Escáner 1784-KT*x*
(Número de catálogo 1784-KTXES, -KTXDES y -KTSES)

Use estas notas informativas con la publicación 1784-6.5.20ES, el Manual de referencia del escáner 1784-KT*x*.

Los cuadros le llaman la atención sobre cada modificación. Seleccionamos este formato de manera que usted pueda cortar y pegar estos bloques de información en su manual, si así lo desea.

En la página P-1, añada esta sección acerca de la terminología al prefacio.

Terminología que utilizamos

Usamos estas abreviaciones, siglas e iniciales en el texto del escáner.

Término	Significado
ADS	estado de decisión del adaptador (bit)
API	interface de programación de aplicación
ASIC	circuito integrado para aplicación específica
BT	transferencia en bloques
COS	cambio de estado
EOI	fin de interrupción
FDG	grupo dependiente de fallo
ISR	rutina de servicio de interrupción
MCB	byte de control de módulo
MDL	longitud de decisión del adaptador
PIC	controlador de interrupción programable
SIF	fallo emitido por el escáner (bit)

En la página P-1, tome nota de la adición de la instrucción README.

Acerca de los ejemplos

Hemos diseñado los ejemplos en este manual para introducir el programador de aplicación al interface y comandos de computadora principal a KT*x*. Los ejemplos demuestran todos los comandos de computadora principal a KT*x* así como otras tareas comunes, tales como:

- lectura y escritura de tablas de imagen de E/S
- lectura de tabla de estados
- activación y desactivación
- temporizador de control (watchdog)

Los ejemplos de comandos de computadora principal a KT*x* están diseñados para emitir un comando, luego encuestar la KT*x* para la confirmación antes de emitir el siguiente comando. La KT*x* le da a algunos comandos, tales como las transferencias en bloques, 4 segundos para completar sus tareas antes de que la KT*x* declare un error de tiempo límite. Para la mayoría de las aplicaciones no sería práctico suspender el procesamiento por tanto tiempo. En lugar de ello, la mayoría de las aplicaciones deberían estar diseñadas para aprovechar la capacidad de la KT*x* de interrumpir la computadora principal cada vez que una confirmación de comando está disponible o cuando ocurren otros eventos importantes.

Se ha incluido un código de fuente para una rutina de servicio de interrupción para demonstrar cómo una aplicación podría administrar las interrupciones de KT*x*. La rutina de servicio de interrupción no ha sido integrada con el resto del ejemplo de código de fuente. Esta tarea ha sido dejada para que la ejecute el programador de aplicación.

README: Estos ejemplos se proporcionan “TAL COMO ESTAN” y para su conveniencia, sin ninguna garantía implícita expresa del producto y su idoneidad.

En la página 2-11, tome nota de la adición del nombre del archivo de protocolo SCANNER.BIN.

Para cargar el archivo de protocolo

Para cargar el protocolo, el controlador debe:

- transferir el archivo cargador al puerto doble
- cargar el archivo de protocolo, SCANNER.BIN

Para transferir el archivo cargador al puerto doble

Primero transfiera el archivo binario cargador (LOADPCL.BIN) al área especificada del RAM de puerto doble. El controlador usa el programa cargador para cargar el protocolo (firmware "suave") a través del interface de puerto doble y al RAM de Z80.

1. Borre todo el RAM en la tarjeta KT*x*.

A. Restablezca la tarjeta KT*x* escribiendo 01h al byte :803h.

B. Copie el programa CLRRAM.BIN al puerto doble, comenzando en :0000h.

C. Escriba **01h** al byte :0802h (para liberar el Z80).

D. Espere que la ubicación :002Ah cambie a cero. Si después de dos segundos esto no ha sucedido, significa que el CLRRAM.BIN no fue exitoso.

E. Restablezca la tarjeta KTx escribiendo **01h** al byte :0803h.

2. Transfiera el programa cargador (LOADPCL.BIN) al puerto doble, comenzando en :0000h.

Luego, cargue el protocolo, SCANNER.BIN.

En la página 4-11, tome nota de que la entrada 'confirmation_length' en la tabla de confirmación del comando Get Scan List ha sido modificada.

Comando Get Scan List

Use el comando Get Scan List para obtener una copia de la lista de escán actual del escáner. Este comando es bastante útil después de haber completado una configuración porque le permite ver cuáles adaptadores se encontraron en el vínculo.

