation_buffer;
            UWORD                   input_image_table[256];
            UWORD                   output_image_table[256];

            /*
            **          The following are KTX hardware registers.  The numbers in
            **          the right column are hex offsets from the start of the dualport.
            */
             UBYTE                  reserved_reg_1;             /* :800    */
             UBYTE                  host_to_ktx_int_reg;        /* :801    */
             UBYTE                  assert_reg;                 /* :802    */
             UBYTE                  deassert_reg;               /* :803    */
             UBYTE                  status_reg;                 /* :804    */
             UBYTE                  reserved_reg_2;             /* :805    */
             UBYTE                  reserved_reg_3;             /* :806    */
             UBYTE                  reserved_reg_4;             /* :807    */
             UBYTE                  host_ack_of_ktx_int_reg;    /* :808    */
             UBYTE                  undefined_1;                /* :809    */
             UBYTE                  card_control_write_reg;     /* :80A    */
             UBYTE                  undefined_2;                /* :80B    */
             UBYTE                  card_control_read_reg;      /* :80C    */
             UBYTE                  undefine_3;                 /* :80D    */
             UBYTE                  reserved_reg_5;             /* :80E    */
             UBYTE                  undefined_4;                /* :80F    */
} KTX_DUALPORT;

/*****************************************************************************
**
**                      NON-DUALPORT STRUCTURE DEFINTIONS
**
*****************************************************************************/

typedef struct          {
            UBYTE       count;
            UWORD       data[8];
} IO_BUFFER;

typedef UBYTE           TABLE_NAME;

typedef     struct      {
            UBYTE       count;
            UWORD       bt_data[64];
} BT_BUFFER;

#endif
```

File ktx_err.h

```
/*************************************************************************
*  KTX_ERR.H                                                             *
*                                                                        *
*  Description:                                                          *
*                                                                        *
*    This file contains all errors that are configured in                *
*    the 1784-KTX Scanner firmware.                                      *
*                                                                        *
*  History:                                                              *
*                                                                        *
*  07/08/93  MDE  Original creation.                                     *
*                                                                        *
*************************************************************************/

#ifndef KTX_ERR_H
#define KTX_ERR_H 1

/*   init_status values   */
#define IN_PROGRESS                              1
#define INVALID_LINK_ADDRESS                     2
#define BINARY_PROTOCOL_MISMATCH                 3
#define INVALID_BAUD_RATE                        4
#define UNAUTHORIZED_PROTOCOL                    5
#define FAILED_PROGRAM_CRC                       6

/*   general errors   */
#define SCANNER_NOT_PROGRAM                      15
#define SCAN_LIST_TOO_LONG                       16
#define SET_SCAN_LIST_IN_PROGRESS                17
#define AUTOCONFIGURING_IN_PROGRESS              18
#define SCAN_LIST_CAUSES_FDG_ORPHAN              19
#define SC_UNKNOWN_COMMAND                       20
#define SC_BAD_REQUEST                           21
#define SC_BAD_PARAM                             22
#define SC_CANNOT_CHANGE_MODE_DURING_SET_SCAN_LIST  23
#define SC_CANNOT_CHANGE_MODE_DURING_AUTOCONFIG  24
#define SCANNER_ALREADY_AUTOCONFIGURING          25
#define BAD_COMMAND_DATA_LENGTH                  26
#define BT_BAD_ADDRESS                           27
#define BT_BAD_ADDRESS_NOT_IN_SCAN_LIST          28
#define BT_BAD_DATA_LENGTH                       29
#define TOO_MANY_REQUESTS_FOR_MODULE             30
#define BT_QUEUE_FULL                            31
#define ADDRESS_NOT_IN_SCAN_LIST                 32
#define CANNOT_SET_FG_WHILE_AUTOCONFIGURING      33
#define CANNOT_SET_FG_WHILE_SETUP_SCAN_LIST      34
#define BAD_CRC                                  35
#define BT_REQUEST_TIMEOUT                       36
#define UNSOLICITED_BT_REQUEST                   37
#define USER_MODULE_REQUEST_TYPE_MISMATCH        38
#define USER_MODULE_LENGTH_MISMATCH              39
#define MODULE_REQUESTED_TOO_MUCH_DATA           40
#define BT_CHECKSUM_ERROR                        41
#define ADAPTER_ADDRESS_ERROR                    42
#define SCANNER_ADDRESS_ERROR                    43
#define ILLEGAL_REPLY_CMD                        44
#define BUFFER_OVERFLOW_ERROR                    45
#define TIMEOUT_ERROR                            46
#define BT_BLOWOFF                               47
#define IO_MASK_DATA_MISMATCH                    48
#define INVALID_ADAPTER_ADDRESS                  49
#define MODULE_IN_FAULT_GROUP_REQUESTED_BT       50
#define ADAPTER_SIZE_OVERLAP                     51
#define BAD_BTW_REPLY                            52
#define ERROR_MORE_THAN_32_UNIQUE_DEVICES        53
#define BAD_RAM                                  54
#define ADAPTER_CONFIG_SUCCESS                   129
#define TOO_MANY_ADAPTERS_ON_LINE                130

/**** Error codes for example routines ****/
#define      INITIALIZATION_TIMED_OUT            150
#define      CONFIRMATION_TIMED_OUT              151
#define      TRANS_NUM_MISMATCH                  152
#define      ADAPTER_NONEXISTENT                 153
#define      UNKNOWN_IO_TABLE                    154

#endif
```

File ktxconst.h

```
/************************************************************************
*  KTXCONST.H                                                           *
*                                                                       *
*  Description:                                                         *
*                                                                       *
*   This file contains defintions for all of the global constants used  *
*  the 1784-KTX Scanner examples.                                       *
*                                                                       *
*  History:                                                             *
*                                                                       *
*  02/22/93  MJG  Original creation.                                    *
*                                                                       *
************************************************************************/

#ifndef KTXCONST_H
#define KTXCONST_H 1

/* boolean definitions */

#define FALSE             0
#define TRUE              1

#define OFF               0
#define ON                1

#define SUCCESS           0
#define FAIL              1

#define NO                0
#define YES               1

/*  initialization constants */

#define      SCANNER_LINK_ADDRESS      0xFD
#define      KTX_SCANNER_PROTOCOL      0x06
#define      LINK_230_KBAUD            0xFE
#define      LINK_115_KBAUD            0xFD
#define      LINK_57_KBAUD             0xFC

/*  int_status_to_host values  */

#define NO_KTX_INT_AVAILABLE          0
#define INIT_DONE_INTERRUPT           1
#define CONFIRMATION_INTERRUPT        2
#define PROCESSING_PROBLEM_INTERRUPT  3

/*  int_status_from_host values  */

#define INTERRUPT_PROCESSED             0
#define SCANNER_COMMAND_FROM_HOST       1
#define CONFIRMATION_PROCESSED          2

/*  possible command ids  */

#define SET_MODE                        1
#define SET_SCAN_LIST                   2
#define AUTOCONFIG                      3
#define BT_WRITE                        4
#define BT_READ                         5
#define SET_FAULT_GROUP                 6
#define GET_SCAN_LIST                   7

/*  modes of operation  */

#define PROGRAM                         0
#define TEST                            1
#define RUN                             2

/* number of words by rack size */

#define      QUARTER_RACK_SIZE         2
#define      HALF_RACK_SIZE            4
#define      THREE_QUARTER_RACK_SIZE   6
#define      FULL_RACK_SIZE            8
```

```
/* miscellaneous  */

#define ADAPTER_DECIDE_LENGTH         0

#define INPUT_TABLE                   1           /* point to input image table  */
#define OUTPUT_TABLE                  2           /* point to output image table */

#define COS_ACKNOWLEDGE               0xFF
#define INTERRUPT_KTX                 1
#define ACKNOWLEDGE_KTX               1


/* host/KTX watchdog values */

#define WDG_SCALER                    5           /* used to scale KTX ticks to host ticks */
#define HOST_ALIVE                    0
#define KTX_ALIVE                     2

/* timeout values */

#define      AUTO_TMO_230             4           /* Autoconfiguration can take up to  */
#define      AUTO_TMO_115             8           /* several seconds, depending on the */
#define      AUTO_TMO_57              13          /* baud rate and # of link devices   */

#define      SETSCAN_TMO_230          2           /* Configuring a new scan list       */
#define      SETSCAN_TMO_115          4           /* takes about half the time as      */
#define      SETSCAN_TMO_57           8           /* autoconfiguration                 */

#define      GET_SCAN_TMO             1           /* Get Scan List returns immediately */
#define      SET_MODE_TMO             1           /* Set Mode returns immediately      */
#define      SET_FDG_TMO              1           /* Set Fault Group returns immed.    */
#define      INIT_TMO                 2           /* Initialization takes < 2 seconds  */

#define      BT_WRITE_TMO             5           /* BTs have 4–5 seconds to complete  */
#define      BT_READ_TMO              5           /* before Scanner clears them        */
#endif
```

File ktx_xmpl.c

