Application Note

Using the DMA Controller on the ZNEO CPU

AN031402-0712

Abstract

The Z16F Series of MCUs are based on Zilog's ZNEO CPU architecture and features a 4-channel DMA controller that supports internal and external data transfers, namely: memory-to-memory, memory-to-peripheral, peripheral-to-memory, and peripheral-to-peripheral data transfers. This application note demonstrates how to set up and use the DMA controller featured in the ZNEO Z16F microcontroller. The use of DMA is beneficial in most systems in such manner that it frees up more CPU bandwidth that can be used in servicing more important tasks than mere data transfers.

Note: The source code files associated with this application note, AN0314-SC01, have been tested with ZDSII version 5.0.1.

Features

The prominent Direct Memory Access features of the ZNEO CPU are:

- Four independent DMA channels
- Supports memory to memory, memory to peripheral, peripheral to memory, and peripheral to peripheral data transfers
- Operates in either direct or linked list mode
- Byte, word, or quad data transfer length
- DMA and CPU bandwidth sharing control
- Up to 64K transfers (64KBytes, 64KWord, or 64KQual)
- External DMA request and DMA acknowledge signals

Discussion

The ZNEO DMA is used to offload the processor from performing repetitive tasks. It transfers data from one memory address to another memory address, or from one peripheral to another peripheral. These tasks require a read and/or write cycle that is generated by the DMA controller. Each DMA transfer requires a minimum of 2 system clock cycles to execute. Figure 1 shows a block diagram of the DMA Controller for the Z16F MCU, which is based on the ZNEO architecture.