Sintaxis del comando

host_command	7
transaction_number	0 – 255
command_length	0

Este comando genera una confirmación inmediatamente. La confirmación tiene la siguiente forma:

Confirmación del comando Get Scan List

host_command	7
transaction_number	0 – 255 (lo que haya suministrado la computadora principal)
confirmation_status	SUCCESS ①
confirmation_length	# de entradas en la lista de escán + 1
scan_list_count	# de entradas en la lista de escán
adapter_address [64]	conjunto de direcciones de adaptadores ha ser escaneadas 1 a 64 entradas

① SUCCESS es la única condición posible.

La rutina en el Ejemplo 4.E instruye al escáner a regresar la lista de escán actual.

En la página 5-2, cambie la sección Longitud decidida por el adaptador (ADL) con la siguiente sección, Longitud decidida por el módulo.

Longitud decidida por el módulo

Para la mayoría de las aplicaciones, la computadora principal dictamina al módulo BT la cantidad de datos a ser transfererida en un bloque de transferencia de lectura o escritura. Hay circunstancias, tales como cuando se configura por primera vez un módulo BT, cuando la computadora principal permite que el módulo BT especifique el número de palabras a ser transferido. Este modo se llama Longitud decidida por el módulo.

Para ejecutar una transferencia en bloques de Longitud decidida por el módulo, la computadora principal emite un Host BT Read o Host BT Write al escáner pero especifica un bt_data_len de cero. Si es un Host BT Write, la computadora principal **debe** suministrar 64 palabras de datos y una suma de verificación de datos a ser transferidos. Si hay menos de 64 palabras en bt_write_data en el búfer de comandos, el escáner envía una confirmación con el código de error:

HOST_SUPPLIED_TOO_LITTLE_DATA

En la página 5-2, tome nota de la adición de la suma de verificación de datos y su fórmula a los ítems que la computadora principal debe suministrar.

Comando Host BT Write

La aplicación de computadora principal emite un comando Host BT Write cuando desea iniciar una transferencia en bloques a un adaptador. Para que la respuesta sea igual a la solicitud al momento de completar la transferencia en bloques, la computadora principal debe suministrar:

- la dirección de *rack lógico*
- la dirección de la ranura del módulo de la transferencia en bloques
- los datos que se van a escribir al módulo
- un número de transacción único
- una suma de verificación de datos (complementos de 1 de la adición de 8 bits de los bytes de datos en la transferencia en bloques)

Se pueden colocar en cola un máximo de 64 comandos de transferencias en bloques a la vez, con la limitación de una BT por módulo de transferencia en bloques (un máximo de 16 BT por rack lógico). El escáner establece un temporizador de 4 segundos para cada solicitud de BT. Si el temporizador llega a su límite antes de que se termine el intercambio BT con el adaptador, la solicitud BT se cancela y la confirmación se devuelve a la computadora principal indicando que se llegó al tiempo límite de BT.

En la página 5-3, tome nota de que se ha cambiado la fórmula para command_length.

Sintaxis del comando

host_command	4
transaction_number	0 – 255
command_length	4 + (2 * número de palabras de datos)
module_slot_address	0 – 15
logical_rack_address	0 – 31
bt_data_len	# de palabras de datos①
bt_write_data	1 – 64 palabras de datos
bt_data_checksum	complemento de 1 de la adición de 8 bits de los bytes de datos en la transferencia en bloques

① Se puede especificar de 0 a 63; si se especifica 0, el módulo decide la longitud de los datos y es necesario suministrar 64 palabras de datos.

Este comando coloca en cola una confirmación cuando se termina el intercambio de transferencia en bloques, es decir, ha recibido un paquete de respuesta BT desde el adaptador, o cuando se llegó al tiempo límite del temporizador de 4 segundos. La confirmación tiene la siguiente forma:

Confirmación del comando Host BT Write

host_command	4
transaction_number	0 – 255 (lo que haya suministrado la computadora principal)
confirmation_status	(vea la Tabla 5.A)
confirmation_length	0
data	N/A

En la página 5-4, cambie la Tabla 5.A con la siguiente tabla revisada.