```
/*************************************************************************
**
**  FILE    KTX_XMPL.C
**
**  DESCRIPTION
**       This file contains the C functions to demonstrate sending commands
**           to the KTX Scanner and getting the confirmations.
**
**  USAGE
**           ktx_xmpl –aXXXX where XXXX is the base hexadecimal address of the KTX.
**
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include <conio.h>
#include <time.h>
#include "ktx_dp.h"       /* 1784–KTX Scanner Dualport definition    */
#include "ktx_err.h"      /* 1784–KTX Scanner error codes            */
#include "ktxconst.h"     /* 1784–KTX Scanner constants              */

#define CLOCK_INTR        0x1C         /* Clock interrupt vector     */

/*************************************************************************
**
**                                  Global variables
**
*************************************************************************/

KTX_DUALPORT              far          *dp;         /* pointer to KTX dualport                */
UBYTE                     wdg_preset;               /* watchdog timer preset value            */
UBYTE                     wdg_accum;                /* watchdog timer accumulated value       */
UBYTE                     ktx_alive;                /* boolean flag for local loop control    */
void interrupt            (*oldhandler)();          /* pointer to old interrupt service routine   */

void interrupt            KTX_watchdog();           /* function prototype declaration         */

/*************************************************************************
**
**  int main(int argc, char *argv[])
**
**  This routine calls a series of example subroutines which return a
**  completion status.  No effort is made to examine the status and
**  handle unsuccessful commands.
**
*************************************************************************/

int          main (int argc, char *argv[])
{
int          i;
UWORD        segment;
UBYTE        status,
             trans_count = 0;
IO_BUFFER out = {8, {0, 1, 2, 3, 4, 5, 6, 7}};
IO_BUFFER in;
BT_BUFFER bt_buffer;
FAULT_GROUP_BYTE fdg[128];

/**** Scan lists: 'auto_scan_list' buffers the scan list returned by
***** the autoconfirgure command.  'expected_scan_list' is the scan
***** list that should be returned by autoconfigure () if the link is
***** configured properly and all adapters are online.  'runtime_scan_list'
***** is the scan list that will be used during normal operation.  It has
***** duplicate link addresses to demonstrate that some adapters may be
***** scan more often if the application requires it.
****/

SCAN_LIST auto_scan_list;
SCAN_LIST expected_scan_list = {3, {0x04, 0x0E, 0x13}};
SCAN_LIST runtime_scan_list = {4, {0x04, 0x0E, 0x04, 0x13}};


/**** Extract dualport base address from command line ****/

if ((argc != 2) || (strncmp(argv[1],"–a",2) && strncmp(argv[1],"–A",2))) {
    printf("\Usage: ktxexmpl –aXXXX\n");
    exit(FAIL);
}

sscanf(&argv[1][2], "%4X", &segment);
if (segment == NULL) exit(FAIL);

dp = (KTX_DUALPORT far *) MK_FP(segment, 0x0000);
```

```
/**** Initialize the KTX scanner ****/

if (initialize_KTX (dp, LINK_57_KBAUD, INIT_TMO) != SUCCESS) {
    printf("\nScanner initialization failed.  Error code: %d",dp->init_status);
    exit(FAIL);
}

/**** Perform an autoconfiguration and check results ****/

status = autoconfigure (dp, ++trans_count, AUTO_TMO_57);
status = get_scan_list (dp, &auto_scan_list, ++trans_count, GET_SCAN_TMO);

if (auto_scan_list.count != expected_scan_list.count)
    printf ("\nAutoconfigure returned unexpected configuration.");
else
    for (i=0; i < auto_scan_list.count; i++) {
             if (auto_scan_list.scan_list[i] != expected_scan_list.scan_list[i]){
                 printf ("\nAutoconfigure returned unexpected configuration.");
                 break;
             }
    }


/**** Set up the scan list ****/

status = set_scan_list (dp, &runtime_scan_list, ++trans_count, SETSCAN_TMO_57);

/**** ENABLE /****  Set the fault dependent group  ****/
/**** First, clear the data structure ****/
/**** then place 0xe and 0x13 in the  ****/
/**** same fault dependent group.     ****/

for (i=0; i < 128; i++) {
    fdg[i].fg_number = 0;
    fdg[i].in_fault_group = FALSE;
}
fdg[0x0E].fg_number = fdg[0x13].fg_number = 1;
fdg[0x0E].in_fault_group = fdg[0x13].in_fault_group = TRUE;
status = set_fault_dependent_group (dp, (SET_FAULT_GROUP_CMD *) &fdg,
                          ++trans_count, SET_FDG_TMO);


/**** Go to TEST mode to check operation ****/

status = set_mode (dp, TEST, ++trans_count, SET_MODE_TMO);
if (dp->operating_status.test_mode != TRUE)
    printf("\nFailed to change mode to TEST.");
if (dp->operating_status.fault_exists == TRUE)
    printf("\nAt least one adapter is faulted.");

WATCHDOG TIMER OPERATION
*****
***** The PC's clock interrupt occurs at approximately 50ms intervals.
***** The KTX's clock interrupt occurs at approximately 20ms intervals.
***** Set the host_dead_counter value to (wdg_preset * 5).  In this
***** example, after 10 PC ticks (~500ms), the PC will clear the alive_flag.
***** The KTX will test the alive_flag after 50 KTX ticks (1000ms).  This
***** margin of error is intentionally high since PCs are not know to
***** be very accurate at timing tasks.
****/

ktx_alive = TRUE;                     /* local variable              */
oldhandler = getvect(CLOCK_INTR);     /* save old vector             */
wdg_accum = 0;                        /* clear current wgd count   */
wdg_preset = 10;                      /* Host counts 10 interrupts */
dp->host_dead_counter =
            wdg_preset * WDG_SCALER;  /* KTX counts 50    */
setvect(CLOCK_INTR, KTX_watchdog);    /* install new interrupt     */
                                      /* service routine        */
```

```
/*************************************************************************
**                                                                      **
**                          MAIN LOOP                                   **
**                                                                      **
**   The main loop writes data into the output image table for each of  **
**   the adapters that are in the scan list.  It also looks for a change **
**   in data for one adapter.  If a certain value appears, a BT command **
**   is initiated.  It waits for completion of the BT before continuing **
**   which is useful for these examples, but not practical in a real    **
**   application.                                                       **
**                                                                      **
*************************************************************************/

while (ktx_alive) {

    /**** Write the output image table for 0x04 (full rack) ****/

    for (i=0, out.count = FULL_RACK_SIZE; i< out.count; i++) out.data[i]++;
    status = put_output_data (dp, 0x04, &out);


  /**** Write the output image table for 0x0E (half rack) ****/

    for (i=0, out.count = HALF_RACK_SIZE; i< out.count; i++) out.data[i]++;
    status = put_output_data (dp, 0x0E, &out);


    /**** Write the output image table for 0x13 (quarter rack) ****/

    for (i=0, out.count = QUARTER_RACK_SIZE; i< out.count; i++) out.data[i]++;
    status = put_output_data (dp, 0x13, &out);


    /**** Read the input image table ****/

    status = get_IO_data(dp, 0x04, &in, INPUT_TABLE);

    /**** Initiate a BT Read and a BT Write ****/
    /**** BT Read 32 words from slot 15,    ****/
    /**** then BT write same data to slot 13 ***/
    /**** if the BT read was successful     ****/
    /**** Assume that in.data[4] can change ****/

    if (in.data[4] & 0x0100)  {
             bt_buffer.count = 32;
             status = bt_read (dp, 15, 0x04, &bt_buffer,
                         ++trans_count, BT_READ_TMO);
             if (status == SUCCESS)
                status = bt_write (dp, 13, 0x04, &bt_buffer,
                          ++trans_count, BT_WRITE_TMO);

    }
}


/**** Reset the old interrupt handler before exiting */

setvect(CLOCK_INTR, oldhandler);


/**** Turn off watchdog ****/

dp->host_dead_counter = 0;
dp->alive_flag = HOST_ALIVE;


/**** Deassert the KTX ****/

dp->deassert_reg = TRUE;

exit (FAIL);
}
```

```
/*****************************************************************************
**
**           void interrupt KTX_watchdog()
**
**           This routine performs the watchdog handshaking with the KTX
**           (if enabled).  If the KTX is dead, it resets the Clock interrupt
**           handler, printf's a message, and exits.
**
*****************************************************************************/

void         interrupt    KTX_watchdog()
{


/**** If accumulated == preset, check alive_flag */

if (++wdg_accum == wdg_preset) {
    if (dp->alive_flag != KTX_ALIVE) {

             /**** Setting this to FALSE will allow main routine ****/
             /**** to drop out of its loop and cleanup the       ****/
             /**** interrupt vectors before exiting              ****/

             ktx_alive = FALSE;

   } else {

             /**** Let KTX know that the host is still alive ****/

             dp->alive_flag = HOST_ALIVE;
             wdg_accum = 0;
   }
}

/**** Call the old routine ****/

oldhandler();
}
```

File confirm.c

```
/************************************************************************
**
**          UBYTE get_confirmation (KTX_DUALPORT far * dp,
**                                  UBYTE trans_num, USHORT timeout)
**
**          This routine waits for <timeout> seconds for a confirmation to
**          become available.  It then validates the transaction number and
**          returns status to the calling routine.  It is the caller's
**          responsibility to acknowledge the confirmation.
**
**          INPUTS
**              KTX_DUALPORT far *dp – points to the base of the KTX dualport
**              UBYTE trans_num       – transaction_number
**              USHORT timeout        – time (in seconds) to wait for completion
**
**          OUTPUT
**              UBYTE status – completion status of the command
**
************************************************************************/