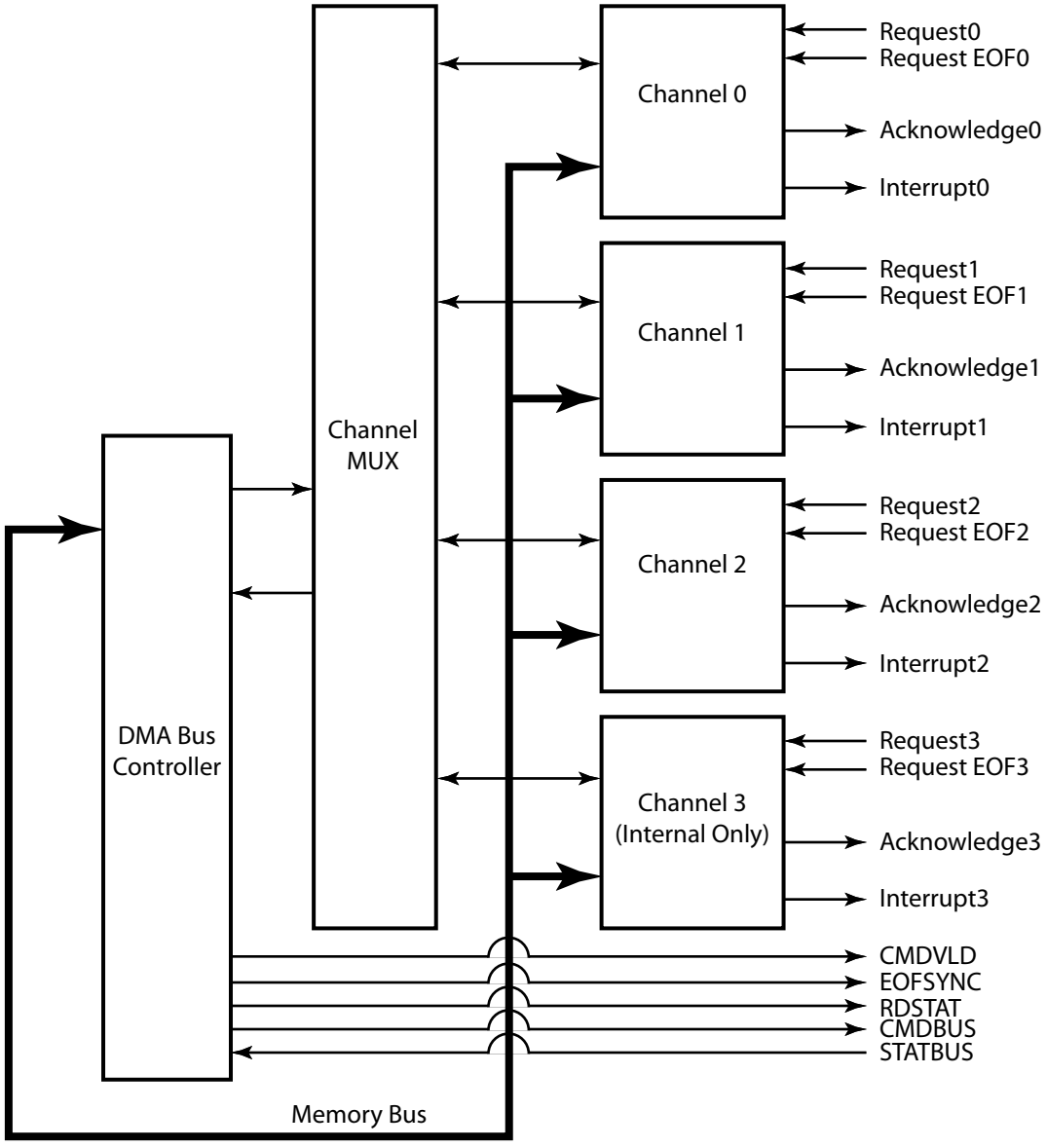

Figure 1. DMA Block Diagram

DMA Register Description

Each DMA channel consists of the following elements:

DMA Control Register (DMAxCTL)	16 bits
Transfer Length (DMAxTXLN)	16 bits
Destination Address (DMAxDAR)	24 bits
Source Address (DMAxSAR)	24 bits
List Address (DMAxLAR)	24 bits

DMA Control Register

The DMA Control Register (DMAxCTL) enables and controls the DMA transfer. Each bit in this register, described in Table 1, defines how the DMA transfer should behave.

Table 1. DMA Control Register (DMAxCTL)

Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Field	DMAxEN	LOOP	TXSIZE		DSTCTL		SRCCTL	
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field	IEOB	TXFR	EOF	HALT	CMDSTAT			
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

15	**DMAxEN (DMA x Enable)** Enables/disables the DMA
14	**LOOP (Loop Mode)** If enabled, this bit prevents the DMA from updating the descriptor when the buffer is closed. This bit is essential when DMA is operating in linked list mode to allow the list to loop by itself without software intervention. This bit is used only when DMA is in linked list mode. Thus, it is mandatory that this bit is reset to zero when running in direct mode.
13:12	**TXSIZE (Transfer Size)** This field defines the width of the data to be transferred. Data transfer width can either be in byte, word, or quad. This field affects the behavior of the destination and source address registers (DMAxDAR, DMAxSAR) in such manner that increments and/or decrements on these registers will vary depending on transfer size. For an 8-bit data transfer, the address registers will increment/decrement by one for each DMA transfer. A 16-bit data transfer will increment/decrement the address registers by two per DMA transfer. While in a 32-bit data transfer, the address registers will increment/decrement by four per DMA transfer.