Tabla 5.A
Condiciones de error del comando Host BT Write

Nemónico de error	Código	Descripción
SUCCESS	0	El comando BT Write fue exitoso.
BAD_COMMAND_DATA_LENGTH	26	La longitud de datos del comando excede el máximo.
BT_BAD_ADDRESS	27	La dirección de rack lógico o la dirección de ranura está fuera de los límites.
BT_BAD_ADDRESS_NOT_IN_SCAN_LIST	28	La dirección BT no está en la lista de escán.
BT_BAD_DATA_LENGTH	29	Se especificaron más de 63 palabras.
TOO_MANY_REQUESTS_FOR_MODULE	30	Este módulo ya tiene una BT en progreso.
BT_QUEUE_FULL	31	Ya hay 64 solicitudes de BT pendientes.
BT_REQUEST_TIMEOUT	36	El intercambio de BT entre el escáner y el adaptador no se completó dentro de 4 segundos.
USER_MODULE_REQUEST_TYPE_MISMATCH	38	La computadora solicitó una escritura, el módulo respondió con una solicitud de lectura.
USER_MODULE_LENGTH_MISMATCH	39	Las longitudes que solicitaron el escáner y el adaptador no son iguales.
HOST_SUPPLIED_TOO_LITTLE_DATA	40	La computadora principal no suministró 64 palabras de datos para una BT de longitud decidida por el módulo.
BT_CHECKSUM_ERROR	41	La suma de verificación suministrada por la computadora principal no es válida.
MODULE_IN_FAULT_GROUP_REQUESTED_BT	50	Se solicitó una transferencia en bloques de escritura para un módulo que está en un grupo con fallo.

La rutina en el ejemplo 5.A instruye al escáner para que ejecute un comando Host BT Write.

En la página 5-5, cambie el Ejemplo 5.A con el siguiente ejemplo revisado.

Ejemplo 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, chk_sum;
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 = 4 + (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, chk_sum=0; i<size; i++){
    dp–>cmd_buffer.cmd.bt_write.bt_data[i] = bt_buf–>bt_data[i];
    chk_sum += (UBYTE)(bt_buf–>bt_data[i]);
    chk_sum += (UBYTE)(bt_buf–>bt_data[i]>>8);
}
*((UBYTE *)&dp–>cmd_buffer.cmd.bt_write.bt_data[i]) = ~chk_sum;

/**** 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;
}
```

En la página 5-6, tome nota de que el pie de página en la tabla de sintaxis de comandos ha sido modificado. También tome nota de la modificación de confirmation_length en la tabla de confirmación del comando Host BT Read.

Comando Host BT Read

La aplicación de computadora principal emite un comando Host BT Read cuando desea iniciar una transferencia en bloques desde un adaptador. Para que la respuesta sea igual a la solicitud al momento de la transferencia en bloques, la computadora principal debe suministrar:

- la dirección del *rack lógico*
- la dirección de la ranura del módulo de la transferencia en bloques
- el tamaño de los datos que se van a leer desde el módulo
- un número de transacción único

Se pueden colocar en cola un máximo de 64 comandos de transferencia en bloques cada vez, con la limitación de una BT por módulo de transferencia en bloques (un máximo de 16 BT por rack lógico). El escáner establece un temporizador de 4 segundos para cada solicitud de BT. Si el temporizador llega a su límite antes de que se termine el intercambio BT con el adaptador, la solicitud BT se cancela y la confirmación se devuelve a la computadora principal indicando que se llegó al tiempo límite de BT.

Sintaxis del comando

host_command	5
transaction_number	0 – 255
command_length	3
module_slot_address	0 – 15
logical_rack_address	0 – 31
bt_data_len	# palabras de datos①

① Se puede especificar de 0 a 63; si se especifica 0, el módulo decide la longitud de los datos.

Este comando coloca en cola una confirmación cuando se termina el intercambio de transferencia en bloques, es decir, ha recibido un paquete de respuesta BT desde el adaptador, o cuando se llegó al tiempo límite del temporizador de 4 segundos. La confirmación tiene la siguiente forma:

Confirmación del comando Host BT Read

host_command	5
transaction_number	0 – 255 (lo que haya suministrado la computadora principal)
confirmation_status	(vea Tabla 5.B)
confirmation_length	lectura exitosa: # de bytes de datos + 1 byte de suma de verificación lectura no exitosa: 0 bytes
bt_read_data	1 – 64 palabras de datos

Importante: La versión 6 del binario de escáner KTX (y todas las versiones siguientes) pasarán la suma de verificación del bloque de transferencia de lectura a la computadora principal en la confirmación del comando host BT read.

En la página 5-8, cambie el Ejemplo 5.B con el ejemplo siguiente revisado.

Ejemplo 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, chk_sum;

/**** 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, chk_sum=0; i < bt_buf–>count; i++){
         bt_buf–>bt_data[i] =dp–>confirmation_buffer.conf.bt_read.bt_data[i];
         chk_sum += (UBYTE)(bt_buf–>bt_data[i]);
         chk_sum += (UBYTE)(bt_buf–>bt_data[i]>>8);
         }
     chk_sum = ~chk_sum;
     if (chk_sum != (UBYTE)(dp–>confirmation_buffer.conf.bt_read.bt_data[i])){
         /* ADD CALL / ERROR HANDLER FOR CHECK SUM ERROR */
         }
     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;
}
```

