Application Note

DMA and HDLC Libraries for ZiLOG's Z80382

AN009201-0501

ZiLOG Worldwide Headquarters • 910 E. Hamilton Avenue • Campbell, CA 95008
Telephone: 408.558.8500 • Fax: 408.558.8300 • www.zilog.com

Acknowledgements

Project Lead Engineer:

Torsten Katschinsky

Application and Support Engineers:

Torsten Katschinsky

System and Code Development:

Torsten Katschinsky

Table of Contents

List of Figures

Abstract

This application note describes and offers the complete source code for usage of the DMA and HDLC channels of the Z80382 communication processor. Because of the nature of applications in which the Z80382 is used, DMA and HDLC channels are the favorite peripherals. The goal of this application note is to give customers access to this powerful on-chip peripheral by providing a complete software library, which makes the use of DMA and HDLC more transparent to the software designer. The user can use both the DMA and the HDLC library separately, but when using HDLC, the user must invoke DMA. Therefore, we designed this example application within the scope of the DMA and HDLC libraries to demonstrate to a customer the easy usage of these libraries.

General Overview

We designed the Z80382 for data communication application. The comprehensive set of peripherals and the enhanced 380 core make this chip to a powerful solution. Because of the increased communication speed and the major applications, the most important peripherals are the DMA and the HDLC channels - especially for ISDN solutions and high-speed serial communication products. When reading the user manual and the feedback of customers indicated that there is a demand for DMA and HDLC support. This application note addresses this demand by offering a complete program library.

Both the DMA and the HDLC library include all necessary functions to enable use of the corresponding channels. API functions encapsulate configuration details. The user needs only to call this function with parameters, defined in separate header files. Interrupt capability supports both DMA and HDLC. The user enables a certain interrupt by calling a library function with the interrupt mask and the pointer to the C-handler function as parameter. The user can disable this interrupt.

The libraries also support dynamic channel requests. Different tasks within a software module allocate a channel without detailed information, depending on the availability of a free channel or the number of a free channel. The library handles channel requests alone and maintains all channels. When the application code requires a channel, it sends a request and the library function returns a free channel number or informs the calling function that there is no channel available. The returned channel number should then be stored as channel handle for further use. When a channel is no longer needed this channel can be released by calling the corresponding library function. This channel is then marked as FREE again and any other module can request it.

When using DMA or HDLC of the Z80382, an initialization function, (InitDMA() or InitHDLC(), must be called before the user can access a certain channel. This initialization routine prepares the internal structures and performs the initialization physically.

Perform a channel configuration by calling the corresponding library function. The user can use all the defined constants, collected in the corresponding header file, to determine the correct configuration parameters. It is the responsibility of the user to configure the DMA and/or HDLC channels according to the application requirements.

There are functions implemented, which give the user access to the information stored in the internal channel structure used by the library. This capability is enabled only for READ purposes. When application code changes stored values inside this internal structure, this change can cause unpredictable problems.

Especially for DMA, there are dedicated library functions for buffer management. The DMA library supports five list entries for each channel, whereas the last list entry has been set to 'TRANSFER IN LIST' to create a ring buffer. This capability ensures that each channel has it's own buffer which wraps around to the first list entry.

The library maintains the linking and unlinking of the user buffer. Application code links a buffer by calling the DMA library function 'LinkDMABuffer'. 'UnlinkDMABuffer' unlinks a buffer from a certain link list entry.

For each library there are help files, which aid the user to find the correct function to access the channels. These help files are available as a Windows Help File, as a Win Word Document, as an HTML file and as an RTF file. By opening the desired file the user gains access to all enumerates, typedefs, structures, functions with input parameters and return values.

The DMA and/or HDLC library can be used as real library, meaning the corresponding *.lib must be included in the project. The user can also add the corresponding C-source file to the project file list.

Application Note Directory and File Structure

All files are included in the main directory of this application note. Figure 1 depicts the directory structure. This directory contains the following files:

- <Help>, which contains all help files.
- The <Lib_C> directory, which contains the source files and the created lib files.
- The <Startup> directory, containing the startup assembly code for the Z80382.
- The <TestAppl> directory, containing the source files of the test application.

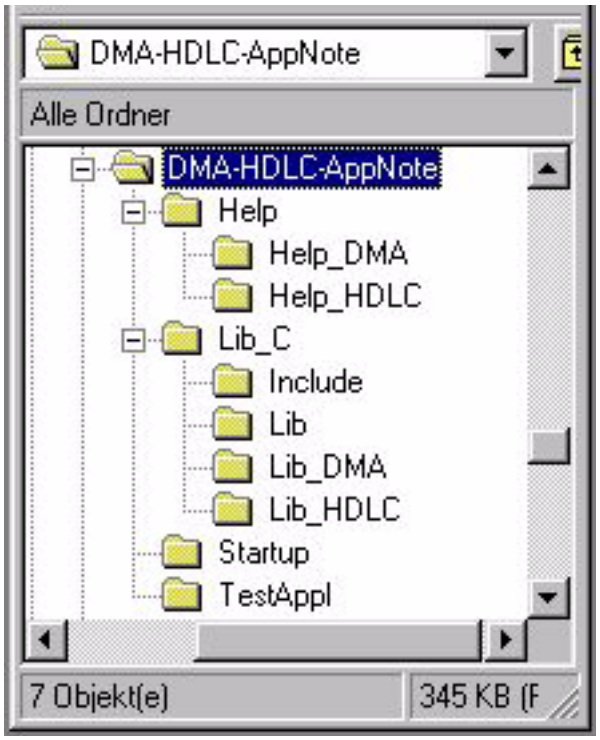

Figure 1. Directory Structure of the DMA/HDLC Application Note

The <Help> directory contains the help sub-directories for DMA and HDLC containing all help files. The user chooses between a Window Help File, a WinWord Document, an HTML Help File or a Text only Help File of RTF format. Figure 2 and Figure 3 depict the contents of the <Help_DMA> and <Help_HDLC> directory.

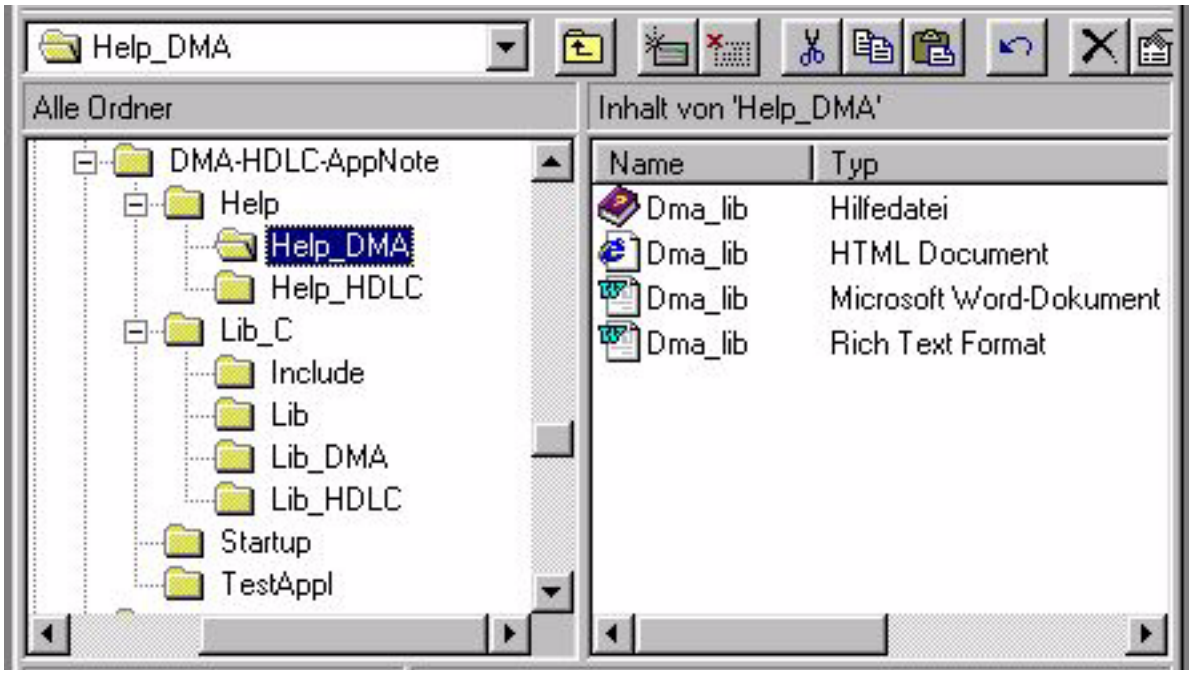

Figure 2. <Help_DMA> Directory Contents

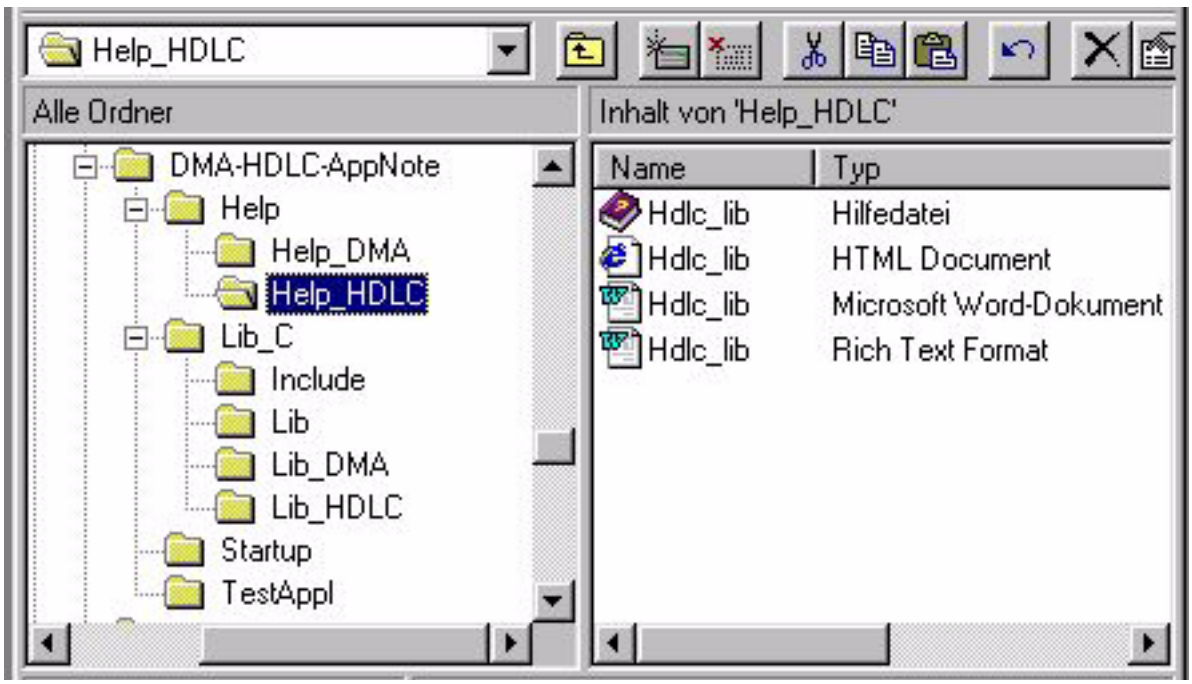

Figure 3. <Help_HDLC> Directory Contents

The library C-source code is located in <Lib_C>. The subdirectory <Include> contains all necessary header files. The created lib files (*.lib) are located in the <Lib> subdirectory. In <Lib_DMA> and <Lib_HDLC> you can find the corresponding C-source code and ZDS project files. See Figure 4 through Figure 7.

The project file allows changing or adding function separately to the library and build the lib file.