#include <time.h>
#include <stdlib.h>

#include "ktx_dp.h"       /* 1784–KTX Scanner Dualport definition            */
#include "ktx_err.h"      /* 1784–KTX Scanner error codes                    */
#include "ktxconst.h"     /* 1784–KTX Scanner constants                      */


UBYTE        get_confirmation (KTX_DUALPORT far *dp, UBYTE trans_num, USHORT timeout)
{

/**** Initial values ****/

time_t       t0 = time(NULL);
UBYTE        status = SUCCESS;


/**** Wait for confirmation ****/

while (dp–>int_status_to_host != CONFIRMATION_INTERRUPT) {
    if ((time(NULL) – t0) > timeout) {
             status = CONFIRMATION_TIMED_OUT;
             break;
    }
}


/**** Check interrupt type ****/

switch (dp–>int_status_to_host) {

    /**** Handle non–confirmation types ****/

    case PROCESSING_PROBLEM_INTERRUPT :
             unrecoverable_error(dp);
             break;

    /**** Confirmations: match up transaction #'s ****/
    /**** If they match, return completion status ****/
    case CONFIRMATION_INTERRUPT :
             if (dp–>confirmation_buffer.transaction_num != trans_num)
                 status = TRANS_NUM_MISMATCH;
        else
                 status = dp–>confirmation_buffer.conf_status;

    default:
             break;
}

return status;
}
```

```
/*************************************************************************
**
**          void         acknowledge_confirmation (KTX_DUALPORT far * dp)
**
**          This routine performs the handshaking with the KTX to indicate
**          that the host has completed processing of the last confirmation.
**          responsibility to acknowledge the confirmation.
**
**          INPUTS
**              KTX_DUALPORT far *dp – points to the base of the KTX dualport
**
**          OUTPUT
**              none
**
*************************************************************************/

void         acknowledge_confirmation (KTX_DUALPORT far *dp)
{

dp->confirmation_buffer.host_command = 0;          /* zero confirmation buffer */
dp->int_status_to_host = 0;                        /* zero interrupt type      */
dp->host_ack_of_ktx_int_reg = ACKNOWLEDGE_KTX;     /* clears hardware int reg  */
dp->int_status_from_host = CONFIRMATION_PROCESSED; /* interrupt to KTX type    */
dp->host_to_ktx_int_reg = INTERRUPT_KTX;           /* interrupt the KTX        */
}
```

File init_ktx.c

```
/*************************************************************************
**
**          UBYTE   initialize_KTX (KTX_DUALPORT far *dp, UBYTE link_baud_rate,
**                                  USHORT timeout)
**
**          This routine initializes the KTX.  It assumes the 1784-KTX binary
**          has already been downloaded into the card.
**
**          PARAMETERS
**              KTX_DUALPORT far *dp  -  points to the base of the KTX dualport
**              UBYTE link_baud_rate  -  desired baud rate: 57.6, 115.2, or 230.4
**              USHORT timeout        -  time (in seconds) to wait for KTX to
**                                       complete initialization.
**
*************************************************************************/

#include <time.h>
#include <stdlib.h>

#include "ktx_dp.h"       /* 1784-KTX Scanner Dualport definition    */
#include "ktx_err.h"      /* 1784-KTX Scanner error codes            */
#include "ktxconst.h"     /* 1784-KTX Scanner constants              */


UBYTE        initialize_KTX (KTX_DUALPORT far *dp, UBYTE link_baud_rate,
                             USHORT timeout)
{
int          i;
UBYTE        status;
time_t       t0;
UBYTE        boot_code[] = {0xC3, 0, 0, 0};

/**** Initialize dualport variables ****/

for (i = 0; i < sizeof(boot_code); i++ )
    dp->boot_code[i] = boot_code[i];

dp->link_address = SCANNER_LINK_ADDRESS;
dp->link_protocol = KTX_SCANNER_PROTOCOL;
dp->link_baud_rate = link_baud_rate;


/**** Start the KTX & timer ****/

dp->assert_reg = TRUE;
t0 = time(NULL);

/**** Wait for completion ****/

while (dp->int_status_to_host != INIT_DONE_INTERRUPT) {
    if ((time(NULL) - t0) > timeout) {
        status = INITIALIZATION_TIMED_OUT;
             break;
    }
}

/**** If not a timeout, return completion status ****/

if (status != INITIALIZATION_TIMED_OUT)
    status = dp->init_status;

return status;
}
```

File put_data.c

```
/*****************************************************************************
**
**      UBYTE put_output_data (KTX_DUALPORT far *dp, UBYTE link_address, IO_BUFFER *io)
**
**           This routine writes data into the KTX Output Image Table in the
**           dualport.  Attempts to write beyond the logical size of the
**           device will be rejected, e.g., the caller requests a write of a
**           full rack of data to a device that is a half-rack in size.
**           This example was designed, in part, to demonstrate how the
**           data in the adapter status table may be accessed an used.  For
**           performance reasons, some applications may perform a block move
**           of data to update multiple adapters.  The boundary checking in
**           this example provides the user a level of security when addressing
**           the Output Image Table by preventing it from accidentally overwriting
**           another adapter's output image.
**
**           INPUT
**               KTX_DUALPORT far *dp - points to the base of the KTX dualport
**               UYBTE link_address - address of device whose data is to be written
**               IO_BUFFER *io - if successful, I/O data will be copied from this
**                                    this variable into dualport
**
**           OUTPUT
**               UBYTE status - completion status of the command
**
*****************************************************************************/

#include "ktx_dp.h"       /* 1784-KTX Scanner Dualport definition    */
#include "ktx_err.h"      /* 1784-KTX Scanner error codes            */
#include "ktxconst.h"     /* 1784-KTX Scanner constants              */


UBYTE put_output_data (KTX_DUALPORT far *dp, UBYTE link_address, IO_BUFFER *io)
{
int          i, max_xfer_size;
UBYTE        *q;
UBYTE        status = SUCCESS;
ADAPTER_CONFIG_INFO far *p;

/**** Point to the adapter status table ****/

p = &dp->adapter_status_table[link_address].adapter_config_info;


/**** Validate the input parameters ****/

if (link_address > 127)
    return INVALID_ADAPTER_ADDRESS;
max_xfer_size = 2 * (p->size + 1);
if (io->count > max_xfer_size)
    return ADAPTER_SIZE_OVERLAP;

/**** Return status of adapter to caller ****/

if (!p->in_scan)
    status = ADDRESS_NOT_IN_SCAN_LIST;
else if (!p->exists)
    status = ADAPTER_NONEXISTENT;


/**** Transfer the data  ****/

for (i=0; i<io->count; i++)
    dp->output_image_table[(link_address * 2) + i] = io->data[i];

return status;
}
```

File read_iot.c

```
/*************************************************************************
**
**      UBYTE get_IO_data (KTX_DUALPORT far *dp, UBYTE link_address,
**                         IO_BUFFER *io, TABLE_NAME io_table)
**
**           This routine reads data from the KTX I/O Image Tables in the dualport
**           and copies the results into a buffer supplied by the caller.  It
**           calculates the number of bytes to copy by reading the size of the
**           device from the KTX status table.  If the device is faulted or is not
**           known by the KTX scanner, it returns a error in the status byte.
**
**           INPUT
**               KTX_DUALPORT far *dp – points to the base of the KTX dualport
**               UYBTE link_address – address of device whose data is to be read
**               IO_BUFFER *io – if successful, I/O data will be assigned to
**                                    this variable
**               TABLE_NAME io_table – either INPUT_TABLE or OUTPUT_TABLE,
**                                     which is used to create a pointer to the
**                                     right table
**
**           OUTPUT
**               UBYTE status – completion status of the command
**
*************************************************************************/

#include ”ktx_dp.h”       /* 1784–KTX Scanner Dualport definition    */
#include ”ktx_err.h”      /* 1784–KTX Scanner error codes            */
#include ”ktxconst.h”     /* 1784–KTX Scanner constants              */


UBYTE get_IO_data (KTX_DUALPORT far *dp, UBYTE link_address, IO_BUFFER *io,
                                     TABLE_NAME io_table)
{
int          i, xfer_size;
UWORD        far *q;
UBYTE        status = SUCCESS;
ADAPTER_CONFIG_INFO far *p;


/**** Validate the input parameters ****/

if (link_address > 127)
    return INVALID_ADAPTER_ADDRESS;
if ((io_table != INPUT_TABLE) && (io_table != OUTPUT_TABLE))
    return UNKNOWN_IO_TABLE;


/**** Point to the adapter status table ****/

p = &dp->adapter_status_table[link_address].adapter_config_info;


/**** Return status of adapter to caller ****/

if (!p->in_scan)
    status = ADDRESS_NOT_IN_SCAN_LIST;
else if (!p->exists)
    status = ADAPTER_NONEXISTENT;