En la página B-2, tome nota de la adición de la instrucción README.

Acerca de los ejemplos

Hemos diseñado los ejemplos en este manual para introducir al programador de aplicaciones al interface y comandos de computadora principal a KT*x*. Los ejemplos demuestran todos los comandos de computadora principal a KT*x* así como otras tareas comunes, tales como:

- lectura y escritura de tablas de imagen de E/S
- lectura de tabla de estados
- temporizador de control (watchdog)
- activación y desactivación

Los ejemplos de comandos de computadora principal a KT*x* están diseñados para emitir un comando, luego encuestar la KT*x* para la confirmación antes de emitir el siguiente comando. La KT*x* le da a algunos comandos, tales como las transferencias en bloques, 4 segundos para completar sus tareas antes de que la KT*x* declare un error de tiempo límite. Para la mayoría de las aplicaciones no sería práctico suspender el procesamiento por tanto tiempo. En lugar de ello, la mayoría de las aplicaciones deberían estar diseñadas para aprovechar la capacidad de la KT*x* de interrumpir la computadora principal cada vez que una confirmación de comando está disponible o cuando ocurren otros eventos importantes.

Se ha incluido un código de fuente para una rutina de servicio de interrupción para demostrar cómo una aplicación podría administrar las interrupciones de KT*x*. La rutina de servicio de interrupción no ha sido integrada con el resto del ejemplo de código de fuente. Esta tarea ha sido dejada para que la ejecute el programador de aplicación.

README: Estos ejemplos se proporcionan “TAL COMO ESTAN” y para su conveniencia, sin ninguna garantía implícita expresa del producto y su idoneidad.

En la página B-3,cambie el archivo de ejemplo ktx_dp.h con el siguiente archivo modificado.

archivo ktx_dp.h

```
/*************************************************************************
*  KTX_DP.H                                                              *
*                                                                        *
*  Description:                                                          *
*                                                                        *
*    This file contains definitions 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

/*=========================================================================
==
  IMPORTANT! Some compilers align structures and/or arrays to word boundaries
  unless instructed otherwise.  The following #pragma statement instructs the
  compiler to align to BYTE boundaries.
=========================================================================
=*/

#pragma pack(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];
            UBYTE   byte_allocation_w;  /* Because the checksum is
dynamically
                                    placed after the last data word, this
                                    is only a placeholder.*/
} 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];
            UBYTE   byte_allocation_r;  /* Because the checksum is
dynamically
                                        placed after the last data word, this
                                        is only a placeholder.*/
} 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 DEFINITIONS
**
*****************************************************************************
/

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

typedef UBYTE   TABLE_NAME;

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

/*============================================================================
  IMPORTANT! Some compilers align structures and/or arrays to word boundaries
  unless instructed otherwise.  The following #pragma statement instructs the
  compiler to return to the alignment properties it was using before parsing
  this header file.
============================================================================*/

#pragma pack()

#endif
```

En la página B-7, tome nota de que el mensaje de error 40 ahora es “Host_Supplied_Too_Little_Data.”