11:8	**DSTCTL/SRCCTL (Destination/Source Control Register)** These fields define the behavior of the destination and source address registers (DMAxDAR, DMAxSAR) after every DMA transfer. These behaviors can be fixed, increment, or decrement. When set to FIXED, the address register will not be modified on each DMA transfer. When set to INCREMENT, the address register increments after each DMA transfer. When set to DECREMENT, the address register decrements after every DMA transfer. Increment or decrement value depends on TXSIZE.
7	**IEOB (Interrupt on End Of Buffer)** This bit forces the DMA channel to generate an interrupt when the buffer is closed. If the DMA is running in direct mode, an interrupt will also be generated when the TXLN reaches the watermark value.
6	**TXFR (Transfer to New List Address)** This bit determines how the DMA will iterate through the linked list. If set to 1, the DMA will use the LAR in the descriptor as the next descriptor to be loaded. This setup enables looping thru the linked lists. If reset to zero, the DMA will increment the current LAR by 16 and use it as the next descriptor address to be loaded. This bit is used only when DMA is running in linked list mode, and must be reset to zero when operating in direct mode.
5	**EOF (End of Frame)** This bit determines if the current buffer is an end of frame or not. If set to 1, an EOF signal is sent to the peripheral on the last transfer in the buffer to signal the peripheral to close this frame. This is only used for on-chip peripherals, and may also be set if a peripheral requests an end of frame before the buffer transfer is completed.
4	**HALT (Halt after this Buffer)** This bit determines if the next descriptor should be loaded or if the DMA should stop at the end of the current buffer. If set to 1, the DMA will stop after the current buffer is closed. Otherwise, the DMA will load the next descriptor based from the LAR. This bit is used only when DMA is running in linked list mode, and must be reset to zero when operating in direct mode.
3:0	**CMDSTAT (Command Status Field)** This field is exported to the requesting device on the first transfer of a new buffer. It can either be set by a software write or from the DMA reading the descriptor. If EOF is set, this field will contain status information from the peripheral at the end of a buffer. Some peripherals require the use of this field, others do not; see the ZNEO Z16F Series Product Specification (PS0220) for more information.

DMA Transfer Length Register

The DMA Transfer Length Register (DMAxTXLN) specifies how many transfers are required for a given buffer. It is equivalent to the size of the buffer pointed to by SAR. See Tables 2 and 3.

Table 2. DMA Transfer Length High Register (DMAxTXLNH)

Bits	7	6	5	4	3	2	1	0
Field	DMAxTXLNH							
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 3. DMA Transfer Length Low Register (DMAxTXLNL)

Bits	7	6	5	4	3	2	1	0
Field	DMAxTXLNL							
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Each time a transfer occurs, this register is decremented by 1. Upon reaching 0, the DMA concludes that the buffer is already complete and DMA will either stop or load a new descriptor. This register is not affected by TXSIZE, thus it always decrements by 1 regardless of data transfer size. In essence, if TXSIZE defines an 8-bit data transfer, this register will decrement by 1. It will also decrement by 1 if TXSIZE denotes a 16-bit or 32-bit data transfer.

DMA Destination Address Register

The DMA Destination Address Register (DMAxDAR) points to the memory locations in which to store the data transferred from the address pointed to by SAR. See Tables 4 through 6.

Table 4. DMA X Destination Address Upper Register (DMAxDARU)

Bits	7	6	5	4	3	2	1	0
Field	DMAxDARU							
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 5. DMA X Destination Address High Register (DMAxDARH)

Bits	7	6	5	4	3	2	1	0
Field	DMAxDARH							
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 6. DMA X Destination Address Low Register (DMAxDARL)

Bits	7	6	5	4	3	2	1	0
Field	DMAxDARL							
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Each time a transfer occurs, the DAR value will stay fixed, increment, or decrement depending on DSTCTL settings. When set to stay fixed, this register will not change its value regardless of the number of transfers that occurred. When set to increment or decrement, this register will increment/decrement by the size of the transfer defined by TXSIZE. That is, if TXSIZE defines an 8-bit data transfer, this register will increment/decrement by 1. If TXSIZE defines a 16-bit transfer, this register will increment/decrement by 2. This register will increment/decrement by 4 if TXSIZE denotes a 32-bit transfer.

DMA Source Address Register

The DMA Source Address Register (DMAxSAR) points to the memory locations in which to obtain (GET) the data to be transferred. See Tables 7 through 9.

Table 7. DMA X Source Address Upper Register (DMAxSARU)

Bits	7	6	5	4	3	2	1	0
Field	DMAxSARU							
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 8. DMA X Source Address High Register (DMAxSARH)

Bits	7	6	5	4	3	2	1	0
Field	DMAxSARH							
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 9. DMA X Source Address Low Register (DMAxSARL)

Bits	7	6	5	4	3	2	1	0
Field	DMAxSARL							
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Each time a transfer occurs, the SAR value will stay fixed, increment, or decrement depending on SRCCTL settings. When set to stay fixed, this register will not change its value regardless of the number of transfers that occurred. When set to increment or decrement, this register will increment/decrement by the size of the transfer defined by TXSIZE.

