Tools and techniques of microprocessor data transfer

by GARY SAWYER
Motorola
Phoenix, Arizona

INTRODUCTION

This paper discusses I/O transfer from four different viewpoints. The first, "I/O Data Transfer Techniques", contrasts the two basic methods of I/O data movement. Having discussed the movement of data, "I/O control" focuses in on how the data movement is governed. "I/O Interface Hardware" takes a closer look at specific hardware that may be used to interface to the MPU. Finally, "An Example of I/O Transfer" concludes the discussion showing the software and hardware required in an actual transfer.

I/O DATA TRANSFER TECHNIQUES

The I/O capability of a microprocessor is a key standard of measure. As microprocessors mature, more techniques are becoming available at better throughput speeds. It, therefore, becomes useful to put these techniques in perspective by categorizing how data is shipped through the microprocessor system. The first, and most commonly used, technique is to ship data through the microprocessor (MPU) wherein the MPU acts as a data funnel to the outside world. The second technique, direct memory access (DMA), transfers data directly between memory and the outside world circumventing the MPU. Following is a description of each with a discussion of associated vices and virtues.

The first technique considered moves data through the system via the MPU under program control. The MPU then becomes the focus for data movement between the peripheral and memory. Figure 1 shows, at a block diagram level, where data is moving during an I/O transfer. It is comprised of four basic hardware blocks: the MPU, memory, I/O interface, and a peripheral. If, for example, data is to be transferred from the peripheral to the system memory, the first link in the chain is the I/O interface. The characteristics of this interface are, as expected, a function of (1) the data and control requirements of the peripheral, and (2) the processor used in the system. A discussion of I/O interface hardware is an important topic and is treated in the section "Interfacing to the Microprocessor". Once data has been shipped to the interface, the MPU reads the data from the interface device. The MPU may now complete the I/O transfer by storing the data in the desired memory location. If, on the other hand, data is transferred from memory to peripheral, the sequence of events is reversed; the MPU reads the data from memory and stores data into the I/O interface for transfer to the peripheral. Data is, therefore, transferred from block to block under program control.

Microprocessors available today use two general classes of instructions to move data: One, use of an Input or Output instruction or, two, a memory reference instruction. When, for example, the MPU is given an Input or Output instruction, the microprocessor will issue control signals and address the desired I/O interface device. The second class of instruction to access I/O is memory reference. Here the MPU interface is assigned a memory address and is accessed during any instruction that specifies the defined I/O interface.

Direct memory access is the second alternative to I/O data transfer. The MPU is circumvented and the data moves directly between memory and peripheral. Figure 2 is a representation of how data will move through the system using DMA techniques. Here, the microprocessor is off the bus and the DMA interface transfers directly to/from the memory. This requires the DMA interface to (1) override the MPU operation causing it to go into an off (high impedance) state, and (2) generate memory address and control signals for the desired data transfer.

Comparison of the two techniques is largely a function of speed and hardware. DMA, for example, will generally require more hardware due to the additional control tasks. On the other side of the coin, DMA is consistently the faster of the two transfers. Here, the data transfer is normally limited only by the cycle time of the memory. This feature becomes valuable when the MPU is not fast enough to handle transfer under program control. When funneling data through the MPU, the I/O interface is straightforward, but the transfer rate is now a function of instruction execution time. In this case the MPU becomes the limiting factor. The choice, as usual, is in the hands of the designer. He may require a fast DMA channel at the expense of hardware, or he may simplify the interface transferring the work load to software.

I/O CONTROL

A major aspect of I/O data transfer is that of control. Where is the I/O transfer initiated? How long does it continue? Which peripheral is to gain access to the system? These questions are recurring in virtually every micropro-
The great strength of a microprocessor is the flexibility of the program. Programs are more than a list of instructions commanding a fixed sequence of operations. These programs can be written to adjust to external events, query peripherals for service or respond to hardware service requests. The list is unending, therefore, the question of control could be answered in a word—software. But hardware certainly has its place in control of I/O transfer. Following is a discussion of hardware/software control options in context with "MPU funneled" transfer and DMA transfer.

Consider first the control of I/O transfer when data is transferred through the MPU as shown in Figure 1. Here the I/O transfer may be initiated by either software or under interrupt control. Interrupts may be issued to the microprocessor from peripherals (either directly or via the I/O interface) to inform the MPU of a request for service. When service is granted, the MPU breaks away from the current program, saves its status, and begins an interrupt service routine. At this point, the MPU needs to determine the source of the interrupt (normally multiple interrupts outnumber the interrupt inputs available at the MPU). This can be done by either hardware or software. Hardware can be used to prioritize all interrupts into an 8-bit word for the MPU to read or software can poll each I/O interface to determine where the interrupt originated.

