ELEC 464 : MICROCOMPUTER SYSTEM DESIGN
FALL 1995

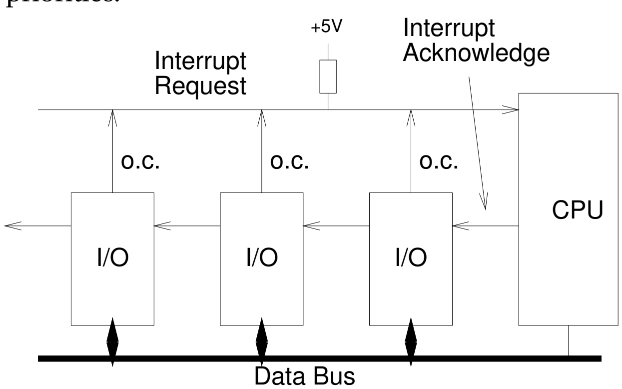
High-Performance I/O Strategies

This lecture deals with a few topics related to high-performance I/O strategies. Direct memory access (DMA) allows a peripheral to transfer data directly to/from memory. Bus arbitration allows multiple DMA peripherals or interrupt sources to ensure that only one device gets control of the bus. These topics are covered in Sections 7.1 and 8.2 of the text.

Interrupt Arbitration

When more than one device needs to use the main processor bus, for example, to provide an interrupt number during an IACK cycle, it is necessary to have some method to make sure that only one device gets control.

A simple method is to "daisy-chain" an interrupt acknowledge signal from the CPU. Each device can choose to pass this signal on to the next device in the chain or block it and respond by putting its interrupt number on the bus. This scheme ensures that only one device will try to put its interrupt vector number on the data bus. It also prioritizes devices, giving different devices in the chain different priorities.



Exercise: Which device in the chain will have the highest priority?

Direct Memory Access (DMA)

We have already looked at polled i/o and interrupt-driven i/o. The third common way to service i/o devices is DMA. As the term implies, DMA transfers data directly from the i/o device to the memory system without going through the CPU.

DMA is used when the data rate is so high (approximately a hundred kBytes per second or more) that the CPU would not be able to copy data between the i/o device and memory sufficiently fast.

Typical types of devices that use DMA are disk drives and LAN interfaces (e.g. ethernet). If the DMA access does not use every bus cycle then DMA also allows the CPU to do other things while i/o is taking place.

Exercise: How many clock cycles does it take to respond to an interrupt, save one word on the stack, copy a byte from one memory location to another and return from the interrupt? What is the maximum transfer rate using this technique?

DMA operations are controlled by a DMA controller (DMAC) circuit, typically implemented as a single IC. The DMAC takes over the operation of the bus and in effect acts like a simple CPU. The DMAC must place an address on the data bus, assert the proper control signals (just like the CPU) and read or write data to/from the i/o device or memory on the data bus. During DMA access the CPU is dormant and the CPU's control lines are in high-impedance state.

In order to avoid bus conflicts due to the DMA controller and the CPU trying to use the bus at the same time, the DMAC uses the 68000's *bus arbitration* lines to obtain ownership of the bus when it needs to. Arbitration is also used to select which one of several DMA devices will get access to the bus.

The bus arbitration control lines are bus request $\overline{BR}$, bus grant $\overline{BG}$, and bus grant acknowledge $\overline{BGACK}$. A DMA device that wants to control the bus pulls $\overline{BR}$ low. The CPU drops $\overline{BG}$. If the DMA device sees $\overline{BG}$ it then pulls $\overline{BGACK}$ low to tell the CPU and other devices that it has control of the bus. When the CPU gives up control of the bus it places all of its output lines in high-impedance state to avoid bus conflicts.

As with vectored interrupts at the same level, when more than one DMAC can request control of the bus at the same time it is necessary to have some method to make sure that only one device gets control. This can be done by having a special circuit with multiple $\overline{BR}$ inputs and $\overline{BG}$ outputs. A simpler alternative is to "daisy-chain" the $\overline{BG}$ signal from the CPU.

A DMA device can transfer one or more bytes/words between i/o devices and/or memory in two different ways. The DMA controller can be interfaced to the i/o device either through dedicated control lines (implicit addressing) or through i/o ports in the memory address space (explicit addressing). In the case of explicit addressing the i/o device places it's data directly on the data bus and the DMAC causes that data to be written directly to memory in the same bus cycle. With explicit addressing the data is read from the i/o device (from a memory-mapped register) into the DMAC and then written back out in a subsequent bus cycle.

Exercise: Which method is faster? Which method is more flexible?

The DMA transfer can be in burst mode (the transfer is completed before giving up control of the bus) or in cycle-stealing mode where the DMAC gives up the bus after each read/write operation.

The actual details of DMA operation vary depending on the DMAC. The DMAC has internal registers which are set up with pointers to the memory location(s) to be accessed and the number of bytes to be transferred. When the registers have been set up, a control word is written to start the DMA operation.

The DMA operation is often started and/or ended by an ISR. For example, a disk controller could be set up to issue an interrupt when its read head reached the desired location on the disk and it was ready to read data. The ISR would then set up the DMAC for a transfer and to generate another interrupt at the end of the transfer.

For example, a typical use of a DMAC would involve loading the following values into control registers in the DMAC:

- the start address in memory for the transfer
- the number of values (bytes, words, longwords) to transfer
- control bits to select the direction of transfer, size of each data value, whether to interrupt when done, etc
- the interrupt vector number to use when the transfer is done (if used)

The interface between the DMAC and the i/o device involves a number of handshaking lines and must usually be custom-designed for each device. Many DMACs support multiple DMA devices (channels) on the same chip.

The bus arbitration technique can be used to implement multiprocessor systems with shared memory.

Comparison of I/O Strategies

The three I/O strategies we have discussed (polling, interrupt-driven and DMA) vary in the amount of hardware required, transfer rate, latency, number of CPU cycles consumed per transfer, design complexity, and cost. You should understand the tradeoffs and be able to justify the selection of an I/O strategy given the requirements of a particular I/O device.

Exercise: Rank the three I/O strategies according to the criteria outlined above. Which technique would you use for each of the following i/o applications: a PC keyboard? a microwave oven keypad? a modem serial port? a hard disk?