/**** Create pointer to I/O image table and get xfer size ****/

xfer_size = 2 * (p->size + 1);
if (io_table == INPUT_TABLE)
    q = &dp->input_image_table[link_address * 2];
else
    q = &dp->output_image_table[link_address * 2];


/**** Transfer the data  ****/

io->count = xfer_size;
for (i=0; i<xfer_size; i++, q++)
    io->data[i] = *q;


return status;
}
```

File set_mode.c

```
/*****************************************************************************
**
**          UBYTE set_mode (KTX_DUALPORT far * dp, UBYTE mode,
**                                  UBYTE trans_num, USHORT timeout)
**
**          This routine set the KTX scanner mode.
**
**          INPUTS
**             KTX_DUALPORT far *dp  – points to the base of the KTX dualport
**             UBYTE mode            – PROGRAM, TEST, or RUN mode
**             UBYTE trans_num       – transaction_number
**             USHORT timeout        – time (in seconds) to wait for completion
**
**          OUTPUT
**             UBYTE status          – completion status of the command
**
*****************************************************************************/

#include ”ktx_dp.h”          /* 1784–KTX scanner dualport definition    */
#include ”ktx_err.h”         /* 1784–KTX scanner error codes            */
#include ”ktxconst.h”        /* 1784–KTX scanner constants              */

UBYTE            set_mode (KTX_DUALPORT far *dp, UBYTE mode,
                                  UBYTE trans_num, USHORT timeout)
{
UBYTE            status;

/**** Initialize the command buffer ****/

dp–>cmd_buffer.host_command = SET_MODE;
dp–>cmd_buffer.transaction_num = trans_num;
dp–>cmd_buffer.command_length = 1;
dp–>cmd_buffer.cmd.set_mode.scanner_mode = mode;

/**** Send the command ****/

dp–>int_status_from_host = SCANNER_COMMAND_FROM_HOST;
dp–>host_to_ktx_int_reg = INTERRUPT_KTX;

/**** Get and acknowledge the confirmation  ****/

status = get_confirmation(dp, trans_num, timeout);

if ((status != CONFIRMATION_TIMED_OUT) && (status != TRANS_NUM_MISMATCH))
    acknowledge_confirmation(dp);

return status;
}
```

File setscanl.c

```
/*****************************************************************************
**
**      UBYTE set_scan_list (KTX_DUALPORT far *dp, SCAN_LIST *sl, UBYTE *orphan,
**                                   UBYTE trans_num, USHORT timeout)
**
**         This routine instructs the KTX scanner to replace the existing scan
**         list with the new one specified in the command.  If the new scan
**         list would create a fault dependent groupt (FDG) orphan, the orphan's
**         address is copied to the caller in addition to the status code.
**
**         INPUT
**            KTX_DUALPORT far *dp     - points to the base of the KTX dualport
**            SCAN_LIST *sl            - the scan list to be sent to the KTX
**            UBYTE *orphan            - where orphan's address is to be written
**            UBYTE trans_num          - transaction_number
**            USHORT timeout           - time (in seconds) to wait for completion
**
**         OUTPUT
**            UBYTE status             - completion status of the command
**
*****************************************************************************/

#include "ktx_dp.h"          /* 1784-KTX scanner dualport definition    */
#include "ktx_err.h"         /* 1784-KTX scanner error codes            */
#include "ktxconst.h"        /* 1784-KTX scanner constants              */

UBYTE set_scan_list (KTX_DUALPORT far *dp, SCAN_LIST *sl, UBYTE *orphan,
                                  UBYTE trans_num, USHORT timeout)
{
UBYTE            status;

/**** Initialize the command buffer ****/

dp->cmd_buffer.host_command = SET_SCAN_LIST;
dp->cmd_buffer.transaction_num = trans_num;
dp->cmd_buffer.command_length = 1 + sl->count;
dp->cmd_buffer.cmd.set_scan_list = *sl;

/**** Send the command ****/

dp->int_status_from_host = SCANNER_COMMAND_FROM_HOST;
dp->host_to_ktx_int_reg = INTERRUPT_KTX;

/**** Get and process the confirmation  ****/

status = get_confirmation(dp, trans_num, timeout);

switch (status) {

    /* break if no confirmation was available */

    case CONFIRMATION_TIMED_OUT:
                break;

    /* break if the transaction numbers didn't match */

    case TRANS_NUM_MISMATCH:
                break;

    /* If an adapter became a FDG orphan, copy        */
    /* orphan's address to caller.*/
    /* (fall through to acknowledge confirmation) */

    case SCAN_LIST_CAUSES_FDG_ORPHAN:
                *orphan = dp->confirmation_buffer.conf.set_list.orphan;
                acknowledge_confirmation(dp);
                break;
```

```
    /* if a different error occurred or SUCCESS, acknowledge */
    /* the confirmation and return the status to the caller  */

    case SUCCESS:
    default:
                acknowledge_confirmation(dp);
                break;
}
return status;
}
```

File autoconf.c file

```
/*****************************************************************************
**
**          UBYTE autoconfigure (KTX_DUALPORT far *dp, UBYTE trans_num,
**                                    USHORT timeout)
**
**          This routine instructs the KTX scanner to perform an
**          autoconfiguration and waits for a confirmation.  An autoconfigure
**          command requires the scanner to poll all 128 possible device
**          addresses on the RIO link, which can take as long as 6 seconds
**          when operating at 57.6 Kbaud.
**
**          INPUTS
**             KTX_DUALPORT far *dp    – points to the base of the KTX dualport
**             UBYTE trans_num         – transaction_number
**             USHORT timeout          – time (in seconds) to wait for completion
**
**          OUTPUT
**             UBYTE status            – completion status of the command
**
*****************************************************************************/

#include ”ktx_dp.h”          /* 1784–KTX scanner dualport definition    */
#include ”ktx_err.h”         /* 1784–KTX scanner error codes            */
#include ”ktxconst.h”        /* 1784–KTX scanner constants              */

UBYTE       autoconfigure (KTX_DUALPORT far *dp, UBYTE trans_num, USHORT timeout)
{
UBYTE       status;

/**** Initialize the command buffer ****/

dp–>cmd_buffer.host_command = AUTOCONFIG;
dp–>cmd_buffer.transaction_num = trans_num;
dp–>cmd_buffer.command_length = 0;

/**** Send the command ****/

dp–>int_status_from_host = SCANNER_COMMAND_FROM_HOST;
dp–>host_to_ktx_int_reg = INTERRUPT_KTX;

/**** Get and acknowledge the confirmation  ****/

status = get_confirmation(dp, trans_num, timeout);

if ((status != CONFIRMATION_TIMED_OUT) && (status != TRANS_NUM_MISMATCH))
    acknowledge_confirmation(dp);

return status;
}
```

File bt _write.c

```
/*****************************************************************************
**
**          UBYTE bt_write (KTX_DUALPORT far * dp, UBYTE slot_number,
**                                      UBYTE link_address, BT_BUFFER *bt_buf,
**                                      UBYTE trans_num, USHORT timeout)
**
**          This routine instructs the KTX scanner to perform a
**          Block Transfer Write.
**
**          INPUTS
**             KTX_DUALPORT far *dp     – points to the base of the KTX dualport
**             UBYTE slot_number        – BT module slot number (0–15)
**             UBYTE link_address       – device link address
**             BT_BUFFER *bt_buf        – BT data buffer
**             UBYTE trans_num          – transaction_number
**             USHORT timeout           – time (in seconds) to wait for completion
**
**          OUTPUT
**             UBYTE status             – completion status of the command
**
*****************************************************************************/

#include ”ktx_dp.h”          /* 1784–KTX scanner dualport definition    */
#include ”ktx_err.h”         /* 1784–KTX scanner error codes            */
#include ”ktxconst.h”        /* 1784–KTX scanner constants              */


UBYTE bt_write (KTX_DUALPORT far *dp, UBYTE slot_number, UBYTE link_address,
                   BT_BUFFER *bt_buf, UBYTE trans_num, USHORT timeout)
{
UBYTE       status;
int         i,size;

/**** Initialize the command buffer ****/

dp->cmd_buffer.host_command = BT_WRITE;
dp->cmd_buffer.transaction_num = trans_num;

/**** Copy entire BT buffer if length is ****/
/**** set to ADAPTER_DECIDE_LENGTH       ****/

if (bt_buf->count == ADAPTER_DECIDE_LENGTH)
    size = 64;
else
    size = bt_buf->count;
dp->cmd_buffer.command_length = 3 + (2*size);
dp->cmd_buffer.cmd.bt_write.module_slot_address = slot_number;

/**** Convert link address to logical rack address ****/

dp->cmd_buffer.cmd.bt_write.logical_rack_address = link_address >> 2;
dp->cmd_buffer.cmd.bt_write.bt_data_length = bt_buf->count;
for (i=0; i<size; i++)
    dp->cmd_buffer.cmd.bt_write.bt_data[i] = bt_buf->bt_data[i];

/**** Send the command ****/

dp->int_status_from_host = SCANNER_COMMAND_FROM_HOST;
dp->host_to_ktx_int_reg = INTERRUPT_KTX;