Control options are equally flexible for DMA types of transfer. Here, the software or the DMA peripheral can initiate an I/O transfer. The user may choose to specify the beginning address and length of transfer with software by loading "control" words into the DMA circuitry. At the other extreme, hardware may be the dominant force. The DMA hardware can conceivably initiate the transfer, generate addresses, define direction of data flow, and length of the DMA transfer. Here, again, the designer has hardware/software options to minimize his system while maximizing the I/O transfer.

A notable example of programmable interface devices available today is the MC6820 peripheral interface adapter (PIA) offered by Motorola. Figure 3 shows the PIA between the MPU and peripheral world. Notice that 16 data lines and 4 control lines are available to interface to a variety of peripherals. Each one of the data signals may be programmed to act as inputs or outputs in any combination. A user could, therefore, tie a number of input or output peripheral devices to a single PIA.

A unique feature of the PIA is the programmable control segment of the interface. The four control signals may be used by the designer to inform either the MPU or peripheral that an I/O transfer is occurring. If, for example, the peripheral transfers data to the MPU via a PIA, a pulse shipped in parallel to the control input will cause the PIA to generate an interrupt to the MPU. Should the MPU need to output data to the peripheral, the data may be stored into the PIA with a memory reference instruction. The PIA will save and transfer this data from the MPU data bus onto the peripheral data bus. The PIA can, for example, be programmed to then generate a control pulse to the peripheral informing the peripheral of new data. As a result, a data transfer to a peripheral with an equivalent "data present" pulse can be accomplished with a single STORE instruction by addressing the appropriate PIA.
A closer view of the PIA will reveal how an interface device may be “programmed” by the microprocessor. Figure 4 shows six registers internal to the PIA divided into “A” and “B” sides. Each side of the PIA contains 8 data signals, 2 control signals, and three 8-bit registers. The user may, therefore, program peripheral data and control signals by loading words into respective PIA registers.

Peripheral data signals, PAO-PAT, may, for example, be programmed as inputs or outputs by loading the “A” data direction register. Each logic “0” of the data direction register will then define the respective peripheral data signal to be an input (the converse is true for a logic “1”). Similarly, the characteristics of the control signals, CA1 and CA2, are programmable by loading a word into the “A” control register. The control lines can be used as an input to the PIA to generate an MPU interrupt on either a rising or falling edge (if desired, the control input may also be masked off). As an output, the control signal may be programmed to act as a strobe or active level when moving data through the PIA. The same argument follows for the “B” side of the PIA. Loading data words into the “B” data direction register characterizes PB0-PB7. Likewise, the “B” control register defines how CB1 and CB2 shall react.

**DMA transfer**

Design of a DMA interface requires close examination of the MPU in the system. In a typical direct memory access configuration the MPU and DMA share the system bus for control of the memory transfer. When the MPU is executing instructions the DMA circuitry is effectively off the bus. When the DMA transfer is initiated the MPU must be switched off the bus as the DMA interface switches on. The manner in which the MPU is removed from the bus becomes a major aspect of the DMA transfer.

A representative example of the mechanics of a DMA transfer is shown in Figure 5 using the Motorola MC6800 MPU. A number of techniques are available to control the MC6800 during a DMA transfer—here the Halt signal provides control over the MPU. The characteristics of the Halt line are such that, when low, the address, data and read/write signals go into a three-state condition at completion of the current instruction. When the Halt is recognized and the MPU in three-state, the MPU will bring the Bus Available signal high. The Halt and Bus Available signals of the MC6800 therefore become convenient DMA controls. Referring to Figure 5, the DMA Request initiates the transfer. Depending upon when the request is made with respect to the current instruction, the MPU will respond with a DMA Acknowledge signal within 2-14 µsec. When the Acknowledge is seen, the DMA interface is then free to take over the bus. At this time, the DMA circuitry has complete control over the memory transfer until the DMA Request returns to a high state. This requires the DMA circuitry to formulate desired addresses, to tie into the data bus, and generate the necessary control (R/W). When the DMA transfer is completed the interface switches off the bus and DMA Request returns to an inactive high state. With Halt inactive the MPU will then switch back onto the bus and continue program execution.
AN EXAMPLE OF I/O TRANSFER

Operation of an I/O transfer is best demonstrated by example. Consider the system seen in Figure 6 showing the Motorola PIA interfaced to a keyboard and display. The "A" side of the PIA is tied to the keyboard representing inputs to the MPU. Conversely, the "B" side will be used to output data to a 16 character display. The keyboard interface is comprised of (1) eight data lines tied directly to PIA lines PA0-PA7, and (2) a keyboard strobe to CA1 to inform the MPU of new input data. The display interface uses PB0-PB7 to transfer both data and control functions (backspace and clear). The control lines are configured into a handshake mode with CB2 generating a "data present" signal when new data is displayed. CB1 then acknowledges with "data taken" to complete the handshake loop.

