EE 3232 DIGITAL SYSTEM III
6.6 DIRECT MEMORY ACCESS

March 2000 CPD

OBJECTIVE

To implement and verify software initiated DMA for memory to IO, memory to memory and IO to IO transfers.

REFERENCES

1. “Embedded Microprocessor System Design”, K.L. Short, Chapter 19, pp. 689 - 703.
2. "The 80C188XL Microprocessor Users Manual”, Chapter 10, available in the lab, also at http://developer.intel.com/design/intarch/manuals/
3. Section 8.1, “The 82C206 Integrated Peripheral Controller”, also at http://www.chips.com/prodframe.htm (Select mature products and scroll to the 82C206)
4. “The SBC188 Users Manual”, available in the lab.
5. “Paradigm Debug/RT - 186 Users Guide”, available in the lab.
6. “Paradigm Locate Reference Manual”, available in the lab.

EQUIPMENT

1. SBC188 board.
2. Host PC.
3. IO interface board.
4. Two 10-pin ribbon cables.
5. One 26-pin ribbon cable.
6. Stepper motor.

BACKGROUND

Direct memory access or DMA is a vehicle for transferring data between combinations of IO and memory without the direct intervention of the CPU. The 80C188XL microprocessor contains two DMA channels on-chip that support memory-memory, memory-IO, IO-memory and IO-IO transfers. The start of a transfer may be initiated by a software trigger, synchronized to the internal timer, TIMER2, or synchronized to an external pin.

The DMA controller must first be initialized before a transfer takes place. The programmer needs to select the DMA control words to program,

- The source and destination (i.e., memory, IO),
- Byte or word transfers,
- 20-bit pointer (physical address) for starting source address
- 20-bit pointer (physical address) for starting destination address,
- 16-bit count value (i.e., the number of bytes/words to transfer),
- Pointer indexing, i.e., auto-increment, auto-decrement, or none,

- Enable or disable interrupt generation on terminal count, TC.

You may consult Sections 19.3, 19.4, 19.5 and 19.6 in Short , or Chapter 10 in the 80C188XL Microprocessor Users Manual for programming details on the internal DMA controllers. There are two program examples,

L361.C // Memory to memory transfer terminates on terminal count, TC, without // interrupt generation.
L362.C // Memory to memory transfer terminates on TC with interrupt generation.

In L361.C, the procedure **void DMA_MEM_TO_MEM(char *SRC, char * DST, unsigned SA, unsigned NO_BYTES)**, initializes DMA0 for a (unsynchronized) memory to memory transfer of bytes and then arms (or initiates) the transfer without interrupt generation. The starting address of the source block is passed as a pointer, SRC, the starting address of the destination block is passed as a pointer, DST, the segment address of the two blocks (both blocks are assumed to reside in the same segment!) is passed as SA, and the number of bytes to be transferred is passed as NO_BYTES.

To program DMA Controller 0 for memory to memory transfers, the following steps are followed.

The procedure computes the 20-bit physical address as, PA = EA + 16*SA . EA is the 16-bit effective address. SA is the 16-bit segment addresses. PA is a 32-bit long variable which holds the 20-bit address in the least significant 20-bits. The procedure first writes the source (physical) address to the 16-bit ports : D0SRCL (low order 16-bits of PA) and D0SCRH (hi-order 16-bits of PA). Then the procedure writes the destination (physical) address to the 16-bit ports: D0DSTL (low order 16-bits of PA) and D0DSTH (hi-order 16-bits of PA).

The number of bytes to be transferred is written to 16-bit port D0TC. Then the DMA0 control word is written to 16-bit port DMA0CON to initiate the transfer. Consult Chapter 10 in the 80C188XL Microprocessor Users Manual available in the lab. The bit format for the control word is selected as follows.

DMEM	DDEC	DINC	SMEM	SDEC	SINC	TC	INT	SYN1	SYN2	P	IDRQ	0	CHG	STRT	WORD
d_{15}	d_{14}	d_{13}	d_{12}	d_{11}	d_{10}	d_9	d_8	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0

d_{15}:d_{13} = 101 Select memory as destination with auto-incrementing pointer.
d_{12}:d_{10} = 101 Select memory as source with auto-incrementing pointer.
d_9:d_8 = 10 Selects termination on terminal count without interrupt generation.
d_7:d_6 = 00 Selects unsynchronized transfer, i.e., transfers start when the the DMAC is armed. (Set to 01 to synchronize transfers to TIMER2)

d_5	=	1 or 0	Selects channel priority.
d_4	=	0	For unsynchronized DMA transfers. Set to '1' for internal DMA requests that are synchronized to TIMER2.
d_2:d_1	=	11	Arms the DMA controller.
d_0	=	0	Selects byte transfers.