/**** Get and acknowledge the confirmation  ****/

status = get_confirmation(dp, trans_num, timeout);

if ((status != CONFIRMATION_TIMED_OUT) && (status != TRANS_NUM_MISMATCH))
    acknowledge_confirmation(dp);

return status;
}
```

File bt _read.c

```
/*************************************************************************
**
**          UBYTE bt_read (KTX_DUALPORT far * dp, UBYTE slot_number,
**                                      UBYTE link_address, BT_BUFFER *bt_buf,
**                                      UBYTE trans_num, USHORT timeout)
**
**          This routine instructs the KTX scanner to perform a Block Transfer
**          Read.
**
**          INPUTS
**             KTX_DUALPORT far *dp     – points to the base of the KTX dualport
**             UBYTE slot_number        – BT module slot number (0–15)
**             UBYTE link_address       – device link address
**             BT_BUFFER *bt_buf        – BT data buffer which is to be written on
**                                        completion of the BT
**             UBYTE trans_num          – transaction_number
**             USHORT timeout           – time (in seconds) to wait for completion
**
**          OUTPUT
**             UBYTE status             – completion status of the command
**
*************************************************************************/

#include ”ktx_dp.h”       /* 1784–KTX scanner dualport definition */
#include ”ktx_err.h”      /* 1784–KTX scanner error codes         */
#include ”ktxconst.h”     /* 1784–KTX scanner constants           */

UBYTE bt_read (KTX_DUALPORT far *dp, UBYTE slot_number, UBYTE link_address,
                          BT_BUFFER *bt_buf, UBYTE trans_num, USHORT timeout)
{
int          i;
UBYTE        status;


/**** Initialize the command buffer ****/

dp–>cmd_buffer.host_command = BT_READ;
dp–>cmd_buffer.transaction_num = trans_num;
dp–>cmd_buffer.command_length = 3;
dp–>cmd_buffer.cmd.bt_read.module_slot_address = slot_number;

/**** Convert the link address to logical rack address ****/

dp–>cmd_buffer.cmd.bt_read.logical_rack_address = link_address >> 2;
dp–>cmd_buffer.cmd.bt_read.bt_data_length = bt_buf–>count;


/**** Send the command ****/

dp–>int_status_from_host = SCANNER_COMMAND_FROM_HOST;
dp–>host_to_ktx_int_reg = INTERRUPT_KTX;

/**** Get and process the confirmation  ****/

status = get_confirmation(dp, trans_num, timeout);

switch (status) {

    /* break if no confirmation was available */

    case CONFIRMATION_TIMED_OUT:
             break;

    /* break if the transaction numbers didn’t match */

    case TRANS_NUM_MISMATCH:
             break;

    /* when successful, copy BT read data to caller  */
    /* and acknowledge the confirmation              */

    case SUCCESS:
             bt_buf–>count = dp–>confirmation_buffer.conf_length/2;
             for (i=0; i < bt_buf–>count; i++)
                 bt_buf–>bt_data[i] = dp–>confirmation_buffer.conf.bt_read.bt_data[i];
             acknowledge_confirmation(dp);
             break;
```

```
    /* if an error occurred, acknowledge the confirmation */
    /* and return the status to the caller                */

    default:
             acknowledge_confirmation(dp);
             break;

}
return status;
}
```

File set_fdg.c

```
/*****************************************************************************
**
**      UBYTE set_fault_dependent_group (KTX_DUALPORT far *dp,
**            SET_FAULT_GROUP_CMD *fdg, UBYTE trans_num, USHORT timeout)
**
**         This routine instructs the KTX scanner to replace the existing
**         fault group list with the new one specified in the command.
**
**         INPUT
**            KTX_DUALPORT far *dp        – points to the base of the KTX dualport
**            SET_FAULT_GROUP_CMD *fdg    – points to fault dependent group data
**            UBYTE trans_num             – transaction_number
**            USHORT timeout              – time (in seconds) to wait for completion
**
**         OUTPUT
**            UBYTE status                – completion status of the command
**
*****************************************************************************/

#include ”ktx_dp.h”         /* 1784–KTX scanner dualport definition */
#include ”ktx_err.h”        /* 1784–KTX scanner error codes         */
#include ”ktxconst.h”       /* 1784–KTX scanner constants           */

UBYTE set_fault_dependent_group (KTX_DUALPORT far *dp, SET_FAULT_GROUP_CMD *fdg,
                                 UBYTE trans_num, USHORT timeout)
{
UBYTE            status;

/**** Initialize the command buffer ****/

dp–>cmd_buffer.host_command = SET_FAULT_GROUP;
dp–>cmd_buffer.transaction_num = trans_num;
dp–>cmd_buffer.command_length = sizeof(SET_FAULT_GROUP_CMD);
dp–>cmd_buffer.cmd.set_fault_group = *fdg;

/**** Send the command ****/

dp–>int_status_from_host = SCANNER_COMMAND_FROM_HOST;
dp–>host_to_ktx_int_reg = INTERRUPT_KTX;

/**** Get and process the confirmation  ****/

status = get_confirmation(dp, trans_num, timeout);

if ((status != CONFIRMATION_TIMED_OUT) && (status != TRANS_NUM_MISMATCH))
    acknowledge_confirmation(dp);

return status;
}
```

File getscanl.c

```
/*************************************************************************
**
**      UBYTE get_scan_list (KTX_DUALPORT far *dp, SCAN_LIST *sl,
**                                   UBYTE trans_num, USHORT timeout)
**
**         This routine instructs the KTX scanner to return the current
**         scan list.
**
**         INPUT
**            KTX_DUALPORT far *dp    - points to the base of the KTX dualport
**            SCAN_LIST *sl           - if successful, the scan list will be
**                                      assigned to this variable
**            UBYTE trans_num         - transaction_number
**            USHORT timeout          - time (in seconds) to wait for
**
**         OUTPUT
**            UBYTE status            - completion status of the command
**
*************************************************************************/

#include "ktx_dp.h"       /* 1784-KTX scanner dualport definition */
#include "ktx_err.h"      /* 1784-KTX scanner error codes         */
#include "ktxconst.h"     /* 1784-KTX scanner constants           */

UBYTE get_scan_list (KTX_DUALPORT far *dp, SCAN_LIST *sl,
                         UBYTE trans_num, USHORT timeout)
{
int          i;
UBYTE        status;

/**** Initialize the command buffer ****/

dp->cmd_buffer.host_command = GET_SCAN_LIST;
dp->cmd_buffer.transaction_num = trans_num;
dp->cmd_buffer.command_length = 0;

/**** Send the command ****/

dp->int_status_from_host = SCANNER_COMMAND_FROM_HOST;
dp->host_to_ktx_int_reg = INTERRUPT_KTX;

/**** Get and process the confirmation  ****/

status = get_confirmation(dp, trans_num, timeout);

switch (status) {

    /* break if no confirmation was available */

    case CONFIRMATION_TIMED_OUT:
             break;

    /* break if the transaction numbers didn't match */

    case TRANS_NUM_MISMATCH:
             break;

    /* when successful, copy the scan list to the   */
    /* caller and acknowledge the confirmation      */

    case SUCCESS:
             sl->count = dp->confirmation_buffer.conf.get_list.count;
             for (i=0; i<sl->count; i++)
                 sl->scan_list[i] =
                     dp->confirmation_buffer.conf.get_list.scan_list[i];
             acknowledge_confirmation(dp);
             break;

    /* if an error occurred, acknowledge the confirmation */
    /* and return the status to the caller           */

    default:
             acknowledge_confirmation(dp);
             break;
}
return status;
}
```

File ktx_int.c

```
/*************************************************************************
**
**      FILE – ktx_int.c
**
**      This file contains a routine for initializing and enabling
**      interrupts to service KTX interrupts and a routine that is a
**      boilerplate for a KTX scanner interrupt service routine
**
*************************************************************************/

#include <dos.h>
#include <assert.h>

#include ”ktx_dp.h”     /* 1784–KTX Scanner Dualport definition */
#include ”ktx_err.h”    /* 1784–KTX Scanner error codes         */
#include ”ktxconst.h”   /* 1784–KTX Scanner constants           */

#define LowPICPort 0x20           /* Port address for Prog. Interrupt Controller for ints 0–7 */
#define HighPICPort 0xA0          /* Port address for Prog. Interrupt Controller for ints 8–15 */
#define EOI 0x20                  /* End Of Interrupt to PIC */
#define INT_PENDING_BIT 0x01      /* bit mask for interrupt pending bit in status register */

extern KTX_DUALPORT far *dp;                    /* pointer to KTX’s dualport */

/*************************************************************************
**
**      void Init_interrupt (unsigned char InterruptNumber, void interrupt
**                           (*InterruptServiceRoutine)())
**
**      This routine sets the interrupt vector and enables interrupts.
**      The host passes in two parameters: the interrupt number and
**      the interrupt service routine that will handle KTX interrupts.
**      The interrupt number must match the interrupt number selected
**      for this channel using the jumper on the KTX hardware.
**
**      INPUTS
**          KTX_DUALPORT far *dp – points to the base of the KTX dualport
**          UBYTE trans_num      – transaction_number
**          USHORT timeout       – time (in seconds) to wait for completion
**
**      OUTPUT
**          Interrupt Vector for KTX installed
**
*************************************************************************/


void interrupt (*OldInterruptServiceRoutine)();      /* pointer to store old interrupt handler */

void InitInterrupt(unsigned char InterruptNumber, void interrupt (*InterruptServiceRoutine)())
{
unsigned char    PICMask,                       /* storage for prog. interrupt controller mask */
                 PICPortBase,                   /* Port address of either first PIC or second PIC */
                 SWInterruptNumber;             /* Software Interrupt number, converted from HW */