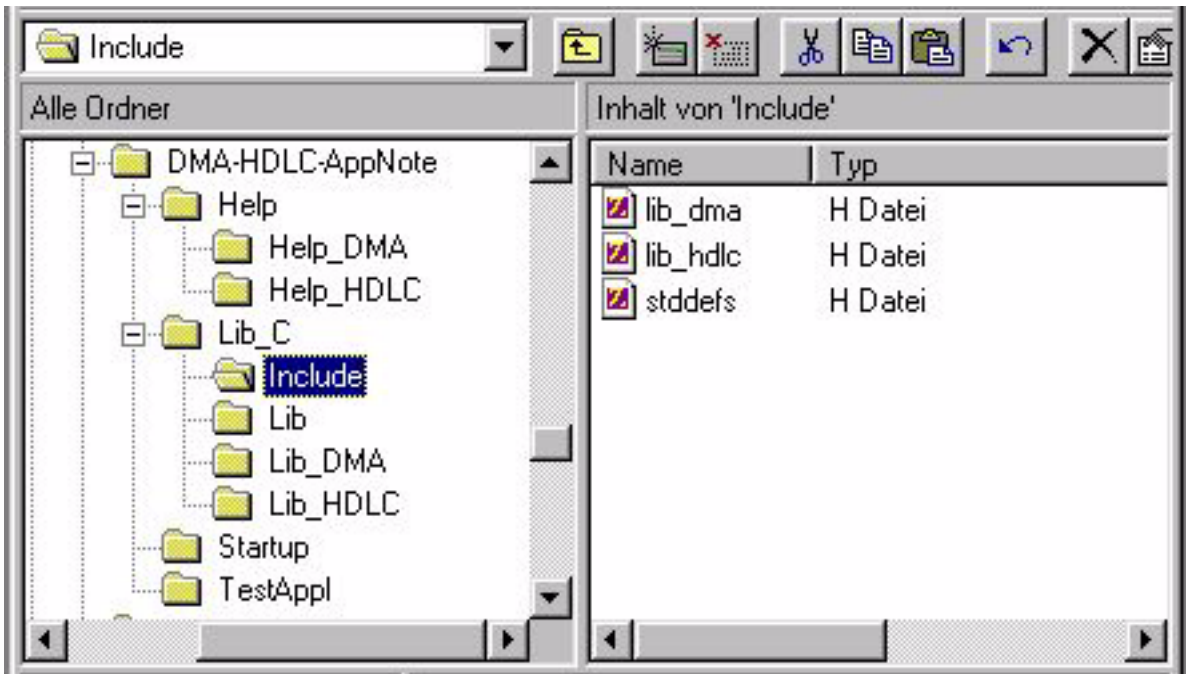

Figure 4. Contents of the <Include> Subdirectory

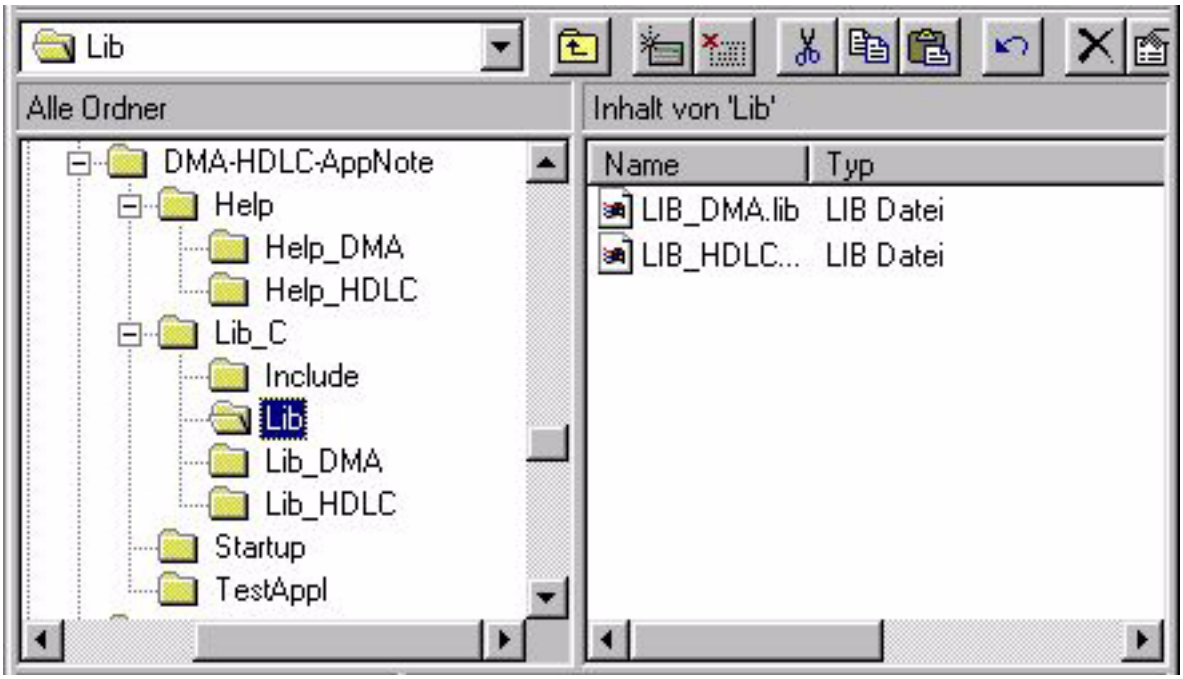

Figure 5. Contents of the <Lib> Subdirectory

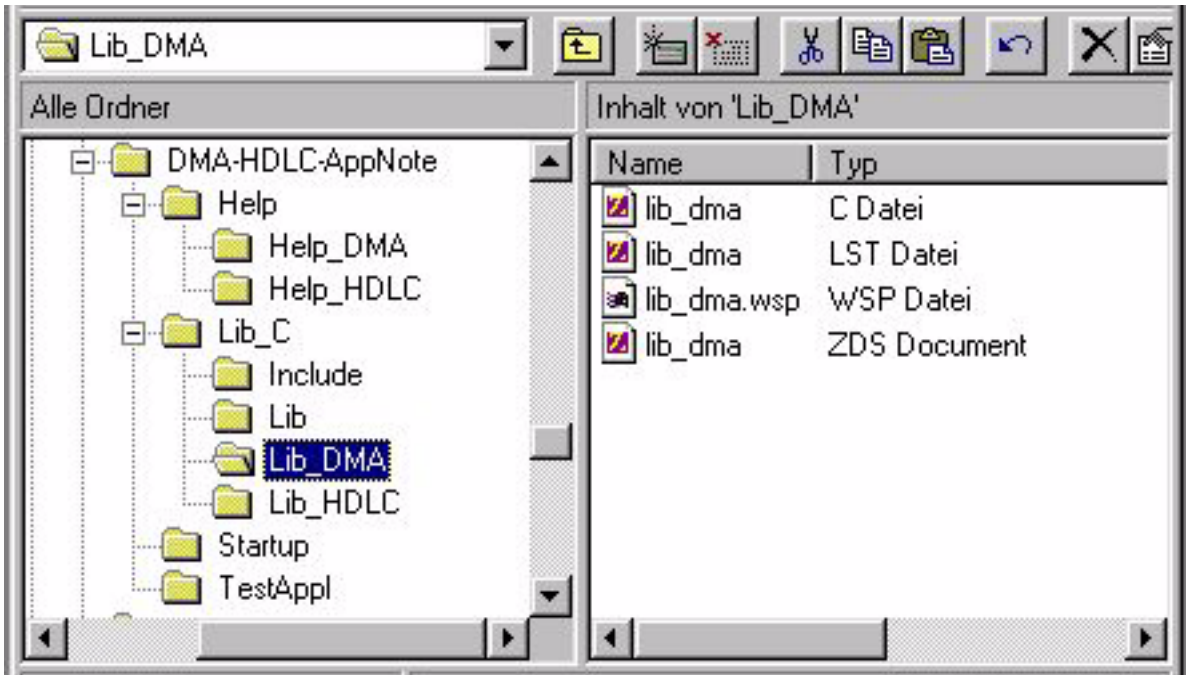

Figure 6. Contents of the <Lib _DMA> Subdirectory

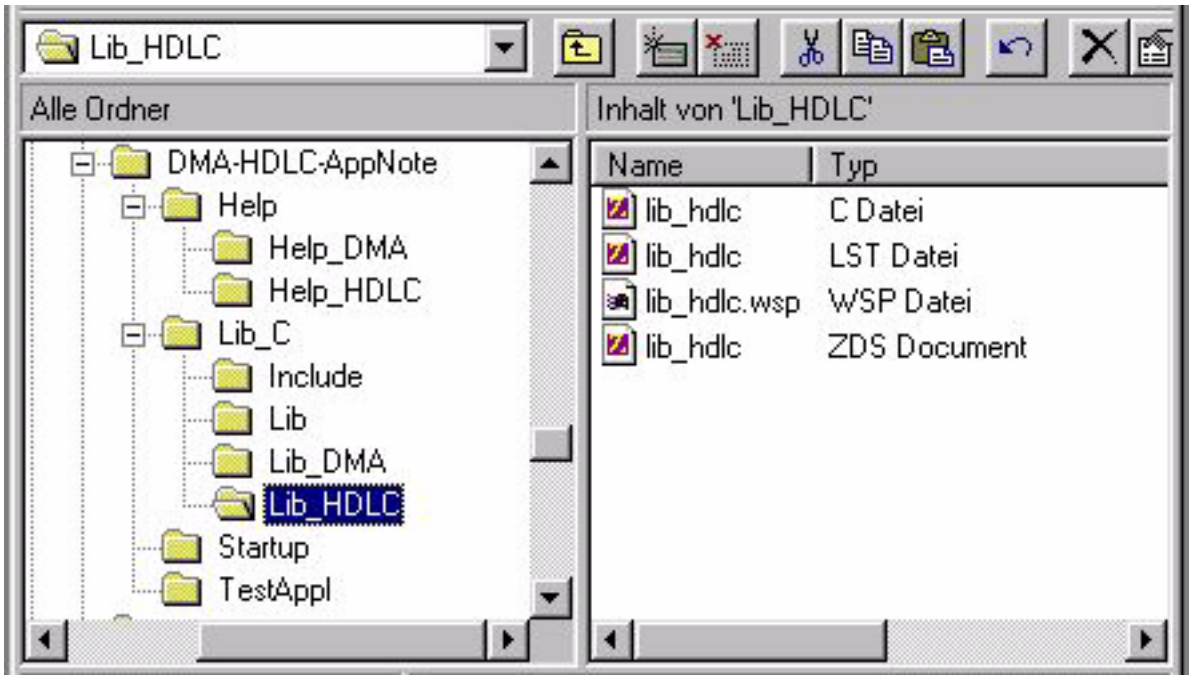

Figure 7. Contents of the <Lib_HDLC> Subdirectory

The <Startup> directory contains only the z380boot.s file, automatically included by ZDS when the user generates a new Z382 project (See Figure 8).