DMA List Address Register

The DMA List Address Register (DMAxLAR) points to the memory locations in which the descriptor lists are located. See Tables 10 through 12.

Table 10. DMA X Source Address Upper Register (DMAxLARU)

Bits	7	6	5	4	3	2	1	0
Field	DMAxLARU							
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 11. DMA X Source Address High Register (DMAxLARH)

Bits	7	6	5	4	3	2	1	0
Field	DMAxLARH							
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 12. DMA X Source Address Low Register (DMAxLARL)

Bits	7	6	5	4	3	2	1	0
Field	DMAxLARL							
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The DMA List Address Register is accessed only in linked list mode. For direct mode, however, the upper byte of this register, DMAxLARU, is used to set a watermark interrupt.

DMA Request Select Register

The DMA Request Select Register (DMAxREQSEL) describes the request source of the DMA transfer. Typically, the request source is the peripheral that will initiate a DMA transfer. See Table 13.

Table 13. DMA Select Register (DMAxREQSEL)

Bits	7	6	5	4	3	2	1	0
Field	CHANSTATE				REQSEL			
Reset	0	0	0	0	0	0	0	0
R/W	R	R	R	R	R/W	R/W	R/W	R/W

The DMA Request Select Register also reflects the current state of the DMA channel.

Buffers and Frames

A buffer is an allocation of contiguous memory bytes where the DMA can store or get the transferred data. Buffers must be allocated by software prior to enabling the DMA. A frame can be a single buffer or a collection of buffers.

DMA Watermark

The watermark allows the DMA to generate an interrupt prior to the buffer becoming empty. When operating in direct mode, the upper byte of the DMAxLAR is used as a watermark. If these bits are set to any value greater than zero, the DMAxLARU is compared to the low byte of the decremented transfer length (DMAxTXLN) during a transfer. An interrupt is generated if IEOB = 1 and DMAxTXLNH = 0 and DMAxTXLNL = DMAxLARU.

DMA Bandwidth Selection

The ZNEO CPU provides a way to control access to the busses thru the DMA bandwidth selection. Up to four levels of DMA controller bus bandwidth can be configured thru the CPUCTL register. See Table 14.

Table 14. CPU Control Register

Bits	7	6	5	4	3	2	1	0
Field	CHANSTATE				REQSEL			
Reset	0	0	0	0	0	0	0	0
R/W	R	R	R	R	R/W	R/W	R/W	R/W

DMA bus bandwidth selection is governed by the following stipulations:

- The DMA function must use 100% of its bandwidth
- The DMA function is allowed one transfer for each CPU operation
- The DMA function is allowed one transfer for every two CPU operations
- The DMA function is allowed one transfer for every three CPU operations

DMA Modes

The DMA can operate in one of two modes - direct or linked list mode. Both modes are almost the same, but differ in the way they are loaded. In direct mode, the user code is responsible for loading the setup parameters into the DMA channel registers. While in linked list mode, the user code only needs to create a setup list which will be automatically loaded onto the DMA registers from the memory.

Direct Mode

Direct mode only uses the registers in the DMA for operation. The software writes these registers directly to setup and enable the DMA. Direct mode is entered by directly setting the appropriate bit in the DMAxCTL0 register. The figure below displays the DMA registers and how they point to the buffers allocated in memory.

Linked List Mode

Linked list mode requires the software to allocate buffers and setup a list of descriptors for each buffer. Once this is done, the software writes to the DMAxLAR with the address of the first descriptor. Then the DMA reads the first descriptor into the DMA control and address registers with the exception of the LAR data. It executes the transfers as specified by the descriptor data in the DMA. When the transfers are complete, the DMA reads in the next descriptor in the list and continues executing transfers.