                 /* Insure that InterruptNumber is valid one for KTX – leave in for debug only */

                 assert((InterruptNumber > 2) && (InterruptNumber < 16));

                 /* Set the base port of the Programmable Interrupt Controller depending on
                    whether the interrupt is on the first or second PIC */

                 if (InterruptNumber < 8) PICPortBase = LowPICPort;
                 else PICPortBase = HighPICPort;

                 /* Convert from hardware interrupt numbers (0–7) to software interrupt numbers
                    0x8–0xf and hardware interrupt numbers (8–15) to software interrupt
                    numbers 0x70–0x78  */
```

```
                SWInterruptNumber = InterruptNumber + 0x8;
                if (SWInterruptNumber > 0xf) SWInterruptNumber += 0x60;

                /* Save the old interrupt handler vector, and install the new one */
                /* Need to disable interrupts when changing vectors */

                disable ();
                OldInterruptServiceRoutine = getvect(SWInterruptNumber);
                setvect(SWInterruptNumber, InterruptServiceRoutine);

                /* Enable the interrupt by setting the appropriate bit in the PICMask to 0 */

                PICMask = inportb(PICPortBase+1);
                delay(50);                      /* Delay for PIC to settle */
                outportb(PICPortBase+1, PICMask & ~(1 << (InterruptNumber % 8)));
                delay(50);                      /* Delay for PIC to settle */
                enable ();                      /* Re-enable interrupts */
}

/*************************************************************************
**
**      void InterruptServiceRoutine (void)
**
**      This routine is an example of how a host might want to service
**      interrupts from a KTX.  It assumes that the host has enabled the KTX
**      scanner to interrupt on input change-of-state and at the end of each
**      pass through the scan list.  It assumes that more than one interrupt
**      can be received at a time, i.e., that because the KTX interrupts are
**      asynchronous to each other there may be more than one interrupt
**      pending.
**
**      INPUTS
**          KTX dualport and hardware status register.
**
**      OUTPUT
**          KTX dualport and hardware status changed
**
*************************************************************************/

void interrupt InterruptServiceRoutine(void)
{

UBYTE   temp;

/*=======================================================================
**
**      Verify it was the KTX that caused the interrupt by reading the
**      status register.
**
=======================================================================*/
/**** Bit 0 of status reg == 1 when KTX ****/
/**** is the source of a host interrupt ****/

if (!(dp->status_reg & 1))
    return;

/**** Clear bit 0 of the status reg by ****/
/**** writing to acknowledge register. ****/

dp->host_ack_of_ktx_int_reg = ACKNOWLEDGE_KTX;


/*=======================================================================
**
**      Service the COS interrupt first, since it can happen most often...
**              1) make sure it's enabled
**              2) if so, see if we got a COS interrupt
**              3) if so, acknowledge it and check for COS overrun
**
=======================================================================*/
if (dp->config_data.enable_cos_detect &&
        (dp->COS_link_address != COS_ACKNOWLEDGE)) {

    temp = dp->COS_link_address ;
    dp->COS_link_address = COS_ACKNOWLEDGE;
    if (dp->operating_status.cos_overrun) {

        /**** When this bit is set, it indicates the host is unable
```

```
        ***** to keep up with COS interrupts, i.e., the system input
        ***** state is changing more rapidly than the host is able to
        ***** process.  COS interrupts should be enabled only in
        ***** systems that have infrequent state changes.
        ****/
    }
    /**** The application programmer should write a routine that
    ***** services the COS.  The handle_input_COS routine called
    ***** below is not one of the supplied examples.
    ****/

    handle_input_COS (temp);

}

/*=============================================================================
**
**      Service the end-of-scan-list interrupt next, since it is also
**      time-critical.
**              1) make sure it's enabled
**              2) if so, see if we got an end-of-scan-list interrupt
**              3) if so, acknowledge it
**
=============================================================================*/

if (dp->config_data.interrupt_after_each_scan
        && dp->running_status.end_of_scan_list) {

    /**** Acknowledge the end_of_scan_list interrupt ****/

    dp->running_status.end_of_scan_list = FALSE;

    /**** The application programmer may want to write a routine that
    ***** services the I/O image tables whenever a end_of_scan_list
    ***** interrupt occurs.  The designer should keep in mind that
    ***** links operating at a high baud rate and with short scan lists
    ***** are likely to cause frequent interrupts.  The host should be
    ***** capable of handling these interrupts in a timely fashion.
    ****/
}

/*=============================================================================
**
**      Service the regular interrupt next, since it is queued
**              1) see what type it is
**              2) service by type...
**
=============================================================================*/

/**** dp->int_status_to_host is the interrupt type ****/

switch (dp->int_status_to_host) {

        /**** Host ack'd last interrupt ****/

        case NO_KTX_INT_AVAILABLE :
                break;


        /**** Initialization is complete ****/

        case INIT_DONE_INTERRUPT :

                /**** Check completion status    ****/

                if (dp->init_status != SUCCESS)
                    unrecoverable_error(dp);

                /**** If good, clear interrupt type ****/

                else
                   dp->int_status_to_host = 0;
                break;


        /**** A confirmation is available ****/

        case CONFIRMATION_INTERRUPT :

                /**** The application programmer needs to write a routine
```

```
                ***** that handles confirmations.  The other examples
                ***** provided show how to do it when the application
                ***** waits for a confirmation before proceeding to the
                ***** next command.  An asynchronous version would need
                ***** to match up confirmations with pending commands
                ***** using the transaction number and acknowledge the
                ***** confirmation.
                ****/

                break;


        /**** A processing problem is an unrecoverable error ****/

        case PROCESSING_PROBLEM_INTERRUPT  :
                unrecoverable_error(dp);
                break;

        /**** Unknown interrupt type ****/

        default :
                break;
}

}


           /* Need to issue the End Of Interrupt to the PIC controller(s) */


           outportb(LowPICPort, EOI);
           if (InterruptNumber > 7) outportb(HighPICPort, EOI); /* if int > 7 then second PIC also */

}
```

File load.c

```
/*****************************************************************************
**
**  FILE – load.c
**
**  This file contains a routine for downloading the scanner binary into
**  KTx memory.
**
 *****************************************************************************/

#include<stdio.h>
#include<mem.h>
#include<dos.h>
#include <fcntl.h>
#include <io.h>
#include<process.h>
#include<string.h>
#include <conio.h>

#ifndef FALSE
#define FALSE 0
#define TRUE ~FALSE
#endif

/*****************************************************************************
   GENERAL KTX DUAL PORT OFFSETS
 *****************************************************************************/
#define KTX_RELEASE 0x802   /* Release the KTX (start it running) */
#define KTX_RESET   0x803   /* Reset the KTX (stop it) */

/*****************************************************************************
   LOADPCL.BIN DUAL PORT OFFSETS
 *****************************************************************************/
#define LOADPCL_DATA_WINDOW 0x80
#define LOADPCL_HANDSHAKE   0x75
#define LOADPCL_SIZE        0x72
#define LOADPCL_LENGTH      0x70
#define LOADPCL_STATUS      0x74

void usage(char *argv[]);

int main(int argc, char *argv[])
{
  char far *base;
  int address = 0,length,firstpass = 1;
  int i,loadfile,ifile,file_into_dp = FALSE;
  unsigned char loadfile_name[80] = {""};

  /* Parse the command line for valid switches:
        –fnnnnnnnn.eee  –> file name to download
        –d              –> load file directly to dual port (diagnostics)
        –axx            –> xx = upper byte of address of KTX (D3 e.g.)
  */
  for (i = 1; i < argc; i++) {
    if (argv[i][0] == '–')
      switch(argv[i][1]) {
        case 'f':
        case 'F':      /* Copy string into file name */
          strcpy(loadfile_name, &argv[i][2]);
        break;

        case 'd':
        case 'D':      /* Set flag to copy directly to dual port */
          file_into_dp = TRUE;
        break;
```

```
      case 'a':
      case 'A':      /* Get the address of the board */
        sscanf(&argv[i][2], "%x", &address);
        if (!(address & 0xff00)) address <<= 8;  /* If address given as a    */
                                                 /* byte, shift it to a word */
      break;

      default:
        usage(argv);
        exit(2);
    }
}

/* Check to see if all required inputs have been issued on the command line */
if ((strlen(loadfile_name) == 0) || (address == 0)) {
  usage(argv);
  exit(2);
}

/* Setup pointer to point to base of dual port */
base = (char far *) MK_FP(address,0x0000);

/* Reset the KTX (stop it from running) */
base[KTX_RESET] = 1;