In L362.C, the procedure **void DMA_MEM_TO_MEM(char *SRC, char * DST, unsigned SA, unsigned NO_BYTES)**, initializes DMA0 in the same manner as L361.C except bit d_8 = 1 in the DMA control word and DMA0 interrupts are unmasked by writing 0 to the 16-bit port DMA0CON. This results in the generation of an interrupt by the DMA controller when the terminal count condition is encountered, i.e., after all the bytes have been transferred.

The rest of the program contains initialization code for the interrupt vector table, the interrupt handler and some alterations to the main program to enable interrupts and wait for an interrupt to occur. It should be evident from the listing that DMA0 generates an interrupt with type = 10. Study each of these programs. You may use these as templates for solving the following problems!

PREPARATION

Consult Sections 19.3, 19.4, 19.5 and 19.6 in Short , or Chapter 10 in the 80C188XL Microprocessor Users Manual.

1. Write a procedure **void T2_INIT(unsigned T))** that initializes TIMER2 to continuously count up to the T2CMPA value in T microseconds. Recall that TIMER2 is clocked internally at 4.0 MHz.

2. Using T2_INIT(), write a procedure **void DMA_MEM_TO_IO(char *SRC, unsigned SA, unsigned PORT, unsigned NO_BYTES, unsigned MILLISECS)**, that initializes DMA0 for a synchronized (to TIMER2) memory to IO transfers. The starting address of the memory block to be transferred is passed as an effective address or pointer, SRC, and segment address, SA. The address of the output port is passed as PORT. The number of bytes to be transferred is passed as NO_BYTES and TIMER2 is to be initialized to time-out continuously at intervals of MILLISECS milliseconds where 1 < MILLISECS < 17.

3. Using DMA_MEM_TO_IO() write a program that writes the array { 0x00, 0x44, 0x10, 0x32, 0x11, 0x55, 0x01, 0x23, 0x00, 0x44, 0x10, 0x32, 0x11, 0x55, 0x01, 0x23, 0x00, 0x44, 0x10, 0x32, 0x11, 0x55, 0x01, 0x23 ... } to output port KEY_PC at 16 millisecond intervals. ***Notes***: This is the stepper motor ***half step*** sequence. Notice that the first 8-elements of the array are repeated. Writing these 8-elements in sequence to the stepper causes the stepper to rotate 4-steps, i.e., 1-revolution. Configure the 8255A with PC as a mode-0 output port. Connect the stepper motor and observe!

4. Modify the program such that DMA0 generates an interrupt on terminal count, thus providing a signal to the main program that the DMA transfer is complete. Now modify the program such that the array is written to the port KEY_PC 10-times, i.e., in a

program loop you will initialize 10 DMA transfers to take place in sequence. **NOTE :** you should not start a new transfer until the previous transfer has completed.

5. Write a procedure **void DMA_IO_TO_IO(unsigned SRC, unsigned DST, unsigned NO_BYTES, unsigned MILLISECS)**, that initializes DMA0 for a synchronized (to TIMER2) IO to IO transfer. The address of the input port is passed as SRC and the address of the output port is passed as DST The number of bytes to be transferred is passed as NO_BYTES and TIMER2 is to be initialized to time-out continuously at intervals of MILLISECS milliseconds.

6. Using DMA_IO_TO_IO() write a program that reads input port KEY_PA and writes to output port KEY_PC.

7. Modify the program in Part 6 such that DMA0 generates an interrupt on terminal count, thus providing a signal to the main program that the DMA transfer is complete.

EXPERIMENT

1. Load, execute and verify the programmed operation of L361.C and L362.C .

2. Load, execute and debug the program developed in Parts 2 and 3 of the preparation.

*3. Load, execute and debug the program developed in Parts 5 and 6 of the preparation.

SUMMARY

1. What happens to the microprocessor system bus during a DMA transfer?

2. List the advantages of using DMA.

3. A DMA application for a memory to IO transfer involves waveform generation at the output of a digital to analog converter. Can you explain the steps necessary to modify the waveform generation problem in Experiment 6.4 so that the transfers are initiated via DMA?