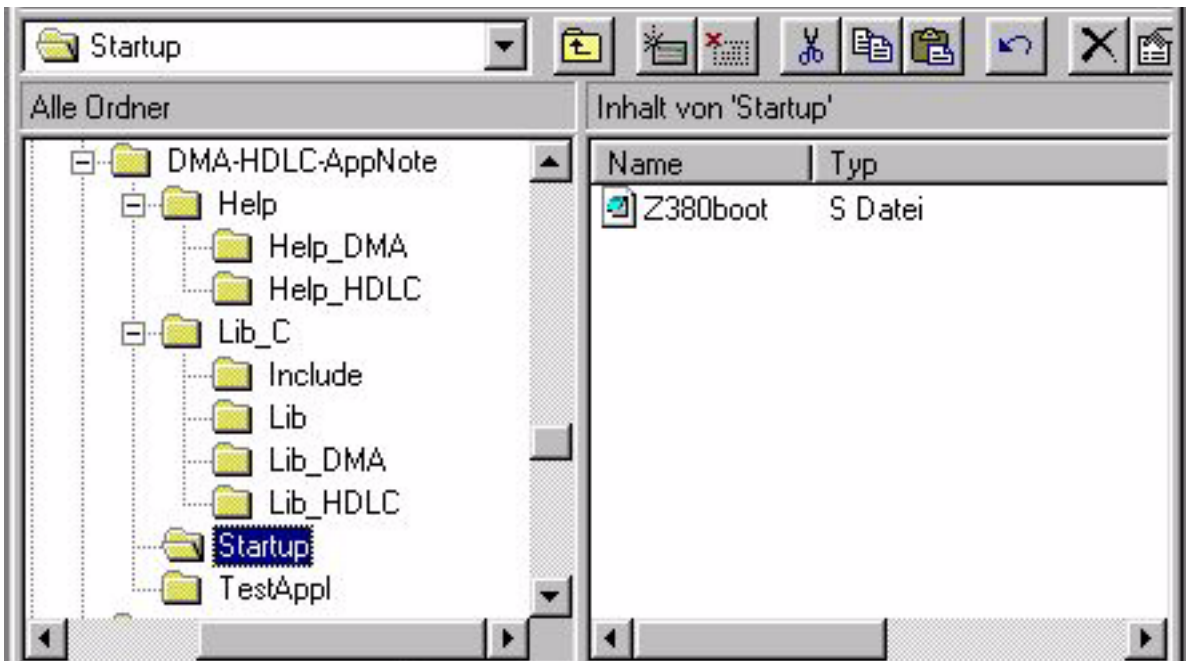

Figure 8. Contents of the <Startup> Subdirectory

In the <TestAppl> subdirectory (Figure 9) are located the source and project files for the test and example application. The "IntTab.s" file contains the interrupt table that supports DMA and HDLC interrupts. For test purposes, the customer can use it as an example or basic application. This test application uses nearly all library functions together with interrupt handler and real data transfer using DMA and HDLC.

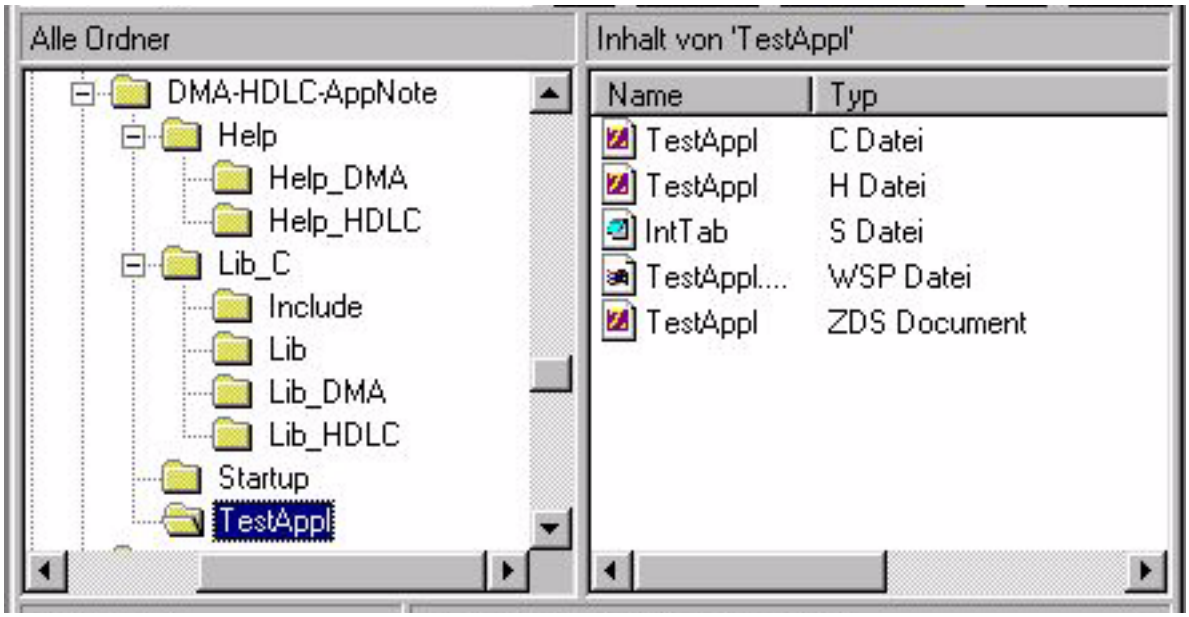

Figure 9. Contents of the <TestAppl> Subdirectory

Discussion

This section describes the basic operation of the DMA and HDLC library. The Technical Support sections, beginning on page 18, contains detailed information about functions and data structures. The user can also obtain this information directly from the help files, located in the <Help> subdirectory. This section explains all functions with input parameters and return values, which the user can pass through. The help files function as a complete reference for the DMA and HDLC libraries.

Theory of Operation

The user can use each of the libraries separately or together with the application code. Because of the nature of the HDLC channels of the Z80382, the user must use both libraries. An HDLC channel can only run with a dedicated DMA channel to transfer data from/to the TDM highway. A DMA channel can be used for any cli-

ent device of the Z382 on-chip peripheral like ASCI, HDLC, MIMIC or I/O Mailbox and can be selected when calling the configuration API function 'ConfigureDMAChannel'.

To ensure proper function, initialize the DMA and HDLC module. For that purpose, the user first calls the 'InitDMA' and 'InitHDLC' library functions. This function occurs immediately after entering the application main routine.

When using one or both libraries in an application, include the compiled lib files by setting the path and lib file name under the menu <Project->Settings->Linker->Object/Library modules>. Another method is to use the library source files and add these files to the project file list.

This application note contains an example test program. Use this example program as reference source code to demonstrate the use of both the DMA and HDLC libraries. In this example program, Loop Back is the standard configuration for the TX and RX HDLC channels. No additional external connections are necessary to test the data transfer using HDLC channel. To transmit and receive data over HDLC/DMA effectively, the buffer interrupt capability of the on-chip DMA controller is used. In this configuration, the application software manipulates data in a block-wise manner.

DMA Library

Depending on the customer, the project might or might not include the library (lib_dma.lib) or add the lib_dma.c source file. In either case, the application source file must include the header file "lib_dma.h".
The DMA library contains the following functions:

- Initializing and configuring DMA
- Linking an Application buffer to a DMA list entry
- Starting and stopping a DMA channel
- Writing and reading DMA register
- Writing DMA commands
- Enabling/disabling DMA channel interrupts

The library contains a global interrupt handler function whose address (address vector) must be placed in the interrupt vector table of the Z382 processor. This global DMA handler calls an application interrupt handler according to the channel that caused this interrupt. The application layer writes and saves this application handler. When enabling a certain DMA channel interrupt, the user must assign a dedicated interrupt handler by calling the library API function *'EnableDMAInterrupt'* (see "Technical Support–DMA Library API Reference" on page 18). The application DMA handler must follow this format:

void ApplicationDMAHandler(U8 *dmaid***, U8** *status*, **pTypeDMAListEntry** *ple***) {...}.**

The parameter *'dmaid'* corresponds to the DMA channel that caused this interrupt, *'status'* contains the current DMA channel interrupt status and *'ple'* is the pointer to the current link list entry. With this information, the application DMA handler is able to process this interrupt event.

The DMA library maintains four List Entries for each channel, which the application software must use. To access the Type/Status byte of a list entry correctly, align the list entries to an 8-Byte boundary, which the library processes. The user does not care about the necessary alignment. A channel list entry is a ring buffer, setting the last list entry to the type "TRANSFER IN LIST". This configuration allows the DMA controller to run through this ring buffer without CPU intervention. This type of list entry handling best fits the requirements for continuous data transfer. When enabling block interrupt for a certain DMA channel, the application software (application interrupt handler) processes incoming frames block-wise. To guarantee high data rates, the software unlinks the current completed buffer and immediately links a new buffer within the application interrupt handler. This procedure ensures that the DMA controller always detects a READY buffer and the DMA channel never stops. In the case that a DMA channel is in stop mode (when there is no READY buffer), the library automatically restarts the channel after invoking DMA_AUTO_RUN. The channel cannot start when there is no READY buffer.

From the data rate aspect the user can adjust the interrupt occurrence by changing the size of buffer linked to a certain DMA channel.

When linking an application buffer by calling '*LinkDMABuffer*', the application software stores the returned link list entry handle. With this handle, the software is always able to start a DMA transfer of a certain buffer. This handle is necessary when the application software links more than one buffer. Even when the DMA library processes the ordering of linked buffers within the DMA channel's link list ring buffer, the application code ensures the start of a DMA transfer with the first buffer linked, guaranteeing the correct sequence at the receiver end.

A complete list of all DMA API functions exists in "Technical Support–DMA Library API Reference" on page 18.

HDLC Library

Depending on the customer, the project might or might not include the library (lib_hdlc.lib) or add the lib_hdlc.c source file. In either case, the application source file must include the header file.
The HDLC library contains the following functions:

- Initializing and configuring HDLC

- Starting and stopping a HDLC channel
- Writing and reading HDLC register
- Writing HDLC commands
- Enabling/disabling HDLC channel interrupts

The library contains a global interrupt handler function whose address (address vector) must be placed in the interrupt vector table of the Z382 processor. This global HDLC handler calls an application interrupt handler according to the channel that caused this interrupt The application layer writes and saves this application handler. When enabling a certain HDLC channel interrupt, the user must assign a dedicated interrupt handler by calling the library API function *'EnableHDLCInterrupt'*(see "Technical Support–HDLC Library API Reference" on page 39). The HDLC channel type (HDLC_RECEIVER or HDLC_TRANSMITTER) determines the assignment of which interrupt handler to use. The HDLC handler must follow this format:

void ApplicationHDCLHandlerTx(U8 *hdlcid***, U8** *tstat***, U8** *istat***)** for the TX handler, and

void ApplicationHDLCHandlerRx(U8 *hdlcid***, U8** *istat***)** for the RX handler.

The parameter '*hdlcid'* corresponds to the HDLC channel that caused this interrupt, '*tstat'* contains the current TCSR register value (only for a TX channel interrupt) and '*istat'* contains the current interrupt status byte. With that information, the application HDLC handler processes this interrupt event.

Because of the Z382 processor specification that any HDLC channel must connect to a certain DMA channel, in most cases there is no need for HDLC channel interrupts. Because data transfer takes place on a block or frame basis (depends on the size of buffer linked to a DMA list entry) the best way is to handle all HDLC related actions within the DMA interrupt handler. It is important to continuously monitor the RUN bit of the HDLC channel, which processes data transmission. When using high data rates, an UNDERRUN condition occurs (only TX HDLC). In that case, the HDLC channel stops all activities until the flag clears under software control. Either monitor this flag in the corresponding DMA channel interrupt (in the example application) or enable the UNDERRUN interrupt.

A complete list of all HDLC API functions exists in "Technical Support–HDLC Library API Reference" on page 39.

Using Library Functions

A complete example application gives users access to the libraries. For detailed information of how to use DMA or HDLC channels, refer to the example application source files (*'TestAppl.c'* and *'TestAppl.h'*). This section addresses only basic functions.

Before using any library functions, call the corresponding initialization routine. Place *'InitDMA()'* and /or *'InitHDLC()*' before any statement using a library function. This routine initializes the internal data structures for further use. When using library functions without prior initialization of the corresponding module, the software does not work correctly.