Descriptors

A descriptor is a 16-byte field stored within memory that describes the DMA settings which will be used for linked list operation. It must be aligned on the 16-byte boundaries. Each descriptor follows the format indicated in Table 15.

Table 15. Linked List Descriptor

Address	Even
LAR	CONTROL
LAR + 02H	TXLN
LAR + 04H	DAR High
LAR + 08H	SAR High
LAR + 0CH	LAR High

Software Implementation

This reference design utilizes the ZNEO Z16F2800100ZCOG development kit. It makes use of the ZNEO CPU's DMA controller, particularly, channel 0 of the DMA, for demonstration purposes. The application also uses the UART0 to provide a menu-driven console that allows the user to test out the different modes of operation of the DMA controller. Additionally, peripherals such as the ADC, Timer2, and the UART1 are used to support DMA demonstration. See Appendix A for description on peripheral initialization.

CPU Bandwidth Selection

Prior to any DMA initialization, make sure that CPU busses are set to allocate the desired/required bandwidth for DMA use. If this is not set, the reset value will be used; that is, DMA is allowed one transfer for every three CPU operations. In this application note, different bandwidth selections are used to provide examples for the different values of CPUCTL. The table below itemizes the CPU bandwidth used for each transfer type. The same CPU bandwidth will be used for both direct mode and linked list mode.

Table 16. CPU Bandwidth Selection For Each DMA Transfer Type

Transfer Type	CPU Bandwidth
ADC to RAM	DMA uses 100% of CPU bandwidth.
ERAM to RAM	DMA transfer occurs once every CPU operation.
ADC to UART1	DMA transfer occurs once every 2 CPU operations.
ROM to ERAM	DMA transfer occurs once every 3 CPU operations.
RAM to PWM	DMA uses 100% of CPU bandwidth.

Direct Mode Operation

For direct mode operation, DMA registers are set directly by software. The code that follows demonstrates the steps upon initializing the DMA for direct mode operation.

```
/*****************************************************************
* Function Name:    DMA0_InitDirectMode
* Parameters:       sDMA_Descriptor arg
* Return Type:      -none-
* Description:      Initiates DMA transfer by setting up its registers
*                   directly.
*****************************************************************/
void DMA0_InitDirectMode(sDMA_Descriptor arg)
{
            // 1. Select the request source
            DMA0REQSEL = mucDMA_ReqSel;

            // 2. Set the destination address
            DMA0DAR = msDMA_Settings.DAR;

            // 3. Set the source address
            DMA0SAR = msDMA_Settings.SAR;

            // 4. Set the transfer length (number of bytes/words/quads to tx)
            DMA0TXLN = msDMA_Settings.TXLN;

            // 5. Set water mark, if required otherwise, write to zero
            DMA0LARU = 0x00;

            // 6. Enable DMA0 interrupt, if required
            if(msDMA_Settings.CtrlStat & DMAFLAGS_IEOB)
            DMA0_EnableInterrupt();

            // 7. Write to DMAxCTL. Ensure direct mode before enabling DMA.
            DMA0CTL = msDMA_Settings.CtrlStat & DMA_MASK_DIRECTMODE;
}
/*****************************************************************/
```

When using direct mode, the DMA automatically disables at the end of buffer. DMA registers are also modified, thus, the user needs to re-initialize all DMA registers, if it needs to be restarted. To avoid re-initializing the DMA, using linked list mode is highly suggested.

Note: The LOOP, TXFR and HALT bits of the DMA Control Register are not used in direct mode and should always be reset to zero.