archivo 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 HOST_SUPPLIED_TOO_LITTLE_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
```

En la página B-14, cambie el archivo de ejemplo confirm.c con el archivo siguiente modificado.

archivo 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 <stdio.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 ((USHORT)(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.
**      It is the caller's 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)
{

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

time_t  t0 = time(NULL);

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

while (dp->int_status_from_host != 0x00){
    if ((time(NULL) – t0) > 0x02){
            printf("\n KTX did not clear int_status_from_host in 2
seconds.\n");
            exit(FAIL);
    }
}

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
*/

t0 = time(NULL);

while (dp->int_status_from_host != 0x00){
    if ((time(NULL) – t0) > 0x02){
            printf("\n KTX did not clear int_status_from_host in 2
seconds.\n");
            exit(FAIL);
    }
}
}
```

En la página B-23, cambie el archivo de ejemplo bt_write.c con el archivo siguiente modificado.

archivo 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, chk_sum;
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 = 4 + (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, chk_sum=0; i<size; i++){
    dp–>cmd_buffer.cmd.bt_write.bt_data[i] = bt_buf–>bt_data[i];
    chk_sum += (UBYTE)(bt_buf–>bt_data[i]);
    chk_sum += (UBYTE)(bt_buf–>bt_data[i]>>8);
}
*((UBYTE *)&dp–>cmd_buffer.cmd.bt_write.bt_data[i]) = ~chk_sum;

/**** 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;
}
```

En la página B-24, cambie el archivo de ejemplo bt_read.c con el archivo siguiente modificado.

archivo 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, chk_sum;

/**** 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, chk_sum=0; i < bt_buf–>count; i++){
          bt_buf–>bt_data[i] =dp–>confirmation_buffer.conf.bt_read.bt_data[i];
```

```
            chk_sum += (UBYTE)(bt_buf->bt_data[i]);
            chk_sum += (UBYTE)(bt_buf->bt_data[i]>>8);
            }
        chk_sum = ~chk_sum;
        if (chk_sum != (UBYTE)(dp->confirmation_buffer.conf.bt_read.bt_data[i])){
            /* ADD CALL / ERROR HANDLER FOR CHECK SUM ERROR */
            }
        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;
}
```

En la página A-2, tome nota de la modificación del esquema del puerto doble.

:00E	host_dead_counter	1			Si es cero, el contador de computadora principal muerta está inhabilitado. Un valor diferente de cero define un período de tiempo límite: 20ms * valor La computadora principal debe establecer el alive_flag = 0 antes que se llegue al tiempo límite o el escáner declarará que la computadora principal está muerta.
:00F	reserved2	3			
:012	alive_flag	1			Vea host_dead_counter.
:013	duplicate_node	1	R	W	Establezca a 1 si se detecta un escáner duplicado.
:014	off_ktx	1			
:015	stopped_flag	1			
:016	module_state	1	R	W	Si hay un error de rutina (prueba CRC o RAM), el código de error se escribirá aquí.
:017	cos_link_address	1	R/W	R/W	Si la computadora principal habilita la detección CHANGE-OF-STATE, el escáner escribirá aquí la dirección del vínculo del adaptador que tuvo un cambio de estado. La computadora principal debe borrar esto escribiendo un 0FFh.
:018	unused	64H			
:07C	num_adapter_addresses	1	R	R/W	Número de adaptadores en el vínculo.
:07D	num_faulted_adapters	1	R	R/W	Número de adaptadores en el vínculo que actualmente están con fallo.
:07E	operating_status	2	R	R/W	Este byte lo utiliza la KT*x* para indicarle a la computadora principal el estado del módulo estableciendo/restableciendo los bits. Los bits (cuando se establecen) se definen como: Bit 0: Modo de programa Bit 1: Modo de prueba Bit 2: Modo de marcha Bit 3: Modo de depuración Bit 4: BT no solicitada se recibió en vínculo Bit 5: BT pendiente Bit 6: Existe fallo Bit 7: Fallo cambiado Bit 8: Respuesta de BT de lectura no solicitada Bit 9: Respuesta de BT de escritura no solicitada Bit 10: La cola de confirmación está llena Bit 11: Tipo desconocido de interrupción de computadora principal Bit 12: El escáner declaró que la computadora principal está muerta Bit 13: La tabla de imagen de entrada cambió de estado Bits 14–15: No usado

Publicación 1784-6.5.20-RN2ES – Abril de 1996
Publicación de reemplazo 1784-6.5.20-RN1ES – Junio 1995
PN 956672-87