The MPU/PIA interface of Figure 6 shows data, address, and interrupt signals. An important point to mention is that the PIA is assigned a memory area by virtue of the address lines into the device. The chip select inputs (CS0, CS1, and CS2) are used to enable the PIA and the register select inputs (RS0, RS1) specify registers within the PIA. The MPU may, therefore, access the PIA registers with the following addresses:

<table>
<thead>
<tr>
<th>Address (HEX)</th>
<th>PIA Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000</td>
<td>&quot;A&quot; data direction (&quot;A&quot; control register bit 2 = 0)</td>
</tr>
<tr>
<td></td>
<td>&quot;A&quot; data (&quot;A&quot; control register bit 2 = 1)</td>
</tr>
<tr>
<td>8001</td>
<td>&quot;A&quot; Control</td>
</tr>
<tr>
<td>8002</td>
<td>&quot;B&quot; data direction (&quot;B&quot; control register bit 2 = 0)</td>
</tr>
<tr>
<td></td>
<td>&quot;B&quot; data (&quot;B&quot; control register bit 2 = 1)</td>
</tr>
<tr>
<td>8003</td>
<td>&quot;B&quot; control</td>
</tr>
</tbody>
</table>

Having established the general scenario, the discussion continues with programming examples of the PIA initialization, keyboard input and display output using the Motorola M6800 instruction set.

Initialization

As mentioned previously, the PIA is entirely programmable. The interface seen in Figure 6 requires the following definition of PIA signals:

1. PA0-PA7 are inputs.
2. The PIA is to generate an interrupt to the MPU on the rising edge (0→1) of the CA1 input.
3. CA2 is unused.
4. PB0-PB7 are outputs.
5. The PIA is to generate an interrupt to the MPU on a falling edge (1→0) of the CB1 input.
6. When data is written into PIA (from MPU) to the display, a CB2 strobe occurs.

Programming the PIA interface becomes a series of LOAD and STORE instructions. LOAD brings the desired data pattern into the MPU and STORE ships the word to the addressed PIA registers. The initialization program of the PIA then becomes the following:

```
Program* Comments
COM $8002** Complement location 8002—defines PB0-PB7 as outputs
LDAA $C7*** Load accumulator A with a value of C7
STAA $8001 Store accumulator A in location 8001—defines CA1 and CA2 characteristics
LDAA $ED Load accumulator B with a value of ED
STAA $8003 Store accumulator B in location 8003—defines CB1 and CB2 characteristics

* All registers are cleared at power-on reset.
** $ denotes hexadecimal.
*** # denotes immediate addressing in which the subsequent character is data instead of address.
```

Input data transfer

Having initialized the PIA, data is moved from the peripheral to the memory by a sequence of LOAD and STORE instructions. But, the movement of data is only half the story, control of the transfer must also be considered. In the example of Figure 6, the system is assumed to be under interrupt control. When an interrupt occurs the MPU jumps to a defined area of memory to begin the interrupt service program. The service program must first determine the origin of the interrupt by polling the PIA. The PIA has bit positions reserved in the control registers (CRA7, CRA6, CRB7, CRB6) that the MPU may read to ascertain the source of the interrupt. In our example, if the keyboard "strobes" the PIA, an interrupt is sent to the MPU and bit 7 is set high in the "A" control register (CRA7). Likewise, a "data taken" interrupt from the display will set bit 7 of the "B" control register (CRB7). The "polling" routine therefore becomes:

```
LDAA $8001 Load contents of "A" control register.
BMI KEYBRD If CRA7 = 1, go to keyboard routine.
LDAA $8003 Load contents of "B" control register.
BMI DISPLAY If CRB7 = 1, go to display routine.
```

The "polling" sequence of the interrupt service program is nothing more than a read of the control register followed by a conditional branch. If the interrupt bits are set the program branches to the appropriate peripheral routine. If the condition is not met, the "polling" routine continues inspection of PIA. Should additional peripherals be added to the system the software adjusts simply by continuing the poll. Also note the interrupt priority is built into the software by the polling sequence. The order in which the peripherals are polled implicitly defines the priority.