Linked List Mode Operation

For linked list mode operation, DMA registers are set thru the descriptors. The DMA will be enabled upon writing to the LAR. The code below demonstrates the steps on initializing the DMA for linked list operation.

```
/*************************************************************
* Function Name:    DMA0_InitLinkedListMode
* Parameters:       sDMA_Descriptor arg
* Return Type:      -none-
* Description:      Initiates DMA transfer via descriptors.
*************************************************************/
void DMA0_InitLinkedListMode(sDMA_Descriptor arg)
{
            // 1. Select the request source in the descriptor.
            DMA0REQSEL = mucDMA_ReqSel;

            // 2. Set the CONTROL field in the descriptor.
            msDMA_LinkedList0.CtrlStat = arg.CtrlStat;

            // 3. Set the source address in the descriptor.
            msDMA_LinkedList0.SAR = arg.SAR;

            // 4. Set the destination address in the descriptor.
            msDMA_LinkedList0.DAR = arg.DAR;

            // 5. Set the transfer length in the descriptor.
            msDMA_LinkedList0.TXLN = arg.TXLN;

            // 6. Set then the LAR address to point to the next descriptor.
            msDMA_LinkedList0.LAR = (UINT32)&msDMA_LinkedList0;

            // Make sure LOOP is set to one to prevent the DMA to update the
            // descriptor when buffer is closed.
            msDMA_LinkedList0.CtrlStat |= DMAFLAGS_LOOP;

            // Make sure TXFR is set to allow DMA looping of linked list
            msDMA_LinkedList0.CtrlStat |= DMAFLAGS_TXFR;

            // 7.If there are additional descriptors in the list then set
```

```
            // them up using the same procedure listed above.

            // 8. Enable DMA0 interrupt, if required
            if(msDMA_LinkedList0.CtrlStat & DMAFLAGS_IEOB)
            DMA0_EnableInterrupt();


            // 9. After the descriptor has been set up, the software must
            // write the DMAxLAR in the appropriate DMA with the address of
            // the descriptor.
            DMA0LAR = (UINT32)&msDMA_LinkedList0;
}
/*****************************************************************************/
```

In linked list mode, the DMA continuously runs until the buffer is full. Afterwards, the DMA will load the next descriptor in the list and continues to run until a descriptor with the HALT bit set is encountered.

Testing/Demonstrating the Application

The application console must be connected to the RS-232 interface of the development kit (UART0 of the ZNEO MCU) using 57600 baud, no parity, 1 stop bit. A snapshot of the menu is shown in Figure 2; a guide to console commands is shown in Table 17.