The following code extract gives the basic steps to establish a transfer path over one HDLC channel. DMA must be involved to close the transfer path. The settings for the HDLC channel are an example. Adjust them to the actual application requirements.

```
*****************************************************************************/
U8              hRxDMA, hTxDMA, hHDLC, hFirstRxLE, hFirstTxLE, RxBuffer[32], TxBuffer[32];
DMA_CLIENT      DMAClient;

InitDMA();                          /*Initialize the DMA module */
InitHDLC();                         /*Initialize the HDLC module */

hRxDMA = RequestDMAChannel();       /* Request a DMA channel for RX*/
hTxDMA = RequestDMAChannel();       /* Request a DMA channel for TX*/

switch( hHDLC )                     /*When using HDLC, the DMAClient must be determined */

{
        case    0:
                DMAClient = DMA_CLIENT_HDLC0;
                break;
        case    1:
                DMAClient = DMA_CLIENT_HDLC1;
                break;
        case    2:
                DMAClient = DMA_CLIENT_HDLC2;
                break;
        default:
                DMAClient = DMA_CLIENT_HDLC0;
                break;
}

hHDLC = RequestHDLCChannel();       /*Request a HDLC channel */


if( (hRxDMA != NO_DMA_CHANNEL) && (hTxDMA != NO_DMA_CHANNEL)  && (hHDLC != NO_HDLC_CHANNEL) )
{
        /* Now, configure the DMA channel. A RxDMA must be configures as CLIENTtoMEM, a Tx DMA as MEMtoCLIENT. */
        /* We will use 16Bit CRC only and the application is responsible for checking RUN bit. */

        ConfigureDMAChannel(    hRxDMA,
                                DMAClient,
                                SCANEND_OneDMAOp,
                                BME_DISABLE | CLIENTtoMEM,
```

```
                                    DMA_CRC16,
                                    DMA_APPL_RUN);

        ConfigureDMAChannel(        hTxDMA,
                                    DMAClient,
                                    SCANEND_OneDMAOp,
                                    BME_DISABLE | MEMtoCLIENT,
                                    DMA_CRC16,
                                    DMA_APPL_RUN);

        /* Now, link the first application buffer. The application defines the size of the buffer. */
        /*We use READY buffer with no command, but with end of buffer notification. */

        hFirstRxLE = LinkDMABuffer( hRxDMA,
                                    &RxBuffer,
                                    DMA_RXBLKSIZE,
                                    DMA_RDY_BUF_NCMD_EOB);

        hFirstTxLE = LinkDMABuffer( hTxDMA,
                                    &TxBuffer,
                                    DMA_TXBLKSIZE,
                                    DMA_RDY_BUF_NCMD_EOB);

        /*Now, enable buffer interrupt with the dedicated application handler. DMA_VECTOR_BASE is the base address */
        /* of  the DMA vector table, defined by the application. */

        EnableDMAInterrupt( hRxDMA, DMA_VECTOR_BASE, BIE_BwSEoB, IrqHandlerRxDMA );
        EnableDMAInterrupt( hTxDMA, DMA_VECTOR_BASE, BIE_BwSEoB, IrqHandlerTxDMA );

        /*Now, configure the HDLC channel accordingly. After configuration, enable both the RX and TX path. */

        ConfigureHDLCChannel(
                hHDLC,
                PARA32( TXMODE_HDLC | TXCONF_TXD_BRG | TXDMAREQ_HALF | UWAIT_SET | UACTION_SET,
                        PREFRAME_SFLG | ISEL_DISJOINT | TXCRC32_DISABLE,
                        0x55,       NULL),
                RXMODE_HDLC | RXCONF_LPBACK | RXDMAREQ_HALF | RXCRC32_DISABLE | SIF_ENABLE,
                PARA32( 0, 8, 0, 8 ),
                ( hTxDMA << 4) | hRxDMA | RXDMA_ENABLE | TXDMA_ENABLE );

        EnableHDLCChannel( hHDLC, HDLC_TRANSMITTER_RECEIVER );
}

/*Now, start the DMA channels - at first RX, then TX! */

StartDMAChannel( hRxDMA, hFirstRxLE );
StartDMAChannel( hTxDMA, hFirstTxLE );
.
.
.
```

/***/

Results of Operation

We tested all library functions using the example application. Presuming that the Z382 Evaluation Board connects the serial cable to the PC/Laptop and the installation of the ZDS Monitor, the example application loads directly into ZDS and starts from RAM.

Verify the correct operation of all API functions, especially the link buffer procedure, by enabling real data transfer with and without DMA interrupts. When using HDLC, DMA must always be involved. Do not use HDLC interrupts for data transfer. Perform HDLC channels handling directly within the corresponding DMA interrupt handler, resulting in better over-all performance.

Summary

Reaffirmation of Results

The DMA/HDLC library, together with the included example application, give customers access to this powerful Z382 on-chip peripheral by including the function into customer's application code – either as a library module or as an additional source file. The benefit for the customer is that the software programmer does not need to study all peripheral register, the settings and procedure in detail, saving development time and reducing the number of failures while writing and testing code. The libraries contain all functions to setup, configure and work with DMA and HDLC, resulting in more readable software architecture. Using DMA/HDLC library, setup of a DMA/HDLC channel requires only a few hours instead of a few days or weeks.

Technical Support–DMA Library API Reference

_interrupt_IrqHandlerDMA

#pragma interrupt void _interrupt_IrqHandlerDMA(*void***)**

Global DMA Interrupt Handler. When an interrupt occurs it calls this handler. It places this handler address in the interrupt handler table.

The global handler clears the IP and IUS bit of the channel that caused this interrupt and calls the channel interrupt handler provided by the application. When the channel's RUN bit clears, the channel restarts according to the

DMA_RUN_MODE. This handler does not support nested interrupts and only processes one channel interrupt at any given time.

Return Value

Returns the following value:

TRUE

DMA channel interrupt successfully disabled

FALSE

Wrong DMA channel number

Parameters

void

VOID parameter.

ConfigureDMAChannel

BOOL ConfigureDMAChannel(U8 *dmaid*, **DMA_CLIENT** *client*, **U8** *scanmode*, **U8** *chanmode*, **DMA_CRC_MODE** *crcmode*, **DMA_RUN_MODE** *runmode*)

This function configures a DMA channel for use in relation to any on-chip modules. Select the DMA channel mode with this function. This function ignores bit fields for interrupt enable and sets them separately by calling the function EnableDMAInterrupt.

The value or mask used for the input parameter 'scanmode' and 'chanmode' corresponds to the values or mask described in the Z382 user manual. An application can request all structure elements by using a pointer to the DMA structure pTypeDMAConfigStatus or pTypeDMAChannelConfigStatus. Request this pointer using either GetpDMAConfigStatus or GetpDMAChannelConfigStatus.

Note: NOTE: Changing the scan mode affects ALL channels. By changing the scan mode, software reacts to a specific DMA load to favor the DMA operation over the host processor operation.

Return Value

Returns the following value:

TRUE

DMA channel successfully configured

FALSE

Wrong DMA channel number

Parameters

dmaid

DMA Channel number

client

DMA client, see DMA_CLIENT.

scanmode

Mode of operation saved in DMACR register

chanmode

Mode of operation saved in the DCSR register

crcmode

Used CRC mode when client is a HDLC channel

runmode

RUN mode selector, see DMA_RUN_MODE.

DisableDMAInterrupt

BOOL DisableDMAInterrupt(U8 *dmaid***, U8** *imask***)**

This function disables the selected DMA channel interrupts

Return Value

Returns the following value:

TRUE

DMA channel interrupt successfully disabled

FALSE

Wrong DMA channel number

Parameters

dmaid

DMA channel number

imask

Interrupt mask to disable

EnableDMAInterrupt

BOOL EnableDMAInterrupt(U8 *dmaid*, **U8** *vbase*, **U8** *imask*, **TypeDMAHandler** *handler*)

This function enables List and buffer interrupts for the selected channel. The calling function is responsible for selecting the correct vector base address.

Return Value

Returns the following value:

TRUE

DMA channel interrupts successfully enabled

FALSE

Wrong DMA channel number

Parameters

dmaid

DMA channel number

vbase

DMA ISR vector base address

imask

Interrupt mask for List Entry and Buffer

handler

Interrupt handler used for this channel, see TypeDMAHandler.

GetDMAListEntryId

U8 GetDMAListEntryId(U8 *dmaid*, **pTypeDMAListEntry** *ple*)

This function returns the LEID (List Entry Index) belongs to the physical Link Entry address.

Return Value

Returns the following value:

leid

List Entry index

LINK_FAILURE

Wrong DMA channel number

Parameters

dmaid

DMA channel number

ple

Pointer to the List Entry start address

GetDMAListEntryStartAddress

pTypeDMAListEntry GetDMAListEntryStartAddress(*void***)**

This function calculates a pointer within DMAListEntry table with address bit0 to bit2 zero. This function is necessary to access Type/Status byte by the DMA controller.

Return Value

A Pointer to the first List Entry

Parameters

void

VOID parameter.

GetNbFreeLE

U8 GetNbFreeLE(U8 *dmaid***)**

This function returns the number of unused list entries. The application function that called this function uses the return value to link a number of buffers according to the given number of free list entries.

Return Value

Returns the following value:

NbFreeLE

Number of unused list entries

NULL

Wrong DMA channel number or no free list entries available

Parameters

dmaid

DMA channel number.

GetpDMAChannelConfigStatus

pTypeDMAChannelConfigStatus GetpDMAChannelConfigStatus(U8 *dmaid***)**

This function returns the pointer to the TypeDMAConfigStatus structure. When using this pointer, an application has access to all structure elements. This function is useful when an application function requires information concerning the current DMA configuration.

Usually this internal structure is set indirectly by calling library API functions. There is no need to set some structure elements directly.The user can set structure elements directly by using a pointer to that structure.

Caution: Because this structure is used as internal configuration and status structure of the DMA library itself, the user should be aware that changing values without using corresponding API functions can cause unpredictable errors.

Return Value

Returns the following value:

pTypeDMAChannelConfigStatus

Pointer to structure TypeDMAChannelConfigStatus.

NULL

Wrong DMA channel number

Parameters

dmaid

DMA channel number

GetpDMAConfigStatus

pTypeDMAConfigStatus GetpDMAConfigStatus(*void***)**

This function returns the pointer to the TypeDMAConfigStatus structure. When using this pointer, an application has access to all structure elements. This access might be useful when an application function requires information about the current DMA configuration.

Because the DMA library uses this structure as its internal configuration and status structure, the application must avoid changing values without using corresponding API functions.

Return Value

Returns the following value:

pTypeDMAConfigStatus

Pointer to structure TypeDMAConfigStatus.

Parameters

void

Void parameter.

GetpDMAListEntry

pTypeDMAListEntry GetpDMAListEntry(U8 *dmaid*, **U8** *leid*)

This function returns the pointer to the list entry according to the given *leid*.

Return Value

Returns the following value:

pTypeDMAListEntry

Pointer to the List Entry

NULL

Wrong DMA channel number

Parameters

dmaid

DMA channel number.

leid

List Entry identifier.

InitDMA

void InitDMA(*void***)**

This function configures all DMA channels to default values. This function sets all status is to FREE and selects the first list entry. To ensure proper function, it sets the interrupt handler to NoDMAHandler. The last list entry links to the first by setting TRANSFER_IN_LIST in the last list entry. This function uses the first list entry in the table as the link address.

Return Value

No return value

Parameters

void

Void parameter

LinkDMABuffer

U8 LinkDMABuffer(U8 *dmaid***, U8 *** *badr***, U16** *blen***, U8** *typsta***)**

This function links a Link Entry table to an application buffer. The buffer address and the buffer length are set in the corresponding link entry field of the selected DMA channel. The Type field of the used Link Entry must be END_OF_LIST. This function allocates the 'NextListEntry' List Entry to link this buffer and wraps to List Entry 0 when reaching LAST_LE_INDEX. When this List Entry is IN PROGRESS or READY or COMPLETED this function returns FALSE to the application. In this case, the application ensures that this DMA channel is running and must re-try to link a buffer after a certain time. When a buffer links, the type goes to a READY status. Use the returned List Entry ID (leid) to release this Link Entry after the application reads/stores the data of the corresponding DMA buffer.

Return Value

Returns the following value:

cleid

DMA buffer successfully linked to Link Entry with index *'cleid'*

LINK_FAILURE

Wrong DMA channel number

LINK_BUSY

Link List in use

Parameters

dmaid

DMA channel number

badr

Pointer to the DMA buffer

blen

DMA buffer length

typsta

Type/Status byte value

NoDMAHandler

void NoDMAHandler(U8 *dmaid***, U8** *status***, pTypeDMAListEntry** *ple***)**

Set this interrupt handler when there are no other defined handlers.