/* open the clrram.bin file, to download it to dual port to clear the
   ram on the KTX */
if ((ifile = open("CLRRAM.BIN", O_BINARY | O_RDONLY)) == -1) {
  printf("Error opening CLRRAM.BIN.\n");
  exit(1);
}

/* Read the bytes from the file directly into dual port ram */
length = read(ifile, base, 0x800);

/* Close the file */
close(ifile);

/* Start the KTX running to execute the CLRRAM.BIN to clear ram to zeros */
base[KTX_RELEASE] = 1;

/* Wait for clrram.bin to finish or user hits keyboard */
while(base[length - 2]) if (kbhit()) exit(1);

/* Stop the KTX */
base[KTX_RESET] = 1;

/* If user did not request load directly to dual port, then first load
   the loadpcl.bin program (bootstrap program) */
if (file_into_dp == FALSE) {
  if ((ifile = open("LOADPCL.BIN", O_BINARY | O_RDONLY)) == -1 ) {
    printf("Error opening LOADPCL.BIN\n");
    exit(1);
  }

  /* Read up to 0x800 bytes from file into start of dual port */
  read(ifile, base, 0x800);
  close(ifile);
}

if ((loadfile = open(loadfile_name, O_BINARY | O_RDONLY)) == -1) {
  printf("Could not open file %s.\nExitting...\n",loadfile_name);
  exit(1);
}
```

```
/* If user did not request direct download to dual port, go through
   the iterations of downloading through the already download loadpcl.bin */
if (file_into_dp == FALSE) {

  while(!eof(loadfile)) {  /* loop until end of file */

    /* read up to 0x780 bytes into dual port offset 0x80, which is where
       the loadpcl.bin program expects to find the binary */
    length = read(loadfile, &base[LOADPCL_DATA_WINDOW], 0x780);

    /* Put the length (number of bytes just read) into appropriate location
       in dual port */
    * (int *) (&base[LOADPCL_SIZE]) = length;

    /* On first pass through LOADPCL.BIN, need to release the KTX from
       reset, subsequent passes through LOADPCL.BIN the procedure is
       different */
    if (firstpass) {
      base[KTX_RELEASE] = 1; /* Release the KTX from reset */
      firstpass = 0;         /* set flag so next pass through is different */
    }
    else base[LOADPCL_HANDSHAKE] = 0; /* Not the first pass through, so set byte to */
                                /* zero to tell loadpcl.bin that another chunk*/
                                /* of data is ready to be transferred.        */

    /* Wait for loadpcl.bin to indicate it is done with this chunk, or user */
    /* presses a key.                                                       */
    while(base[LOADPCL_HANDSHAKE] == 0) if (kbhit()) exit(1);

    /* Check status byte for errors */
    switch(base[LOADPCL_STATUS]) {
      case 0:  /* OK – no error */
      break;

      case 1:
        printf("\nDownload error: Block too large.\n");
        exit(2);
      break;

      case 2:
        printf("\nDownload error: RAM full.\n");
        exit(2);
      break;

      default:
        printf("\nDownload error: Unknown – %02x.\n", base[LOADPCL_STATUS]);
        exit(2);
      break;
    }
  }

  printf("\nTotal bytes downloaded: %04x\n", *(int *) (&base[LOADPCL_LENGTH]));
  base[KTX_RESET] = 1;
  return(0);
}
else {  /* user requested a direct load into dual port */

  /* do the read directly from the file into dual port */
  length = read(loadfile, base, 0x800);

  /* Check to see if user requested a file which was too long to fit in
     dual port.                                                        */
  if (length == 0x800) {
    printf("\nFile is too long to fit in dual port.\n");
    exit(1);
  }
```

```
    printf("\nTotal bytes downloaded to dual port: %04x\n", length);
  }

  return(0);
}

void usage(char *argv[])
{
  printf("Usage:\n");
  if (strrchr(argv[0], '\\'))
    printf("%s -a<xx> -f<nnnnnnnn.eee> [-d]\n", strrchr(argv[0], '\\') + 1);
  else
    printf("%s -a<xx> -f<nnnnnnnn.eee> [-d]\n", argv[0]);