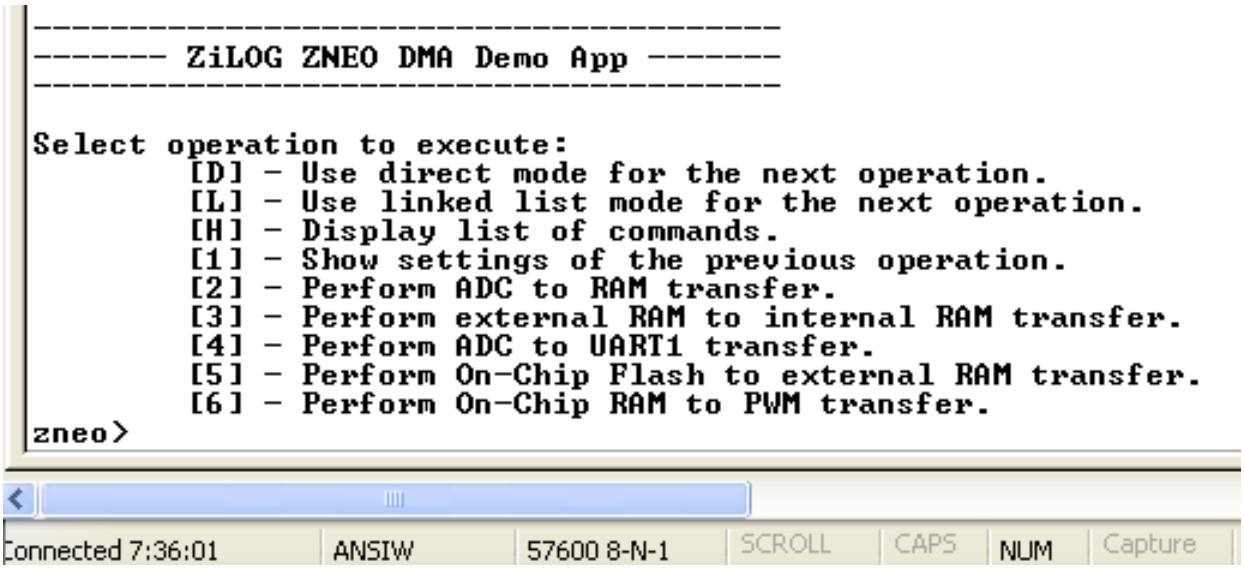

Figure 2. Linked List Diagram

Table 17. Console Commands

Command	Description
D/L	Changes the DMA mode to use on the next transfer operation. If not set, the previous mode will be used. At startup, the default operation is linked list mode.
H	Displays a list of all commands.
1	Displays the current DMA settings.
2 ~ 6	Select and start DMA transfer.

Note: When testing the DMA with an ADC, make sure that J1 on the development board is installed to enable the analog ports on JP4 and R10. This task will, in turn, disable external memory (ERAM and EROM).

Results

Direct mode was tested with IEOB and EOF bits of DMA0CTL set to 1 and no watermark value set. That is, an interrupt will be generated only at the end of the current buffer. While linked list mode was tested using the same settings as direct mode with HALT bit of DMA0CTL set to 1.

Summary

This application note provides a brief discussion on the DMA controller of the ZNEO MCU together with its setup and usage. The software implementation along with this document is easy to customize for any DMA setup.

Other DMA transfers can also be implemented using SPI and I2C transfers, as well as in UART TXD/RXD transfers, but are not presented in this document.

References

The following documents supported the development of this application note; all are available on www.zilog.com.

- ZNEO Z16F Series Product Specification (PS0220)
- ZNEO CPU User Manual (UM0188)
- ZNEO Z16F Series of Microcontrollers Development Kit User Manual (UM0202)

Appendix A. Peripheral Initialization

This section describes the initialization procedure for the peripherals used in DMA transfers.

ADC Initialization

The ADC block should be initialized as it would be for normal ADC conversion, with special attention devoted to the ADC0CTL and ADC0MAX registers, which enable the ADC for DMA transfer. These registers are listed in the code segment that follows; this code demonstrates the ADC initialization routine used for this reference design.

```
/************************************************************************
* Function Name:    ADC_Init
* Parameters:       -none-
* Return Type:      -none-
* Description:      Initializes ADC.
************************************************************************/
void ADC_Init(void)
{
        UINT8 ctr;
        UINT8 mask = 0x01;// mask for Port B inputs
        UINT8 mask2 = 0x01;// mask for Port H inputs

        for(ctr = ADC_MAXCHANNELS; ctr > 0; ctr--)
        {
                    if(ctr <= 7)
                    mask = (mask << 1) | 0x01;
                    else
                    mask2 = (mask << 1) | 0x01;
        }

        if(ADC_MAXCHANNELS <= 7)
                    mask2 = 0x00;

        PBAFL |= mask;// initialize ports for ADC alternate function
        PHAFL |= mask2;

        ADCSST = 0x09;// set ADCSST to contain number of clocks
                    // required to meet at least 0.5usec
        ADCST = 0x02;// set ADCST to contain number of clocks
                    // required to meet at least 1usec
        ADCCP = 0x00;// ADC clock == system clock == 5.52960 MHz

        ADC0CTL = 0x70 | ADC_STARTCHANNEL;// convert on read, internal vref
        ADC0MAX = ADC_MAXCHANNELS;

        ADC0CTL |= 0x80;// enable adc
}
/************************************************************************/
```

ADC0 Control Register 0 (ADC0CTL)

Convert on Read (CVTRD0). This bit must be set to 1 to let the DMA control data transfers from ADC0D (ADC0 Data) to memory. By definition, this bit enables continuous ADC conversion wherein the ANAIN field increments until it reaches the value set in the ADC0MAX register. Upon reaching the ADC0MAX value, ANAIN resets to its initial value, i.e., the value that ANAIN was originally set to when the ADC started.