Return Value

No return value

Parameters

dmaid

DMA channel number

status

DMA status value

ple

pointer to the current List Entry

ReadDMAphys

U32 ReadDMAphys(U8 *dmaid***, U8** *nbyte***)**

This function reads *N*-bytes in the register according to the channel number. When reading LAR or BAR, read three bytes. When reading BLR, read two bytes and when reading DCSR, read only one byte. The calling function ensures that the register selection bits contain the correct value to access the requested register.

Return Value

Read Result as 32-bit value

Parameters

dmaid

DMA channel number

nbyte

Number of bytes to read

ReadDMARegister

U32 ReadDMARegister(U8 *dmaid*, **DMA_REGISTER** *regnum*)

This function reads a value from the corresponding DMA register. When accessing the LAR, BAR or BLR the register set the selection bits in the DCSR according to multiplex address mode. When reading from DMACR or DMAVR, this function tests the DMA channel number but does not use it for register selection.

Return Value

Returns the value (always 32 bit) read from the register.

Parameters

dmaid

Identifies the DMA channel

regnum

Accessed Register, see DMA_REGISTER

ReleaseDMAChannel

BOOL ReleaseDMAChannel(U8 *dmaid*)

This function releases the selected DMA channel and sets the DMA_CHANNEL_STATUS back to DMA_FREE.

Clearing the RUN bit disables all interrupts for this channel and stops the DMA. No further action is possible when releasing a DMA channel.

Return Value

Returns the following value:

TRUE

DMA channel successfully released

FALSE

Wrong DMA channel or other failure

Parameters

dmaid

DMA channel number to release

RequestDMAChannel

U8 RequestDMAChannel(*void*)

This function determines if a free DMA channel is available. When this function detects a free channel it returns the corresponding channel number and marks it as BUSY.

Return Value

Returns the following value:

0 to 7

Free DMA channel

NO_DMA_CHANNEL

No free channel available

Parameters

void

Void parameter.

RunDMAChannel

BOOL RunDMAChannel(U8 *dmaid*)

This function sets the selected DMA channel into RUN mode by sending the SET RUN command.

Return Value

Returns the following value:

TRUE

DMA channel successfully started

FALSE

Wrong DMA channel number

Parameters

dmaid

DMA channel number

StartDMAChannel

BOOL StartDMAChannel(U8 *dmaid***, U8** *leid***)**

This function starts the selected DMA channel by writing the corresponding List Entry address into the LAR register. When writing the MSB of LAR the channel starts. The function WriteDMARegister notes the correct sequence when writing LAR (low, mid, high).

When this function is called, at least one Link Entry must be assigned to an application buffer beforehand and must have the type code READY.

Return Value

Returns the following value:

TRUE

DMA channel successfully started

FALSE

Wrong DMA channel number or NULL pointer

Parameters

dmaid

DMA channel number

leid

List Entry ID

StopDMAChannel

BOOL StopDMAChannel(U8 *dmaid***)**

This function stops the selected DMA channel by sending the CLEAR RUN command.

Return Value

Returns the following value:

TRUE

DMA channel successfully started

FALSE

Wrong DMA channel number

Parameters

dmaid

DMA channel number

UnlinkAllDMABuffer

BOOL UnlinkAllDMABuffer(U8 *dmaid*)

This function unlinks all DMA buffers independent of the Type Status byte.

Return Value

Returns the following value:

TRUE

All DMA buffer successfully unlinked

FALSE

Wrong DMA channel number

Parameters

dmaid

DMA channel number

UnlinkAllDMABufferCompleted

U8 UnlinkAllDMABufferCompleted(U8 *dmaid*)

This function unlinks all DMA buffers with TypeStatus COMPLETED. It sets the TypeStatus to End Of list, and returns the number of free List Entries. The user application can use the return value of this function to link N (FreeLEs=Number of free list entries) new application buffers.

Return Value

Returns the following value:

FreeLEs

Number of free list entries in the List Entry Table

Parameters

dmaid

DMA channel number

UnlinkDMABuffer

BOOL UnlinkDMABuffer(U8 *dmaid***, U8** *leid***)**

This function unlinks an application buffer from the specified List Entry. It sets the address field to NULL and resets the Type/Status byte to DMA_END_OF_LIST. After unlinking a DMA buffer, this List Entry can now link a new DMA buffer.

Return Value

Returns the following value:

TRUE

DMA Buffer successfully unlinked

FALSE

Wrong DMA channel number or List Entry

Parameters

dmaid

DMA channel number

leid

List Entry to release.

WriteDMACommand

BOOL WriteDMACommand(U8 *dmaid***, U8** *cmd***)**

This function writes a DMA command to a specific DMA channel

Return Value

Returns the following value:

TRUE

DMA channel command successfully transmitted

FALSE

Wrong DMA channel number

Parameters

dmaid

DMA channel number

cmd

DMA channel command

WriteDMAphys

void WriteDMAphys(U8 *dmaid*, **U8** *nbyte*, **U32** *value*)

This function writes N-bytes in the register according to the channel number. When writing LAR or BAR, write three bytes. When writing BLR, write two bytes and when writing DCSR write only one byte. The calling function ensures that the register selection bits contain the correct value to access the requested register.

Return Value

WriteDMAphys contains no return value

Parameters

dmaid

DMA channel number

nbyte

Number of bytes to read

value

Value to Write

WriteDMARegister

BOOL WriteDMARegister(U8 *dmaid*, **DMA_REGISTER** *regnum*, **U32** *value*)

This function writes a value to the corresponding DMA register. When accessing the LAR, BAR or BLR the register selection bits in the DCSR must be set accordingly due to multiplex address mode. When writing to DMACR or DMAVR this function tests the DMA channel *'dmaid'* but does not use it for register selection.

Return Value

WriteDMARegister returns the following value:

TRUE

Writing DMA register successfully terminated

FALSE

False DMA channel number

Parameters

dmaid

Identifies the DMA channel

regnum

Register to access, see DMA_REGISTER

value

Value to write into the selected register

DMA_CHANNEL_STATUS

```
enum DMA_CHANNEL_STATUS {
                DMA_FREE,
                DMA_BUSY,
};
```

DMA channel status definition for DMA channel access handling

Members

DMA_FREE

Channel FREE status

DMA_BUSY

Channel BUSY status

DMA_CLIENT

```
enum DMA_CLIENT {
                DMA_CLIENT_ASCI0,
                DMA_CLIENT_ASCI1,
                DMA_CLIENT_MIMIC,
                DMA_CLIENT_HOST_MAILBOX,
                DMA_CLIENT_HDLC0,
                DMA_CLIENT_HDLC1,
                DMA_CLIENT_HDLC2,
};
```

DMA client definition used for special treatments

Members

DMA_CLIENT_ASCI0

ASCI Port 0

DMA_CLIENT_ASCI1

ASCI Port 1

DMA_CLIENT_MIMIC

MIMIC Interface

DMA_CLIENT_HOST_MAILBOX

Host Mailbox Interface

DMA_CLIENT_HDLC0

HDLC channel 0

DMA_CLIENT_HDLC1

HDLC channel 1

DMA_CLIENT_HDLC2

HDLC channel 2

DMA_CRC_MODE

```
enum DMA_CRC_MODE {
              DMA_NO_CRC,
              DMA_CRC16,
              DMA_CRC32,
};
```

CRC mode definition used for a specific DMA channel. This information is useful when the client device is an HDLC channel. The handler, which assembles received frames, uses this information to discard the CRC bytes and deliver only the raw data to the application buffer.

Members

DMA_NO_CRC

No CRC support by HDLC

DMA_CRC16

CRC16 used with HDLC

DMA_CRC32

CRC32 used with HDLC

DMA_REGISTER

enum DMA_REGISTER {

DMA_CR,

DMA_VR,

DMA_LAR,

DMA_BAR,

DMA_BLR,

DMA_CSR,

DMA_NOREG,

};

Logical DMA register definition used for read & write operation

Members

DMA_CR

Global DMA Control register

DMA_VR

Global DMA Vector register

DMA_LAR

DMA channel Link Address register

DMA_BAR

DMA channel Buffer Address register

DMA_BLR

DMA channel Buffer Length register

DMA_CSR

DMA channel Control/Status register

DMA_NOREG

Logical value for NO DMA register

DMA_RUN_MODE

```
enum DMA_RUN_MODE {
                DMA_APPL_RUN,
                DMA_AUTO_RUN,
};
```

DMA channel RUN mode definition. DMA_APPL_RUN defines that the application is responsible for restarting the DMA channel, which has cleared its RUN bit. DMA_AUTO_RUN defines that the DMA library restarts the DMA channel when it detects a cleared RUN bit.

Members

DMA_APPL_RUN

Application RUN mode - application restarts DMA channel

DMA_AUTO_RUN

Auto RUN mode - DMA library automatically restarts the DMA channel

PDMAREG

PDMAREG defines a pointer to the DMA I/O register set.

pTypeDMAChannelConfigStatus

pTypeDMAChannelConfigStatus defines a pointer to the structure TypeDMAConfigStatus.

pTypeDMAConfigStatus

pTypeDMAConfigStatus defines a pointer to the structure TypeDMAConfigStatus.

pTypeDMAListEntry

pTypeDMAListEntry defines a pointer to the structure TypeDMAListEntry.

TypeDMAChannelConfigStatus

```
struct {
              DMA_CHANNEL_STATUS ChannelStatus;
              DMA_CLIENT Client;
              U8 ChannelMode;
              U8 RunMode;
              U8 NextListEntry;
              U8 CrcMode;
              U8 NbLinkedBuffer;
              U16 FrameLength;
              TypeDMAHandler IrqHandler;
} TypeDMAChannelConfigStatus;
```

This structure describes the configuration and status entries used for each separate DMA channel.

Members

ChannelStatus

Status of the DMA channel, see DMA_CHANNEL_STATUS

Client

DMA client identifier, see DMA_CLIENT.

ChannelMode

Channel mode selector without interrupt mask

RunMode

Run mode indicates how to use the RUN bit when cleared

NextListEntry

Index of the next list entry used for buffer linking

CrcMode

Used CRC mode when client device is a HDLC channel, see DMA_CRC_MODE

NbLinkedBuffer

Number of linked buffer in use

FrameLength

Current length of the rx/tx frame

IrqHandler

Interrupt handler for this DMA channel, see TypeDMAHandler.

TypeDMAConfigStatus

```
struct {
            U8 Vector;
            U8 ScanMode;
            TypeDMAChannelConfigStatus
ChannelConfigStatus[NB_DMA_CHANNEL];
} TypeDMAConfigStatus;
```

This structure describes the configuration and status entries used for each separate DMA channel.

Members

Vector

DMA vector when using with interrupts

ScanMode

Relevant for all channels

ChannelConfigStatus[NB_DMA_CHANNEL]

Configuration and status information of channel, see TypeDMAChannelConfig-Status.

TypeDMAHandler

Global DMA interrupt handler function. The DMA library calls this handler if a DMA interrupt occurs. The application creates this handler and enables a DMA interrupt with this handler as parameter.

Handler Parameter	
dmaid	DMA channel requested this interrupt
status	DCSR register value of this channel
ple	Pointer to the List Entry currently used by this channel

When an application creates a DMA channel interrupt handler, the syntax of this handler takes the following form:

void ApplicationIrqHandler(U8 *dmaid*, **U8** *status*, **pTypeDMAListEntry** *ple*)**{...};**

TypeDMAListEntry

```
struct {
              U8 LinkAdrL;
              U8 LinkAdrM;
              U8 LinkAdrH;
              U8 zero1;
              U8 TypeStatus;
              U8 zero2;
              U16 Length;
} TypeDMAListEntry;
```

This structure describes the general format of a DMA list entry used by the Z382 DMA controller. This type of list contains 8 bytes.

Members

LinkAdrL

Address bits 0-7 of the associated data buffer or next list entry

LinkAdrM

Address bits 0-7 of the associated data buffer or next list entry

LinkAdrH

Address bits 0-7 of the associated data buffer or next list entry

zero1

Reserved for future use - must be set to zero

TypeStatus

Defines the type of this entry

zero2

Reserved for future use - must be zero

Length

Provides the length of the associated data buffer

Technical Support–HDLC Library API Reference

_interrupt_IrqHandlerHDLC

#pragma interrupt void _interrupt_IrqHandlerHDLC(*void*)

Global HDLC interrupt handler. When any interrupt occurs it calls this handler. This handler appears in the interrupt handler table.