  printf("  Where:  xx           --> Upper byte of address of KTX board (e.g. D3)\n");
  printf("          nnnnnnnn.eee --> File name of binary to download\n");
  printf("          -d           --> Required to download file directly to dualport\n");
  printf("                           The diagnostic files require this switch.\n");
}
```

Anhang C

KT*x*-Hardwareregister

Inhalt des Anhangs

Die folgende Tabelle beschreibt die KT*x*-Hardwareregister.

Offset von Basisadresse	Mnemonik	E/ISA-Zugriff	Beschreibung
:800h	keys_read_reg	16-Bit-Nur-Lesen	Siehe Beschreibung des M16-Tests.
:801h	host_to_ktx_int_reg	Schreiben	Das Schreiben eines beliebigen Wertes erzeugt einen Interrupt an den KTx-Scanner.
:802h	assert_reg	Schreiben	Das Schreiben eines beliebigen Wertes aktiviert (schaltet) den KTx-Mikroprozessor (ein).
:803h	deassert_reg	Schreiben	Das Schreiben eines beliebigen Wertes deaktiviert (schaltet) den KTx-Mikroprozessor (aus).
:804h	status_reg	Lesen	Bit 0: Interrupt anstehend. Im gesetzten Zustand gibt dieses Bit an, daß der KTx-Scanner den Host unterbrochen hat. Bits 1–4: Nicht definiert Bit 5: Modus 816. Bei rückgesetztem Bit wird der KTx-Scanner im 16-Bit-Modus betrieben; bei gesetztem Bit wird der KTx-Scanner im 8-Bit-Modus betrieben. Bit 6: Halt. Bei rückgesetztem Bit wird der KTx-Mikroprozessor angehalten. Bit 7: Rücksetzen. Bei rückgesetztem Bit befindet sich der KTx-Mikroprozessor im Rücksetzzustand.
:805h	key_0_write_reg	Schreiben	Siehe Beschreibung des M16-Tests.
:807h	key_1_write_reg	Schreiben	Siehe Beschreibung des M16-Tests.
:808h	host_ack_of_ktx_int_reg	Schreiben	Das Schreiben eines beliebigen Wertes setzt das Bit “Interrupt anstehend” (Bit 0) des KTx-Statusregisters zurück.
:80Ah	card_control_write_reg	Schreiben	Bit 0: Interrupt-Modus Im rückgesetzten Zustand erzeugt der KTx-Scanner Impuls-Interrupts. Im gesetzten Zustand erzeugt der KTx-Scanner Interrupts auf EISA-Ebene. Bit 1: M16-Modus Im rückgesetzten Zustand wird der KTx-Scanner im erweiterten 16-Bit-Modus betrieben. Im gesetzten Zustand wird der KTx-Scanner im standard-mäßigen 16-Bit-Modus betrieben. Eine detaillierte Beschreibung finden Sie in Kapitel 2. Bits 2–7: Nicht definiert
:80Ch	card_control_read_reg	Lesen	Bits 0–4: Nicht definiert Bit 5: Blinken der LED. Im gesetzten Zustand blinkt die LED (sofern eingeschaltet). Bit 6: Farbe der LED. Im rückgesetzten Zustand leuchtet die LED grün (sofern eingeschaltet). Im gesetzten Zustand leuchtet die LED rot (sofern eingeschaltet). Bit 7: LED eingeschaltet. Im gesetzten Zustand ist die LED eingeschaltet.

Glossar

Inhalt des Anhangs

Dieses Glossar enthält Definitionen für Begriffe, die häufig in Scannerdokumentationen von Allen-Bradley verwendet werden.

Abfrageliste Eine Liste, welche die Reihenfolge, in der E/A-Adapter abgefragt werden sollen, angibt. Diese Liste wird vom Host angegeben und an das Scannermodul als Scanner-Verwaltungsbefehl gesandt. Obwohl maximal 16 E/A-Adapter zulässig sind, kann die Abfrageliste bis zu 64 E/A-Adapter enthalten. Der Scanner kann somit Adapter mehrmals während seines Abfragezyklus abfragen, falls häufigere Aktualisierungen gewünscht sind.

ASIC ASIC ist ein Akronym für “application-specific integrated circuit” (anwendungsspezifische integrierte Schaltung).

Ausgangsdatentafel Ein Speicherbereich, der Ausgangsschalter von Ausgangsmodulen steuert. Wird ein Bit gesetzt, so wird der entsprechende Ausgang eingeschaltet. 256 16-Bit-Worte (4096 Punkte) stehen zur Verfügung.

Blocktransfer Die Übertragung von Daten (jeweils bis zu 64 Worte) an ein bzw. aus einem intelligenten E/A-Modul.

Diskrete E/A E/A, die sich durch einen Anschluß je Datentafelbit auszeichnen (Anschluß = Punkt).

E/A-Aktualisierung Die serielle Abfrage des Scanners aller E/A-Chassis im E/A-Untersystem. Diese Abfrage verläuft asynchron zu Abfragen zwischen Scanner und Host-Prozessor.

E/A-Chassis Eines von vier Gehäusen, die 4, 8, 12 oder 16 diskrete E/A-Module aufnehmen können.

E/A-Gruppe Ein Adressierungskonzept, das 16 Eingangsbits (ein Eingangsdatentafelwort) und 16 Ausgangsbits (ein Ausgangsdatentafelwort) darstellt.

Im Bereich der Hardware kann eine E/A-Gruppe einen oder zwei Steckplätze in einem E/A-Chassis darstellen und besitzt eine Adressennummer.

E/A-Racknummer Ein Adressierungskonzept, das 8 E/A-Gruppen darstellt. Eine E/A-Racknummer kann auf ein, zwei, drei oder vier E/A-Chassis verteilt werden. Es können je nach Chassisgröße und Anwendungsanforderungen auch zwei Racknummern einem E/A-Chassis zugeordnet werden.

Eingangsdatentafel	Ein Speicherbereich, der Eingangsanschlüsse von Eingangsmodulen überwacht. Wird ein Eingangsschalter geschlossen, so wird das entsprechende Eingangsbit gesetzt. 256 16-Bit-Worte (4096 Punkte) stehen zur Verfügung.
Einslot-Adressierung	Adressierung, bei der eine E/A-Gruppe einen Steckplatz darstellt.
End of interrupt (EOI)	Befehl, der an eine speicherprogrammierbare Interrupt-Steuerung erteilt wird, um Interrupts zu quittieren.
Halbslot-Adressierung	Adressierung, bei der eine E/A-Gruppe einen halben Steckplatz darstellt.
Interrupt-Bearbeitungsroutine (ISR)	Eine Funktion für die Bearbeitung eines Interrupts.
Modul mit hoher Schreibdichte	Diskretes E/A-Modul mit 16 Eingangs- oder Ausgangspunkten.
Modul mit normaler Schreibdichte	Diskretes E/A-Modul mit 4, 6 oder normalerweise 8 Eingangs- oder Ausgangspunkten.
Modul mit vierfacher Schreibdichte	Diskretes E/A-Modul mit 32 Eingangs- oder Ausgangspunkten.
Moduladresse	Eine Adresse, welche die physikalische Position des Moduls im E/A-Chassis durch seine E/A-Racknummer und die Nummer des beginnenden Steckplatzes definiert.
Modul-Steuerbyte (MCB)	Das in die Ausgangsdatentafel eingegebene, für ein Blocktransfermodul bestimmte Byte zeigt durch den gesetzten Zustand eines der zwei höherwertigen Bits an, daß eine Blocktransfer-Lese- bzw. -Schreibanforderung vorliegt.
Scanner-Verwaltungsanforderung	Ein Befehl vom Host an den Scanner, der zur Steuerung und Konfiguration der Scanner-Platine verwendet wird.
Speicherprogrammierbare Interrupt-Steuerung (PIC)	Die Logik auf der PC-Mutterplatine, die Interrupts bearbeitet.
Steckplatz	Position in einem E/A-Chassis mit der Breite eines diskreten E/A-Moduls.
Störungsabhängige Gruppe	Eine Gruppe von E/A-Adaptern, die als Einheit für Störungserkennungszwecke behandelt wird. Liegt eine Störung in einem der Adapter vor, sind alle Adapter in der Gruppe fehlerhaft.
Zweislot-Adressierung	Adressierung, bei der eine E/A-Gruppe zwei Steckplätze darstellt.

Unterstützungsleistungen

Bei Allen-Bradley bedeutet Kundendienst erfahrene Repräsentanten in Kundendienstzentren in wichtigen Städten weltweit für Verkaufsberatung und -unterstützung. Zu unseren wertsteigernden Dienstleistungen gehören:

Technische Unterstützung

- SupportPlus-Programme
- Telefonische Unterstützung und eine rund um die Uhr besetzte Hotline für Notfälle (in den USA)
- Software- und Dokumentationsaktualisierungen
- Technische Abonnementdienste

Konstruktions- und Vorortdienstleistungen

- Unterstützung beim Anwendungsdesign
- Unterstützung bei der Integration/Inbetriebnahme
- Dienstleistungen vor Ort
- Wartungsunterstützung

Technische Schulung

- Vortrags- und Laborkurse
- Computer- und Video-Schulung im Eigentempo
- Arbeitsplatzhilfen und Workstations
- Analyse der Schulungsanforderungen

Reparatur- und Austauschleistungen

- Ihre einzige “autorisierte” Quelle
- Aktuelle Revisionen und Verbesserungen
- Weltweite Austauschmöglichkeit
- Örtliche Unterstützung

A

B

C

D

E

R

S

U

V

W

Z

Rockwell Automation

Allen-Bradley • Sprecher+Schuh

Allen-Bradley und Sprecher+Schuh helfen ihren Kunden seit mehr als 90 Jahren bei ihrer Produktivitätssteigerung und Qualitätsverbesserung. Wir entwickeln, produzieren und unterstützen weltweit ein umfassendes Sortiment von Steuerungs- und Automatisierungsprodukten. Dazu zählen speicherprogrammierbare Steuerungen, Niederspannungsgeräte, Antriebs- und Achssteuerungen, intelligente Bediengeräte (MMI), Sensoren und zahlreiche Softwareprodukte. Allen-Bradley und Sprecher+Schuh gehören zu Rockwell International, einem der führenden Technologieunternehmen der Welt.

Unsere Niederlassungen finden Sie an wichtigen Standorten weltweit.

Ägypten • Algerien • Argentinien • Australien • Bahrain • Belgien • Brasilien • Bulgarien • Chile • Costa Rica • Dänemark • Deutschland • Ecuador • El Salvador • Finnland • Frankreich • Griechenland • Guatemala • Honduras • Hongkong • Indien • Indonesien • Irland • Island • Israel • Italien • Jamaika • Japan • Jordanien • Jugoslawien • Kanada • Kolumbien • Korea • Kroatien • Kuwait • Libanon • Malaysia • Mexiko • Myanmar • Neuseeland • Niederlande • Norwegen • Oman • Österreich • Pakistan • Peru • Philippinen • Polen • Portugal • Puerto Rico • Qatar • Rumänien • Rußland - GUS • Saudi Arabien • Schweiz • Singapur • Slowakei • Slowenien • Spanien • Südafrikanische Republik • Taiwan • Thailand • Tschechische Republik • Türkei • Ungarn • Uruguay • USA • Venezuela • Vereinigte Arabische Emirate • Vereinigtes Königreich • Vietnam • Volksrepublik China • Zypern

Hauptverwaltung: Allen-Bradley, 1201 South Second Street, Milwaukee, WI 53204 USA. Tel: (1) 414 382 2000, Fax: (1) 414 382 4444
Hauptverwaltung Europa: Allen-Bradley • Sprecher+Schuh, Avenue Herrmann Debroux, 46, 1160 Brüssel, Belgien. Tel: (32) 2 663 06 00, Fax: (32) 2 663 06 40

Deutschland: Allen-Bradley • Sprecher+Schuh, Düsselberger Straße 15, 42781 Haan-Gruiten. Tel: (49) 2104 9600, Fax: (49) 2104 960121
Schweiz: Rockwell Automation AG, Gewerbepark, Hintermättlistrasse 3, 5506 Mägenwil, Tel: (41) 62 889 7777, Fax: (41) 62 889 7766
Österreich: Allen-Bradley • Sprecher+Schuh, Bäckermühlweg 1, 4030 Linz. Tel: (4370) (0732) 38 909 0, Fax: (04370) (0732) 38 909 61

Vertriebsbüros Deutschland–

Düsseldorf:	Tel: (49) 211 748350, Fax: (49) 211 7483511
Frankfurt:	Tel: (49) 6103 37970, Fax: (49) 6103 379710
Hannover:	Tel: (49) 511 674020, Fax: (49) 511 6740222
Stuttgart:	Tel: (49) 711 77790, Fax: (49) 711 7779101
Hamburg:	Tel: (49) 40 770171, Fax: (49) 40 7658843
München:	Tel: (49) 89 4274430, Fax: (49) 42744323
Berlin:	Tel: (49) 30 8913013, Fax: (49) 30 8913042
Mittweida:	Tel: (49) 37 2792221, Fax: (49) 37 2798985

Vertriebsbüros Österreich–

Graz:	Tel: (43) (0) 316 9153190, Fax: (43) (0) 316 9153195
Innsbruck:	Tel: (43) (0) 512 34 13 62, Fax: (43) (0) 512 39 13 62
Linz:	Tel: (4370) (0732) 38 909 0, Fax: (4370) (0732) 38 909 61
Wien:	Tel: (431) (0222) 6966060, Fax: (431) (0222) 1 69660660

Vertriebsbüros Schweiz –

Bulle:	Tel: (41) 292 0264, Fax: (41) 292 0267
Mägenwil:	Tel: (41) 62 889 7777, Fax: (41) 62 889 7766
Aarau:	Tel: (41) 62 837 2222, Fax: (41) 62 837 2907
Bern:	Tel: (41) 31 9929800, Fax: (41) 31 9929803
Lamone:	Tel: (41) 91 604 6262, Fax: (41) 91 604 6264
Renes:	Tel: (41) 21 6313232, Fax: (41) 21 6313231
Wil:	Tel: (41) 71 929 9225, Fax: (41) 71 929 9266

Publikation 1784-6.5.20DE – Dezember 1994

PN 956470-52