Analog Input Select (ANAIN). Upon initialization, this field must be set to the ADC channel where conversion needs to start. If CVTRD0 is set, this field increments on every conversion until it reaches the ADC0MAX value.

ADC0 Max Register (ADC0MAX)

The ADC0 Max Register determines the highest channel number that Convert On Read (CVTRD0) increments to. This register needs to be greater than or equal to the initial value set in ANAIN.

Timer2 Initialization

The timer must be initialized in the same manner as setting up a timer for normal operation. The code below demonstrates the Timer2 initialization routine used for this reference design.

```
/*************************************************************
* Function Name:    TMR2_InitPWMSingle
* Parameters:       -none-
* Return Type:      -none-
* Description:      Initializes TIMER2 for PWM operation.
*************************************************************/
void TMR2_InitPWMSingle(void)
{
  T2CTL1 &= ~0x80;            // 1. Disable the timer
  T2CTL1 = 0x13;              // 2. Configure timer for PWM mode, prescale=4
  T2CTL0 = 0x00;
  T2H = 0x00;                 // 3. Set the starting count value
  T2L = 0x01;
  T2R = RELOAD;               // 4. Set the timer reload value
  T2PWM = RELOAD / 2;         // 5. Set PWM registers
  PCAFH &= ~0x80;             // 6. Set the timer input
                              // (and output port pin, if needed)
  PCAFL |= 0x80;
  T2CTL1 |= 0x80;             // 7. Enable the timer
}
/*************************************************************************/
```

This timer is configured for PWM single output mode, without a delay in between assertion and de-assertion of the two PWM output pins. A timer interrupt will occur only on a

reload event; however, timer interrupt is disabled (or not set) in the interrupt controller - this is where DMA will interfere to update the PWM values. The same initialization sequence may be used for the other timers - Timer0 and Timer1, if desired.

UART1 Initialization

The UART1 must be initialized in the same manner as it is for a normal operation. The code below demonstrates the UART1 initialization routine used for this reference design.

```
/*************************************************************
* Function Name:    UART1_Init
* Parameters:       -none-
* Return Type:      -none-
* Description:      Initializes UART1 (no parity, 1 stop bit).
*************************************************************/
void UART1_Init(void)
{
            PDDD |= 0x30;     // Setup ports for alternate function
            PDAFL |= 0x30;
            PDAFH &= ~0x30;

            U1BRH = (UINT8)((BAUDRATE & 0xFF00) >> 8);// Setup baud rate
            U1BRL = (UINT8)((BAUDRATE & 0x00FF) & 0x00FF);

            DI();
            IRQ2 |= 0x40;
            IRQ2ENH |= 0x40;
            U1CTL1 = 0x20;    // set RDAIRQ
            U1CTL0 = 0xC0;    // Receive Enable, No Parity, 1 Stop bit
            EI();
}
/*************************************************************************/
```

DMA transfer to/from UART1 is configured separately - one for receive data, and another for transmit data. If transmit data is to be moved via DMA, the transmit interrupt must be disabled in the interrupt controller. If receive data is to be moved via DMA, the RDAIRQ bit in the UART Control 1 register must be set. This disables receive data interrupts while still enabling error interrupts. The receive interrupt must be enabled in the interrupt controller only for processing error conditions received.

Customer Support

To share comments, get your technical questions answered, or report issues you may be experiencing with our products, please visit Zilog's Technical Support page at http://support.zilog.com.

To learn more about this product, find additional documentation, or to discover other facets about Zilog product offerings, please visit the Zilog Knowledge Base at http://zilog.com/kb or consider participating in the Zilog Forum at http://zilog.com/forum.

This publication is subject to replacement by a later edition. To determine whether a later edition exists, please visit the Zilog website at http://www.zilog.com.

Warning: DO NOT USE THIS PRODUCT IN LIFE SUPPORT SYSTEMS.

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer

 Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering.

ZNEO is a trademark of Zilog, Inc. All other product or service names are the property of their respective owners.