The global handler clears the IP and IUS bit of the channel caused this interrupt and calls the channel interrupt handler provided by the application.

This handler does not support nested interrupts and processes only one channel interrupt.

Return Value

Returns the following value:

TRUE

DMA channel interrupt successfully disabled

FALSE

Wrong DMA channel number

Parameters

void

VOID parameter.

ConfigureHDLCChannel

BOOL ConfigureHDLCChannel(U8 *hdlcid*, **U32** *txmode*, **U8** *rxmode*, **U32** *cap*, **U8** *dmasel***)**

This function configures a HDLC channel for using with a DMA channel. The mode uses 32-bit values and features the following structure (bytes 0(LSB)) to 3(MSB):

txmode	
byte 1	fill character
byte 2	control byte
byte 3	transmit mode value
rxmode	
byte 0	receive mode value

The 32-bit CAP value contains the TDM start time and length for both the RX and TX path.

According to the transmit mode (txmode: byte 3) the TX CAP values are identified either as TX TDM start and length value or as BRG time constant LSB and MSB! Both values in this function are treated in the same manner. The calling function is responsible for assigning the correct values.

cap	
byte 0	RX TDM Length
byte 1	RX TDM Start
byte 2	TX TDM Length / BRG Time Constant High
byte 3	TX TDM Start / BRG Time Constant Low

The DMA parameter contains the DMA channel for TX (in the upper four bits) and for the RX (in the lower four bits) of the 8-bit DMA selector word. After configuration, this function disables both the transmitter and receiver. Calling EnableHDLCChannel enables the transmitter and receiver.

Return Value

Returns the following value:

TRUE

HDLC channel successfully configured

FALSE

Wrong HDLC channel number

Parameters

hdlcid

HDLC Channel number.

txmode

Transmit mode of the HDLC channel

rxmode

Receive mode of the HDLC channel

cap

Counter Access Port values for TX/RX

dmasel

DMA Channels to be used for this channel

DisableHDLCChannel

BOOL DisableHDLCChannel(U8 *hdlcid***, HDLC_CHANNEL_TYPE** *type***)**

This function disables the HDLC Transmitter and/or Receiver by loading the register TMR and/or RMR with a Mode value equal to RXMODE_DISABLE or TXMODE_DISABLE.

Return Value

Returns the following value:

TRUE

HDLC channel successfully disabled

FALSE

Wrong HDLC channel or other failure

Parameters

hdlcid

HDLC Channel number

type

HDLC Channel type (see HDLC_CHANNEL_TYPE).

DisableHDLCInterrupt

BOOL DisableHDLCInterrupt(U8 *hdlcid***, HDLC_CHANNEL_TYPE** *type***, U8** *imask***)**

This function disables the corresponding HDLC channel interrupts of the transmitter or receiver.

Return Value

Returns the following value:

TRUE

HDLC channel interrupt successfully disabled

FALSE

Wrong HDLC channel number

Parameters

hdlcid

HDLC channel number

type

Selection of TX or RX path, see HDLC_CHANNEL_TYPE.

imask

Interrupt mask to disable

EnableHDLCChannel

BOOL EnableHDLCChannel(U8 *hdlcid***, HDLC_CHANNEL_TYPE** *type***)**

This function enables the HDLC Transmitter and/or Receiver by loading the register TMR and/or RMR with a Mode value greater than RXMODE_DISABLE or TXMODE_DISABLE.

Return Value

Returns the following value:

TRUE

HDLC channel successfully enabled

FALSE

Wrong HDLC channel or other failure

Parameters

hdlcid

HDLC Channel number

type

HDLC Channel type (see HDLC_CHANNEL_TYPE).

EnableHDLCInterrupt

BOOL EnableHDLCInterrupt(U8 *hdlcid*, **HDLC_CHANNEL_TYPE** *type*, **U8** *vbase*, **U8** *imask*, **void*** *handler*)

This function enables HDLC interrupts for the selected channel and either for the transmitter or receiver part. The calling function is responsible for selecting the correct vector base address.

Return Value

Returns the following value:

TRUE

HDLC channel interrupts successfully enabled

FALSE

Wrong HDLC channel number

Parameters

hdlcid

HDLC channel number

type

Selects the TX or RX path

vbase

HDLC ISR vector base address

imask

Interrupt mask corresponding to HDLC_CHANNEL_TYPE.

handler

Interrupt handler corresponding to HDLC_CHANNEL_TYPE.

GetpHDLCChannelConfigStatus

pTypeHDLCChannelConfigStatus GetpHDLCChannelConfigStatus(U8 *hdlcid*)

This function returns the pointer to the pTypeHDLCChannelConfigStatus structure. When using this pointer, an application has access to all structure elements. This function is useful when an application function requires information about the current HDLC configuration. To guarantee that the HDLC library operates correctly, the application might execute only the READ operation.

Return Value

Returns the following value:

pTypeHDLCChannelConfigStatus

Pointer to structure pTypeHDLCChannelConfigStatus.

NULL

Wrong DMA channel number

Parameters

hdlcid

HDLC channel number

GetpHDLCConfigStatus

pTypeHDLCConfigStatus GetpHDLCConfigStatus(*void*)

This function returns the pointer to the pTypeHDLCChannelConfigStatus structure. When using this pointer, an application has access to all structure elements. This function is useful when an application function requires information about the current HDLC configuration. To guarantee that the HDLC library operates correctly, the application might execute only the READ operation.

Return Value

Returns the following value:

pTypeHDLCConfigStatus

Pointer to structure pTypeHDLCChannelConfigStatus.

Parameters

void

Void parameter

InitHDLC

void InitHDLC(*void*)

This function configures all HDLC channels to default values. This function sets all channel status entries to HDLC_FREE (see HDLC_CHANNEL_STATUS), enabling the application to request a HDLC channel. To ensure proper function it sets interrupt handler to **NoHDLCHandler**.

Return Value

No return value

Parameters

void

Void parameter

NoHDLCRxHandler

void NoHDLCRxHandler(U8 *hdlcid*, U8 *istat*)

Use and set this interrupt handler when there are no other defined handlers

Return Value

No return value

Parameters

hdlcid

HDLC channel number

istat

HDLC interrupt status

NoHDLCTxHandler

void NoHDLCTxHandler(U8 *hdlcid*, U8 *tstat*, U8 *istat*)

Use and set this interrupt handler when there are no other defined handlers.

Return Value

No return value

Parameters

hdlcid

HDLC channel number

tstat

HDLC TX status byte

istat

HDLC interrupt status

ReadHDLCRegister

U8 ReadHDLCRegister(U8 *hdlcid***, HDLC_REGISTER** *regnum***)**

This function reads a value from the corresponding HDLC register. The return value is always 8-bit.

Return Value

Returns the value read from the register

Parameters

hdlcid

Identifies the HDLC channel

regnum

Register to be accessed, see HDLC_REGISTER

ReleaseHDLCChannel

BOOL ReleaseHDLCChannel(U8 *hdlcid***)**

This function releases the selected HDLC channel and sets the channel status back to HDLC_FREE. Disabling the RX and TX DMA in the selection register disables the receiver and transmitter and unlinks the DMA channel. No further action is possible when releasing the HDLC channel.

Return Value

Returns the following value:

TRUE

HDLC channel successfully released

FALSE

Wrong HDLC channel or other failure

Parameters

hdlcid

HDLC channel number to release

RequestHDLCChannel

U8 RequestHDLCChannel(*void***)**

This function determines if a free HDLC channel is available. When this function detects a free channel it returns the corresponding channel number and marks it as HDLC_BUSY (see HDLC_CHANNEL_STATUS).

Return Value

Returns the following value:

0 to 7

free HDLC channel

NO_HDLC_CHANNEL

No free HDLC channel available

Parameters

void

Void parameter

WriteHDLCCommand

BOOL WriteHDLCCommand(U8 *hdlcid***, HDLC_CHANNEL_TYPE** *type***, U8** *txcmd***, U8** *rxcmd***)**

This function writes a HDLC command to a specific channel.

Return Value

Returns the following value:

TRUE

HDLC channel command successfully transmitted.

FALSE

Wrong HDLC channel number

Parameters

hdlcid

HDLC channel number

type

Selection of TX or RX path, see HDLC_CHANNEL_TYPE

txcmd

HDLC TX channel command

rxcmd

HDLC RX channel command

WriteHDLCRegister

BOOL WriteHDLCRegister(U8 *hdlcid*, **HDLC_REGISTER** *regnum*, **U8** *value*)

This function writes a value to the corresponding HDLC register. All registers contain 8-bit values.

Return Value

Returns the following value:

TRUE

Writing HDLC register successfully terminated

FALSE

False HDLC channel number

Parameters

hdlcid

Identifies the HDLC channel

regnum

Register to be accessed, see HDLC_REGISTER.

value

Value to write into the selected register

HDLC_CHANNEL_STATUS

```
enum HDLC_CHANNEL_STATUS {
              HDLC_FREE,
              HDLC_BUSY,
};
```

HDLC_CHANNEL_STATUS defines the channel status to dynamically request and release HDLC channels.

Members

HDLC_FREE

Channel FREE status

HDLC_BUSY

Channel BUSY status

HDLC_CHANNEL_TYPE

```
enum HDLC_CHANNEL_TYPE {
              HDLC_TRANSMITTER,
              HDLC_RECEIVER,
              HDLC_TRANSMITTER_RECEIVER,
};
```

HDLC_CHANNEL_TYPE defines the HDLC channel type to distinguish between the HDLC Transmitter and Receiver.

Members

HDLC_TRANSMITTER

HDLC Transmitter

HDLC_RECEIVER

HDLC Receiver

HDLC_TRANSMITTER_RECEIVER

Both, HDLC Transmitter and Receiver

HDLC_REGISTER

```
enum HDLC_REGISTER {
              HDLC_TMR,
              HDLC_TIR,
              HDLC_TCSR,
              HDLC_TFR,
              HDLC_RMR,
              HDLC_RIR,
              HDLC_CAP,
              HDLC_DSR,
              HDLC_VR,
              HDLC_NOREG,
};
```

Logical HDLC register definition used for read and write operations

Members

HDLC_TMR

HDLC Transmit Mode Register

HDLC_TIR

HDLC Transmit Interrupt Register

HDLC_TCSR

HDLC Transmit Control/Status Register

HDLC_TFR

HDLC Transmit Fill Register

HDLC_RMR

HDLC Receive Mode Register

HDLC_RIR

HDLC Receive Interrupt Register

HDLC_CAP

HDLC Counter Access Port Register

HDLC_DSR

HDLC DMA Select Register

HDLC_VR

HDLC Vector Register

HDLC_NOREG

Number of HDLC register for test purposes

PHDLCREG

Defines a pointer to the HDLC I/O register set

pTypeHDLCChannelConfigStatus

Defines a pointer to the structure TypeHDLCChannelConfigStatus

pTypeHDLCConfigStatus

Defines a pointer to the structure TypeHDLCConfigStatus

TypeHDLCChannelConfigStatus

```
struct {
              HDLC_CHANNEL_STATUS ChannelStatus;
              U8 TxMode;
              U8 TxCtrl;
              U8 TxIntMsk;
              U8 TxFillChar;
              U8 RxMode;
              U8 RxIntMsk;
              U8 BRGTClow;
              U8 BRGTChigh;
              U8 TxTDMStart;
              U8 TxTDMLength;
              U8 RxTDMStart;
              U8 RxTDMLength;
              U8 DMASelect;
              TypeHDLCHandlerTx IrqHandlerTx;
              TypeHDLCHandlerRx IrqHandlerRx;
} TypeHDLCChannelConfigStatus;
```

This structure describes the configuration entries used for each separate HDLC channel.