As another alternative, software can poll all peripherals,
evaluate the peripherals requesting service, then branch to the desired routine. In this manner the MPU acquires full visibility of outstanding service requests. Based upon the combination of requests the program can then decide which service routine to enter. The important point to remember is the flexibility of I/O transfer under software control.

Having determined the source of the interrupt, the program branches to the desired peripheral routine. If, for example, the keyboard service routine is entered the MPU performs an input data transfer. A single data transfer to memory may be accomplished in two instructions:

LDAA $8000 Load keyboard data.
STAA $0100 Save in memory location 0100.

When the MPU reads the PIA data, the PIA will automatically reset the corresponding control bits (i.e., bits CRA7 and CRA6 are cleared when reading "A" data), and clear the interrupt to the MPU. The interrupt service routine is then completed with a return from interrupt (RTI) instruction. The cycle is complete: the interrupt was acknowledged, PIA's polled for service, selected peripheral serviced, interrupt conditions reset, and the MPU returned to its operating program.

Output data transfer

Data may be transferred from memory to I/O under control similar to data input. Transfer may be initiated by the software or hardware. In this case, the program initiates the data transfer and the display responds with a "data taken" pulse to signify when another character may be sent. The actual data transfer can be done in two instructions:

LDAA $0100 Load data from memory location 0100.
STAA $8002 Store data into location 8002.

The STAA instruction will load new data into the PIA "B" data register. The PIA will then transfer the data and a "data present" pulse to the display.

Block transfer

The I/O transfer demonstrated in the example shows only a single data transfer to give the reader a feel for data movement under program control. To transfer multiple words or large blocks of data between the peripheral and memory, the program requires more management, but the technique remains the same. The programming ease and speed of I/O data transfer becomes largely a function of the MPU under use. MPU features such as available addressing modes and instruction set become important tools for efficient transfer.

As an example of multiple data transfer, the Motorola MC6800 coupled with the PIA can perform an input operation under full program control as shown below:

```
RDLOOP  LDA $8001 Read PIA control Register 4usec
BPL RDLOOP Branch to RDLOOP if Bit 7 is plus (0) 4usec
STAA OFFSET,X Store data at address defined by [index reg + offset] 6usec
DEX Decrement index register 4usec
BNE RDLOOP Branch to RDLOOP if not equal to zero Total 26usec
```

The first two instructions loop until a request for transfer is received from the peripheral. The request is made to the control input, CAl, of the PIA which then sets CRA7 of the control registers. The LDAA/BPL instructions monitor the control register inspecting the most significant bit (CRA7). The program remains in this two instruction RDLOOP until CRA7 is set to a logic 1. When set, the program breaks out of the loop to begin the data transfer. The MPU reads the data from the peripheral at PIA address 8000 hex. When the data is read into the MPU, the PIA automatically resets the CRA7 bit to zero. The data is then stored into memory using the MC6800 indexed addressing mode. Here the OFFSET (byte 2 of the instruction) is added to a 16-bit index register internal to the MPU. The resulting 16-bit word is used to address the destination of the data. The next instruction, DEX, decrements the index register in preparation for the next I/O transfer. The BNE instruction conditionally branches back to the RDLOOP until the index register is decremented to zero. Once back in the RDLOOP the program cycles waiting for another peripheral service request. When the request is made, CRA7 is set and the program enters the next byte into memory. The cycle is continued until the result of the DEX instrumentation is zero. When the index register is decremented to zero the transfer is complete and the routine is exited.

This multiple word transfer through the MPU is a good contrast to the interrupt controlled transfer shown in the keyboard example. Here the interrupts are disabled and the program polls the peripheral for service via the PIA. Notice that the peripheral need not be synchronous with the program due to the RDLOOP. The program can complete the full cycle in 26usec. If the peripheral is not prepared for another transfer the program will simply cycle in RDLOOP until a peripheral request is made. Also, note that the length of transfer and location of data storage are varied by presetting the index register prior to entry into the routine. The resulting I/O transfer can move a block of data from a peripheral into contiguous memory locations at up to a 38.5K byte rate.
SUMMARY

In microprocessor based systems today, data movement is commonly an important aspect of system operation. As new applications evolve, users will be evaluating microprocessors with a critical eye toward I/O transfer. The number of instructions is less important than the nature of the instruction and usable addressing modes. How quick can the MPU respond to a peripheral interrupt and how is the interrupt managed? What is available from the vendor in the way of interface devices? The list of questions extends in proportion to the needs of the user. As semiconductor houses continue to move into the second and third generation microprocessors, the answers will be easier and faster.