Members

ChannelStatus

Status of the HDLC channel (see HDLC_CHANNEL_STATUS).

TxMode

Transmit mode

TxCtrl

Transmit control value

TxIntMsk

Transmit interrupt mask

TxFillChar

Transmit fill character

RxMode

Receive mode

RxIntMsk

Receive interrupt mask

BRGTClow

Counter Access Port - BRG Time Constance LSB

BRGTChigh

Counter Access Port - BRG Time Constance MSB

TxTDMStart

Counter access port - TX TDM start time

TxTDMLength

Counter access port - TX TDM length

RxTDMStart

Counter access port - RX TDM start time

RxTDMLength

Counter access port - RX TDM length

DMASelect

DMA channels used by this channel

IrqHandlerTx

TX Interrupt handler for this HDLC channel (see TypeHDLCHandlerTx

IrqHandlerRx

RX Interrupt handler for this HDLC channel (see TypeHDLCHandlerRx

TypeHDLCConfigStatus

```
struct {
            U8 Vector;
            TypeHDLCChannelConfigStatus
ChannelConfigStatus[NB_HDLC_CHANNEL];
} TypeHDLCConfigStatus;
```

This structure describes the configuration entries used for all HDLC channels. This structure has access to the channel structure elements of each channel. For further information see TypeHDLCConfigStatus structure definition.

Members

Vector

HDLC vector when using with interrupts

ChannelConfigStatus[NB_HDLC_CHANNEL]

Configuration information of this channel

TypeHDLCHandlerRx

Global HDLC receive interrupt handler function. The HDLC library calls this handler if an interrupt caused by an HDLC receiver occurs. The application creates this handler, containing code, to process this interrupt request. It enables an HDLC interrupt by calling EnableHDLCInterrupt with the handler address as the parameter.

Handler Parameter:	
hdlcid:	HDLC channel requested this interrupt
istat:	Receive Interrupt Status byte of this channel

When an application creates a HDLC RX interrupt handler, the syntax of this handler must take the following form:

void ApplicationRxIrqHandler(U8 *hdlcid*, **U8** *istat*)**{...};**

TypeHDLCHandlerTx

Global HDLC transmit interrupt handler function. The HDLC library calls this handler if an interrupt caused by an HDLC transmitter occurs. The application creates this handler, containing code, to process this interrupt request. It enables an

HDLC interrupt by calling EnableHDLCInterrupt with the handler address as the parameter.

Handler Parameter:	
hdlcid:	HDLC channel requested this interrupt
tstat:	TCSR register value of this channel
istat:	Transmit Interrupt Status byte of this channel

When an application creates a HDLC TX interrupt handler, the syntax of this handler must take the following form:

void ApplicationTxIrqHandler(U8 *hdlcid*, **U8** *tstat*, **U8** *istat*)**{...};**

Source Code(s)

The following files contain the complete source code, viewable after extracting the entire application note on the local computer.

- STDDEF.H
- LIB_DMA.H
- LIB_DMA.C
- LIB_HDLC.H
- LIB_HDLC.C
- TESTAPPL.H
- TESTAPPL.C
- INTTAB.S

Timing Diagrams/Tables

Not applicable

Technical Drawings

Not applicable

Test Procedure

Equipment Used

- PC/Laptop with Windows 9X, NT

- Z80380 C-Compiler V1.08
- ZDS3.5 or higher (I propose ZDS 3.64Beta3)
- Z80382 Evaluation Board with new monitor for ZDS
- Power Supply for the Evaluation Board
- Serial Interface cable to connect Z80382 Evaluation Board to the PC/Laptop
- DMA Library (Source code or Lib)
- HDLC Library (Source Code or Lib)
- Test Application (Source code, ZDS project file)

General Test Setup and Execution (Include Emulation Configurations)

The Z382 Evaluation board must have the ZDS Monitor installed. Program the EPROM with the '382_ZDS.HEX' file, downloadable from ZiLOG's home page (www.zilog.com) or taken from the ZDS directory. While downloading the .hex file, read the 'Z382README.TXT' file for further information.

Before using the application note, install at least ZDS version 3.5. Use the latest ZDS version 3.64 Beta3, downloadable from ZiLOG's home page.

1. Install the C-Compiler version 1.08 on your computer.
2. Using the serial cable, connect the Z382 evaluation board to the PC on which ZDS is running. Use the default jumper settings on the evaluation board and connect the serial cable to J16 (COM2). Use the Z382 evaluation board user manual as reference.
3. Load the project file 'TESTAPPL.ZWS' by using the <Load Project File> menu under ZDS file menu. Double clicking 'TESTAPPL.ZWS' within Windows Explorer also loads this file. ZDS sends a notice that the current project has moved to another location.
4. Confirm the following question to update the stored C-Compiler path within the project file. ZDS automatically replaces the C-Compiler path settings within the project.
5. Place the correct C-Compiler include path in the 'Additional Include Directories' list field of the compiler setting menu.
6. Change the Include path of the DMA/HDLC library according to the directory in which you installed the DMA/HDLC library. When finished, the 'Preprocessor' page of the compiler settings appears similar to those in Figure 10 (except that your path is different).

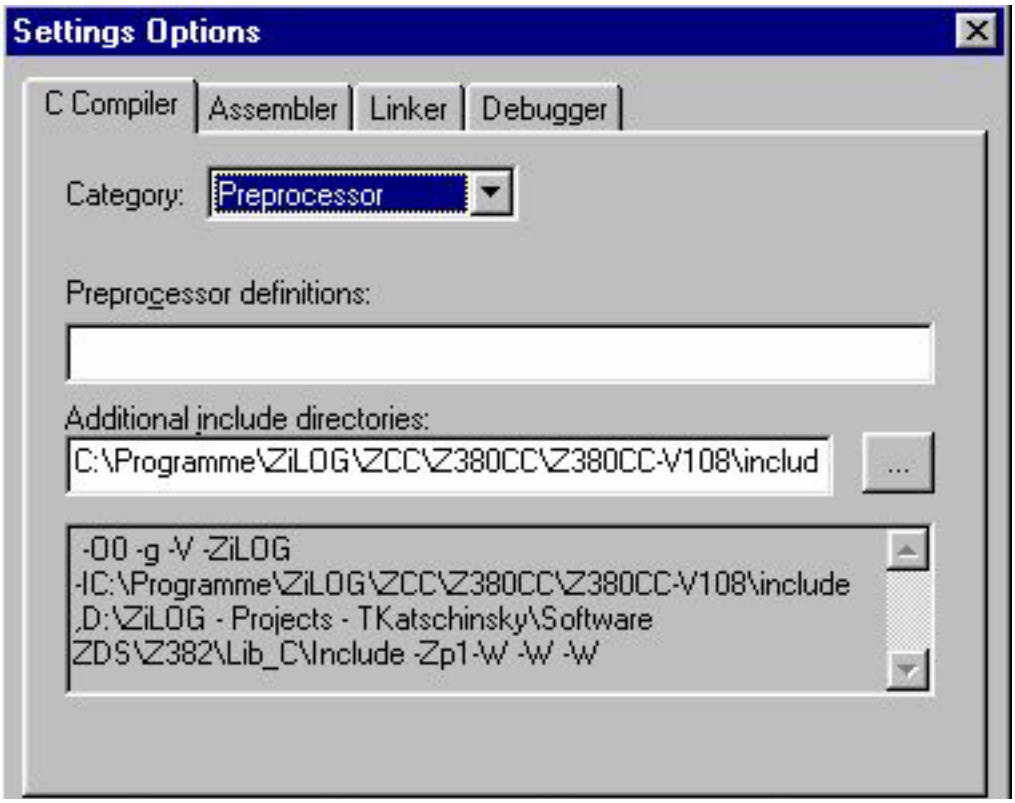

Figure 10. C-Compiler Settings–Preprocessor

7. Change the library path for the DMA and HDLC library under <Project – Settings – Linker – General - Object/Library> according to the directory in which you installed the libraries. See Figure 11.

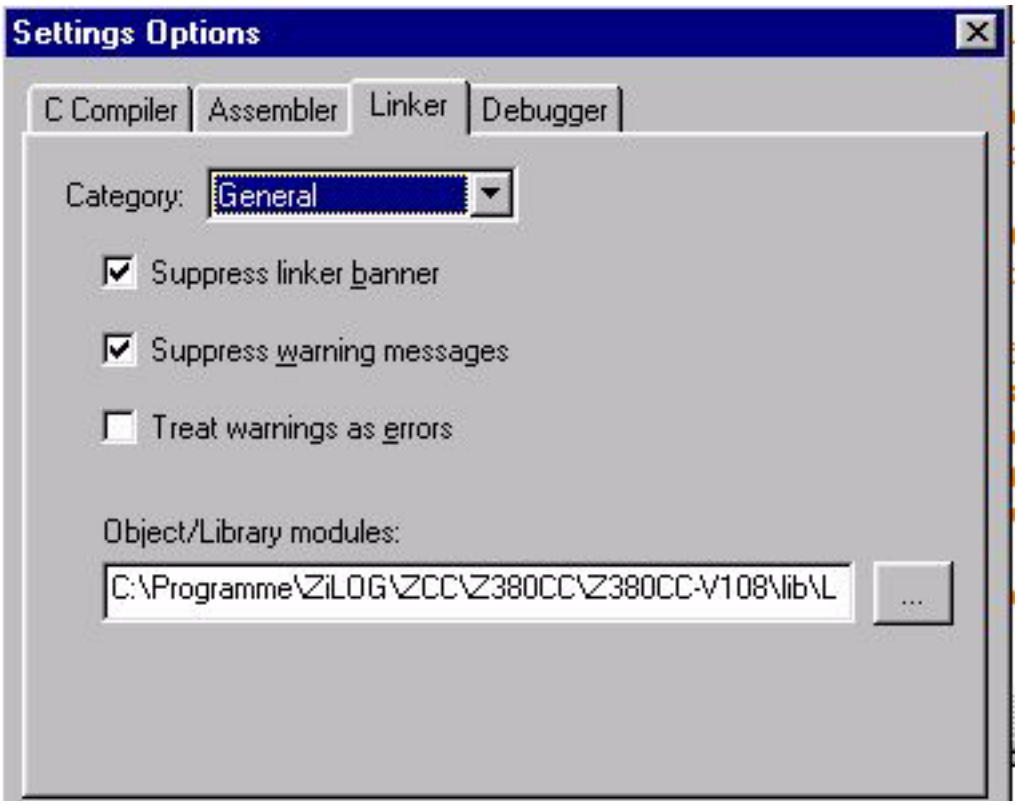

Figure 11. Linker Settings–General

8. Close the project setting menu and store the project.
9. Load the application into RAM by clicking the RESET/GO button at ZDS tool bar. ZDS indicates that the download is in progress.

The easiest way to verify the code is to send a number of bytes and check the receive buffer. When all data have been placed correctly in the receive buffer without any errors or missing bytes, the test was successful. Check the content of a by installing a corresponding C-Watch symbol within ZDS. Get the information for this procedure from the ZDS help functions.

If the software does not function, check the following settings (see Figure 12 through Figure 15).

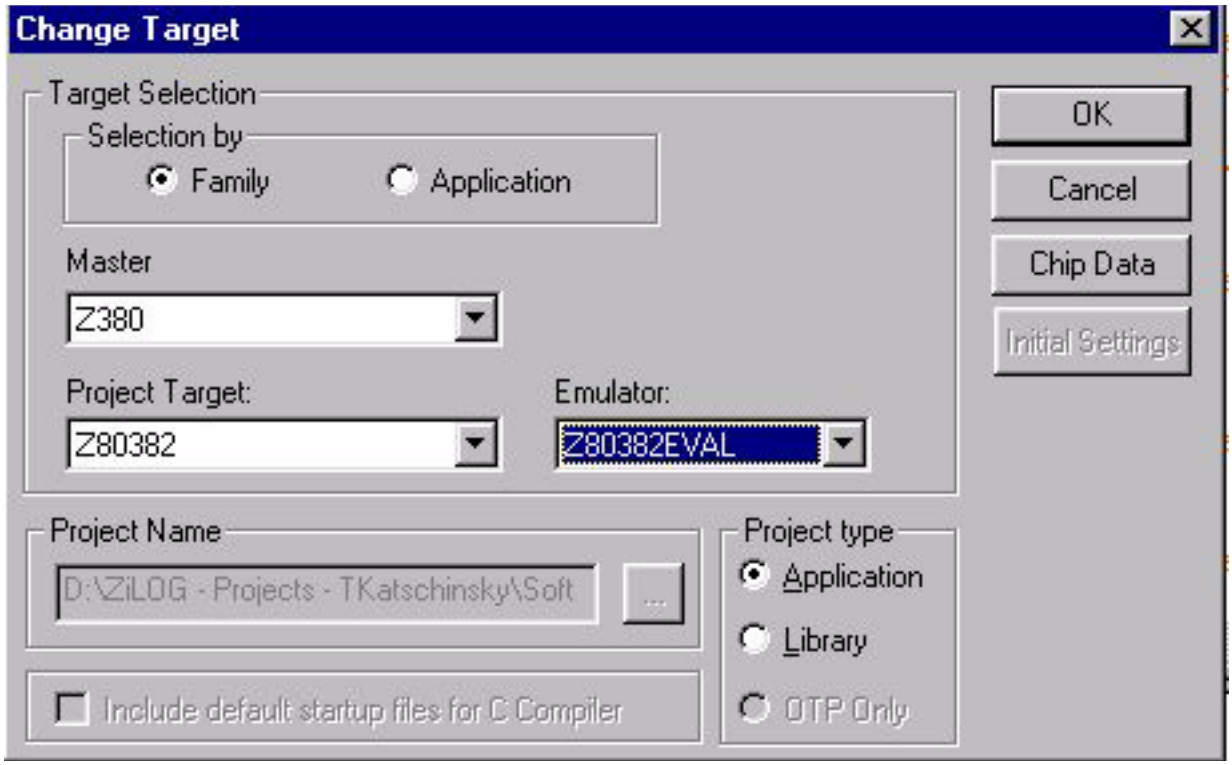

Figure 12. Target Settings

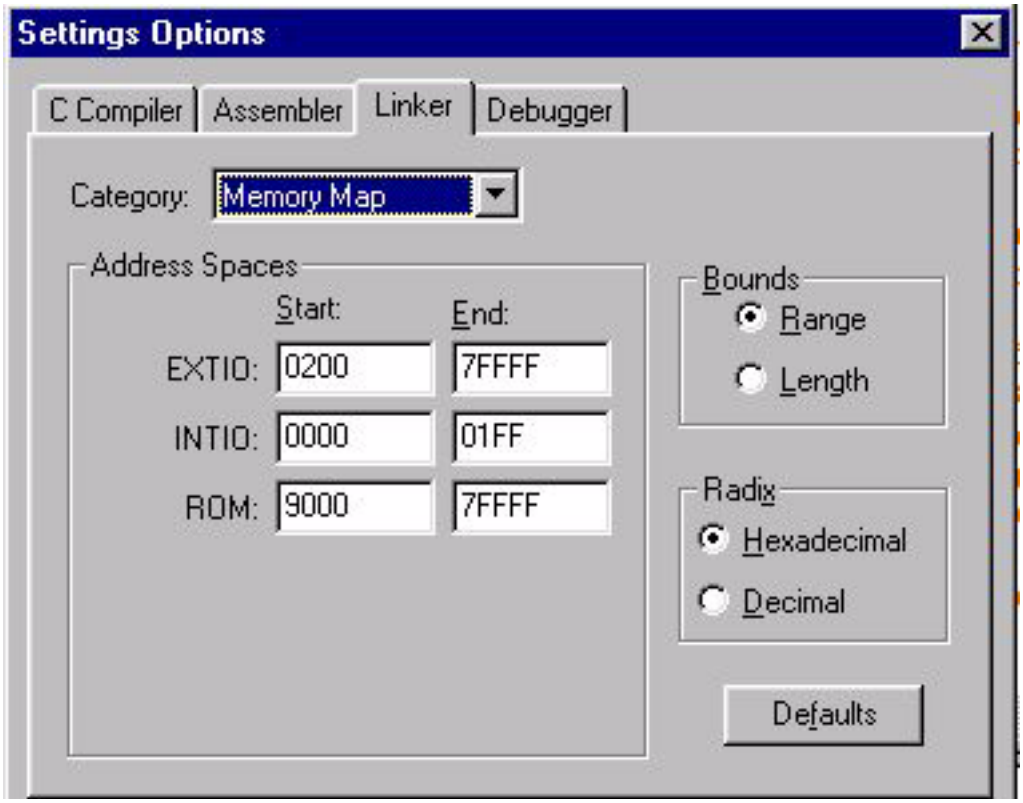

Figure 13. Linker Settings–Memory Map

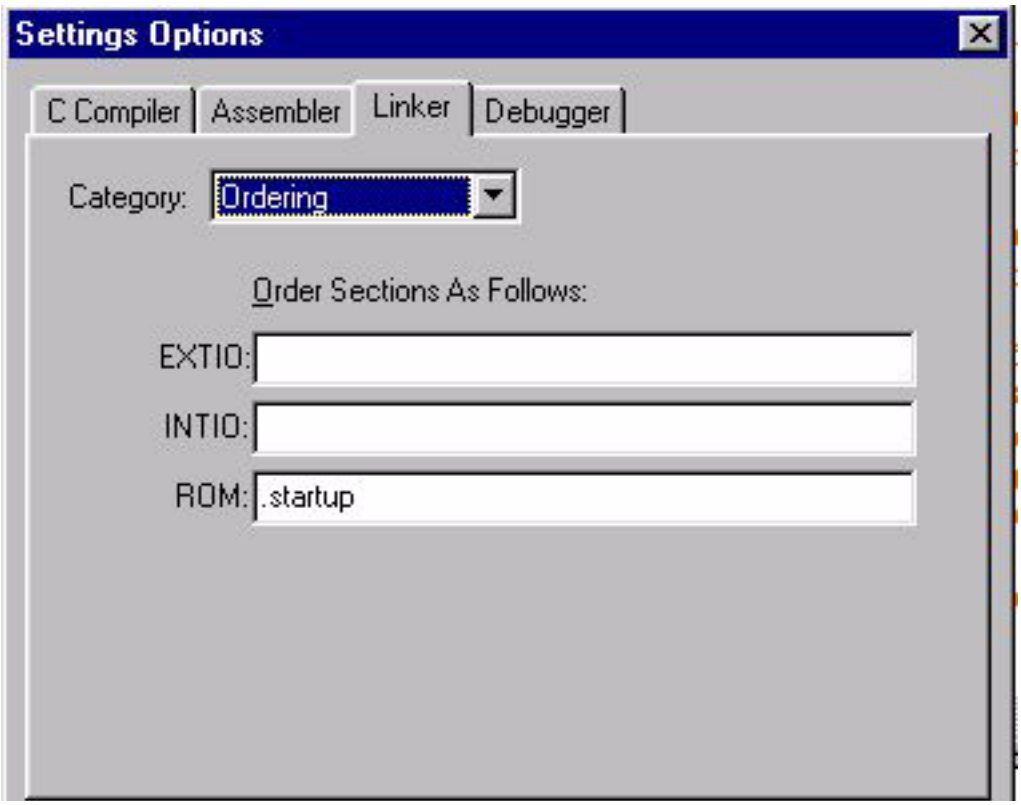

Figure 14. Linker Settings–Ordering

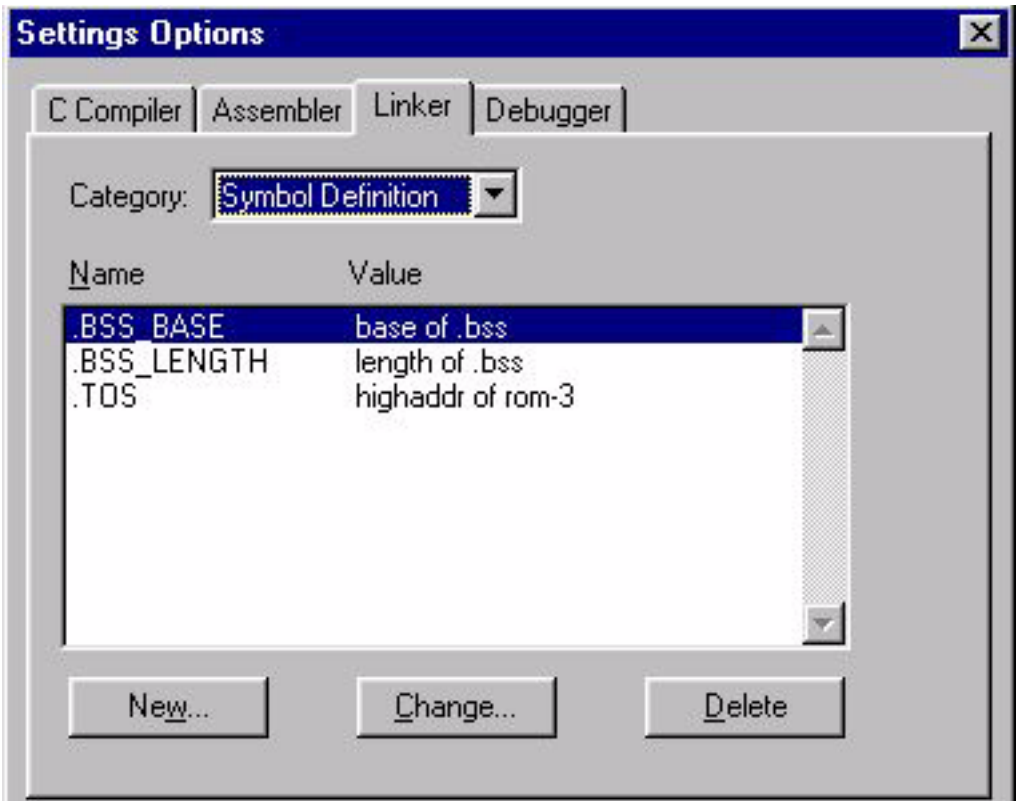

Figure 15. Linker Settings–Symbol Definition

Test Results

We tested both the DMA and HDLC library on the Z80382 evaluation board. For test purposes, we selected only basic functions for the HDLC and DMA module to verify a specific library function. Especially for HDLC, only the Loop Back mode was used to establish a transfer path between the transmitter and receiver. All physical links to certain on-chip peripherals – like GCI or to a certain TDM bus slot – were out of scope of these libraries, but the user can achieve these links by using the corresponding parameters when configuring a channel. The library itself ensures the correct physical setting of all registers, which belong to the DMA/HDLC on-chip peripheral.

We tested all library API functions separately and within a real test application. At this time, there are no known problems. Functions worked exactly as expected.

Even when all library API functions operate correctly, some issues will arise while implementing a real application in a completely different environment. This cannot

be reduced to bad implementation of a library itself, but more to the special conditions of a real application.

REFERENCES

- Z80382/Z8L382 Product Specification
- Z80382/Z8L382 User's Manual
- Z80382 Evaluation Board User's Manual
- Z80380 C-Compiler User's Manual

Glossary

- DMA – Direct Memory Access
- HDLC – High Speed Data Link Control
- ZDS – ZiLOG Developer Studio

Appendix

PCB Artwork

Not applicable.

Schematics

Not applicable.

NOTES

The libraries do not operate in ZiLOG's 'RealOS' operating system. The goal was to offer DMA and HDLC support for applications written for a non-RealOS environment. The example application and all proposed steps to enable DMA/HDLC support within this document do not reflect any customer's application requirements. The customer is responsible for application code to setup both the DMA and HDLC channels according to the mode required for that application.

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form besides the intended application requires a license agreement between both par-

ties. ZiLOG is not responsible for any code(s) used beyond the intended application. Contact your local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer