A

Seminar report

On

Direct Memory Access

Submitted in partial fulfillment of the requirement for the award of degree
Of
Computer Science

SUBMITTED TO: **SUBMITTED BY:**

www.studymafia.org
www.studymafia.org

Preface

I have made this report file on the topic **Direct Memory Access**; I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

My efforts and wholehearted co-corporation of each and everyone has ended on a successful note. I express my sincere gratitude to ……..who assisting me throughout the preparation of this topic. I thank him for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

Introduction

In many I/O interfacing applications and certainly in data acquisition systems, it is often necessary to transfer data to or from an interface at data rates higher than those possible using simple programmed I/O loops. Microprocessor controlled data transfers within the PC (using the IN (port) and OUT (port) instructions) require a significant amount of CPU time and are performed at a significantly reduced data rate. Further to this, the CPU cannot perform any other processing during program controlled I/O operations.

While the use of interrupts might allow the CPU to perform some concurrent tasks, certain applications exist where the amount of data to be transferred and the data rate required is too high. Two such applications are as follows:

• Transferring screen information to the 'video card adapter' on board memory
• Transferring data from a remote I/O device (data acquisition board) to the PC's memory

Direct memory access (DMA) facilitates the maximum data transfer rate and microprocessor concurrence. Unlike programmed or interrupt controlled I/O, where data is transferred via the microprocessor and its internal registers, DMA (as its name implies) transfers data directly between an I/O device and memory (memory to memory DMA transfers are also possible). Whichever CPU is being used, it must have a DMA feature to determine when DMA is required, so that it can relinquish control of the address and data buses, as well as the control lines required to read and write to memory. In addition, the CPU must inform the I/O device that requires the DMA data transfer when it again requires control of the address and data buses and I/O control lines. Further to this, a separate DMA controller is required to actually perform the DMA I/O operations.

What is DMA?

The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer .this transfer technique is called Direct memory access(DMA).During DMA transfer the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the IO device and memory.

The process of DMA is handled by a DMA chip and it can be considered as a primitive secondary processor whose job is to relieve the main processor of much of the burden of memory transfers between memory and I/O Devices. The DMA chip was originally an Intel 8237 device. There are three possible ways that the 8237 can transfer data - I/O to Memory, Memory to I/O and Memory to Memory.

History

An IBM (International Business Machines) compatible computer system includes two Intel 8237 compatible DMA controllers. A complete description of the 8237 DMA controller is found in the 8237A High Performance Programmable DMA Controller datasheet published by Intel Corporation, and hereby incorporated by reference.

Why we need DMA?

The assumption about I/O devices is that it's really, really slow, compared to the speed of the CPU. I/O devices include keyboard, mouse, printers, etc. While some devices may need quick response, usually when downloading or uploading data, I/O devices are general considered slow.

Rather than the CPU "polling" or asking the I/O device if it needs some action taken on its behalf, the I/O device interrupts the CPU by sending an interrupt signal (such as a positive edge). The CPU then temporarily stops what it is doing, and services the I/O device.

The code to do this is called the *interrupt handler*. The handler must already be loaded in memory before the interrupt has occurred .Occasionally, an I/O device needs to read from memory or write to memory. This can be very slow, as we shall see using Diagram of I/O devices.

Now imagine that the I/O device wants to store a lot of data to memory. Perhaps you have just taken some pictures on a digital video camera, and want to store this information.

One way to do this is to alert the CPU that the I/O device wants to store data to memory. These are the steps that might be taken.

- Ø CPU sends signal to I/O device to put data on data bus.
- Ø I/O device places data on data bus, and signals it has done so.
- Ø CPU reads in the data into a register, and signals the I/O device that it has finished reading the data.
- Ø CPU signals memory that it is ready to write data to memory.
- Ø CPU places address and data on data bus.
- Ø Memory writes data from data to some memory location.
- Ø Memory signals CPU it has written data.

Thus the data flows from the I/O device to the CPU, and from the CPU to memory.

After all, how can memory who is sending signals to it? How does it know that the CPU sends signals, instead of the device itself?

If the I/O device generates all the signals that the CPU does, then the I/O device can bypass the step where the data transfers to the CPU, thus, potentially doubling the speed.

What is actually DMA?

Direct memory access (DMA) is a system whereby samples are automatically stored in system memory while the processor does something else. The process of transferring data via DMA is given below:

1. When data is ready for transfer, the board directs the system DMA controller to put it into in system memory as soon as possible.
2. As soon as the CPU is able (which is usually very quickly), it stops interacting with the data acquisition hardware and the DMA controller moves the data directly into memory.
3. The DMA controller gets ready for the next sample by pointing to the next open memory location.
4. The previous steps are repeated indefinitely, with data going to each open memory location in a continuously circulating buffer. No interaction between the CPU and the board is needed.

- **Problems with DMA**

There are some potential problems. Before we introduced I/O devices, the CPU was in charge of the bus. It was in control of when the bus was being accessed.

With I/O devices, both the CPU and the I/O device may be contending to use the address and data bus, so there needs to be some way of deciding who gets the bus at which time.

Furthermore, if there are more than one I/O devices trying to access the bus, we may have to decide which I/O device gets control.

One potential solution to the problem is to have a hardware device that is a bus arbitrator, which decides which device gets the bus, and helps coordinate who sends what signals to the memory.

The CPU may be placed in an idle state in variety of ways. One common method extensively used in microprocessor is to disable the buses through special control signals. Fig shows two control signals in the CPU that facilitate the DMA transfer. The bus request (BR) input is used by the DMA controller to request the CPU to relinquish control of the buses. When this input is active, The CPU terminates the execution of the current instruction and places the address bus, data bus, and read & writes lines into a high-impedance state. The high impedance state behaves like an open circuit, which means that the output is disconnected and does not have logic significance. The CPU activates the bus grant (BG) output to inform the external DMA that the buses are in the high impedance state the DMA that originated the bus request can now take control of the buses to conduct to memory transfer without processor intervention. When the DMA terminates the transfer, It disables the bus request line. The CPU disables the bus grant, take control of the buses, and returns to its normal operation.

When the DMA takes control of the bus system, it communicates directly with the memory. The transfer can be made in several ways. In DMA burst transfer, a block sequence consisting of a number of memory words is transferred in a continuous burst while the DMA controller is master of the memory buses. This mode of transfer is needed for fast device such as magnetic disks, where data transmission can not be stopped or slowed down until and entire block is transferred. And alternative technique called stealing allows the DMA controller to transfer one data word at a time, after which it must return control of the buses to the CPU. The CPU merely delays its operation for one memory cycle to allow the direct memory IO transfer to “steal” one memory cycle.

Ø When starting an operation, the CPU informs the DMA module about:

- What operation (read or write);

- The address of the I/O device involved;

- The starting location in memory where information has to be

Stored to or read from;

- The number of words to be transferred.

DMA Channels

The 8237 DMA controller provided by IBM (International Business Machines) is a peripheral interface circuit for allowing peripheral devices to directly transfer data to or from main memory. It includes four independent channels and may be expanded to any number of channels by cascading additional controller chips. In the IBM architecture, two DMA controllers are used. One DMA controller is used for byte transfers, and the second DMA controller is user for word (16-bit) transfers. All four channels (designated 0, 1, 2 and 3) of the byte-wide DMA controller are dedicated to performing byte DMA operations. Of the four channels (designated 4, 5, 6 and 7) of the word-wide DMA controller channels 5, 6 and 7 are dedicated or word DMA operations. Channel 4 is used for cascading the two controllers together and, therefore, is not available for normal DMA.

DMA used for

0 8-bit transfer

1 8-bit transfer
2 Floppy disk controller
3 8-bit transfer
4 Cascade from 0-3
5 16-bit transfer
6 16-bit transfer
7 16-bit transfer

Different Modes of DMA operation

There are three different modes of DMA operations:

1. Continuous DMA

2. Cycle stealing

3. interleaved DMA

1. Continuous DMA

It is also referred to as DMA transfer. In this scheme a sequence of arbitrary length of data is transferred in a single continuous burst during which the DMA controller is the master of the system bus. The controller halts the CPU at the end of current instruction while CPU's address and data bus are put in floating state by means of tristate buffers.

The DMA controller then subsequently calls up the memory addresses at the specified clock rate and exchanges data with memory via the data bus as instructed. The data source/destination specified by the IOAR is modified after each transfer. also the data counter in DCR gets decremented. The block transfer terminates while DCR becomes zero. At this point controller deactivates HALT signal and relinquishes control of the system bus. It may also send an interrupt signal to the CPU.

This type of processor halted DMA can support highest IO data transmission rate. So it is suitable to handle IO devices needing high data transfer rate such as disk drive where data transmission cannot be stopped or slowed down without loss of data.

2. Cycle stealing

The underlying concept of such techniques is to enable DMA controller to steal a few cycles while it controls system bus to exchange data with memory. Subsequently the control is return back to the CPU. This reduces DMA controller's interference In CPU's activities.

Halt and restart is the simple technique derived out of the previous scheme of DMA block transfer with the modification that the CPU

Is halted for a small duration while a few data words are exchanged by the DMA controller. The CPU is allowed to return back to its ON state after elapse of the specified halting duration. By varying the inactive duration of CPU, a trade off between DMA rate and CPU's processing Requirement of a computing environment.

CPU can be equipped with special cycle stretching inputs which suspend processor's clock for a few microseconds upon receiving DMA request signal. One or more data word can then be exchanged with memory under DMA control without actually halting the CPU or waiting for the completion of current instruction. Clock suspension is limited to small duration such that internal data is not lost in CPU employing dynamic circuits.

The cycle stretching input for a few microprocessors are: WAIT for Z80, HOLD for 8085.DMA for 6809, TSC for 6800.

Transparent DMA is the technique which employs cycle stealing in true sense. Usually there exists instructions in all processors while it works on data stored in its internal registers and so does not utilize system bus. Such cycles can be truly stolen by the DMA controller while taking note of the signals generated during execution of such instructions by the CPU.

3. Interleaved (Multiplexed) DMA

If main memory is, roughly speaking, twice as fast as the processor, then the processor and DMA accesses of memory can be interleaved on alternate half cycle of the clock as

shown in fig. the DMA accesses the memory during first half of the clock cycle while CPU accesses during second half. A scheme can be easily implemented with Motorola 68000 family processor which executes memory read/write during second half of clock. To implement the scheme it is necessary to buffer the address and data buses so that processor can be

Disconnected from the memory bus during first half of the CPU clock. No aspect of the main program execution by the CPU or its timing gets affected. Hence this is a true transparent DMA operation achieved with the additional cost of high speed memory employed in the system.

The block diagram of a system equipped with multiple DMA controllers is shown in fig. Any DMA controller can raise the bus request line. The bus controller, CPU in the present case, resolve the bus arbitration. The units, as noted in fig can be DMA controller or coprocessors, some of which can also be act as potential master of the bus. The bus controller unit in CPU on getting a DMA request responds by getting DMA ack after getting control of system bus at DMA breakpoint. The requesting the DMA ack signal now becomes bus master and executes the DMA operations.DMA chaining is the scheme which enables the DMA controller to carry out a sequence of DMA block transfers without any reference to CPU. At the end of the transfer of the current data block, the DMA controller registers are initialized to the desired value and the next DMA block transfer operation is initiated if the DMA chaining flag is set in the control register IOCSR. Additional circuit in the DMA controller can recognize the chaining operation and realize the intended behavior.

The flowchart shown in fig summarizes the sequence of actions associated with DMA block transfer. At the end of IO data transfer, the DMA controller can draw CPU'S attention by generating the IO interrupt signal. In response, the CPU can check the status of execution and take appropriate action as may be executing the interrupt service routine.

DMA Operational Modes and Settings

The 8237 DMA can be operated in several modes. The main ones are:

Single

A single byte (or word) is transferred. The DMA must release and re-acquire the bus for each additional byte. This is commonly-used by devices that cannot transfer the entire block of data immediately. The peripheral will request the DMA each time it is ready for another transfer.

The standard PC-compatible floppy disk controller (NEC 765) only has a one-byte buffer, so it uses this mode.

· **Block/Demand**

Once the DMA acquires the system bus, an entire block of data is transferred, up to a maximum of 64K. If the peripheral needs additional time, it can assert the READY signal to suspend the transfer briefly. READY should not be used excessively, and for slow peripheral transfers, the Single Transfer Mode should be used instead.

The difference between Block and Demand is that once a Block transfer is started, it runs until the transfer count reaches zero. DRQ only needs to be asserted until -DACK is asserted. Demand Mode will transfer one more bytes until DRQ is de-asserted, at which point the DMA suspends the transfer and releases the bus back to the CPU. When DRQ is asserted later, the transfer resumes where it was suspended.

Older hard disk controllers used Demand Mode until CPU speeds increased to the point that it was more efficient to transfer the data using the CPU, particularly if the memory locations used in the transfer were above the 16Meg mark.

· **Cascade**

This mechanism allows a DMA channel to request the bus, but then the attached peripheral device is responsible for placing the addressing information on the bus instead of the DMA. This is also used to implement a technique known as ``Bus Mastering''.

When a DMA channel in Cascade Mode receives control of the bus, the DMA does not place addresses and I/O control signals on the bus like the DMA normally does when it is active. Instead, the DMA only asserts the -DACK signal for the active DMA channel.

At this point it is up to the peripheral connected to that DMA channel to provide address and bus control signals. The peripheral has complete control over the system bus, and can do reads and/or writes to any address below 16Meg. When the peripheral is finished with the bus, it de-asserts the DRQ line, and the DMA controller can then return control to the CPU or to some other DMA channel.

Cascade Mode can be used to chain multiple DMA controllers together, and this is exactly what DMA Channel 4 is used for in the PC architecture. When a peripheral requests the bus on DMA channels 0, 1, 2 or 3, the slave DMA controller asserts HLDREQ, but this wire is actually connected to DRQ4 on the primary DMA controller instead of to the CPU. The primary DMA controller, thinking it has work to do on Channel 4, requests the bus from the CPU using HLDREQ signal. Once the CPU grants the bus to the primary DMA controller, -DACK4 is asserted, and that wire is actually connected to the HLDA signal on the slave DMA controller. The slave DMA controller then transfers data for the DMA channel that requested it (0, 1, 2 or 3), or the slave DMA may grant the bus to a peripheral that wants to perform its own bus-mastering, such as a SCSI controller.

Because of this wiring arrangement, only DMA channels 0, 1, 2, 3, 5, 6 and 7 are usable with peripherals on PC/AT systems.

DMA channel 0 was reserved for refresh operations in early IBM PC computers, but is generally available for use by peripherals in modern systems.

When a peripheral is performing Bus Mastering, it is important that the peripheral transmit data to or from memory constantly while it holds the system bus. If the peripheral cannot do this, it must release the bus frequently so that the system can perform refresh operations on main memory.

· **Auto initialize**

This mode causes the DMA to perform Byte, Block or Demand transfers, but when the DMA transfer counter reaches zero; the counter and address are set back to where they were when the DMA channel was originally programmed. This means that as long as the peripheral requests transfers, they will be granted. It is up to the CPU to move new data into the fixed buffer ahead of where the DMA is about to transfer it when doing output operations, and to read new data out of the buffer behind where the DMA is writing when doing input operations.

This technique is frequently used on audio devices that have small or no hardware ``sample'' buffers. There is additional CPU overhead to manage this ``circular'' buffer, but in some cases this may be the only way to eliminate the latency that occurs when the DMA counter reaches zero and the DMA stops transfers until it is reprogrammed.

help you get started in understanding DMA and Windows drivers, this section describes key concepts in DMA device architecture and introduces the terminology that will be used in this paper.

How DMA Interface?

Figure shows a logical pin out and internal registers of 8237, it also shows the interface with the 8085 using a 3-to 8 decoder. The 8237 has four independent channels, CH0 to CH3, Internally, two 16-bit registers are associated with each channel: One is used to load a starting address of the byte to be copied and the second is used to load a count of the number of bytes to be copied. Figure shows eight such registers that can be accessed by the MPU. The addresses of these registers are determined by four address lines, A3 to A0, and the chip select (CS) signal. Address 0000 on lines A3-A0 selects CH0 Memory Address Register (MAR) and address 0001 selects the next register, CH0 Count. Similarly, all the remaining registers are selected in sequential order. The last eight registers are used to write commands or read status as shown. In Figure, the MPU accesses the DMA controller by asserting the signal Y0 of the decoder. 'Therefore, the addresses of these internal registers range from 00 to 0FH as follows:

DMA Signals

In figure shows signals are divided into two groups: (i) one group of signals shown on the left of the 8237 is used for interfacing with the MPU, (ii) the second group shown on the right-hand side of the 8237 is for communicating with peripherals. Some of these signals are bidirectional and their functions are determined by the DMA mode of operation (I/O or processor). The signals that are necessary to understand the DMA operation are explained as follows:

- **DREQ0-DREQ3-DMA Request:** These are four independent, asynchronous input Signals to the DMA channels through peripherals such as floppy disks and the hard disk. To obtain DMA service, a request is generated by activating the DRBQ line of the channel.
- **DACK0-DACK3-DMA Acknowledge:** These are output lines to inform the individual peripheral that a DMA is granted, DREQ and DACK are equivalent to handshake signals in I/O devices.

- **AEN and ADSTB—Additives Enable and Address Length:** These are active high output signals that are used to latch a high-order address byte to generate a 16-bit address.
- **MEMR and MEWR—Memory Read and Memory Write:** These are output signals used during the DMA cycle to write and read From memory.
- **A_3-A0 and A7-A4—Address:** A3-A0 are bidirectional address lines. These are used as inputs In access control registers as shown. During the DMA cycle, these lines are used as output lines to generate a low-order address than is combined with the remaining address lines A_7-A4.
- **HRQ and HLDA—Hold Request and Hold Acknowledge:** HRQ is an output signal used to request the MPU control of the system bus, After receiving the HRQ The MPU completes the bus cycle in process and issues the HLDA signal.

Memory DMA (MemDMA)

The Memory DMA (MemDMA) controller provides memory-to-memory DMA transfers among the ADSP-BF535 processor memory spaces. These memory spaces include the Peripheral Component Interconnect (PCI) address spaces, L1, L2, external synchronous and asynchronous memories.

The MemDMA controller consists of two channels—one for the source, which is used to read from memory, and one for the destination, which is used to write to memory. Both channels share a 16-entry, 32-bit FIFO. The source DMA channel fills the FIFO; the destination DMA channel

Empties it. The FIFO depth significantly improves throughput on block transfers between internal and external memory. The FIFO supports 8-, 16-, and 32-bit transfers. However,

8- and 16-bit transfers use only part of the 32-bit data bus for each transfer; therefore, the throughput for these transfer sizes is less than for full, 32-bit DMA operations.

Two separate linked lists of descriptor blocks are required—one for the source DMA channel and one for the destination DMA channel. The separation of control allows an off-chip host processor to manage one list through the PCI port bus and the core processor to manage the other list. Because the source and destination DMA channels share a single FIFO buffer, the descriptor blocks must be configured to have the same transfer count and data size.

It is preferable to activate interrupts on only one channel. This eliminates ambiguity when trying to identify the channel (either source or destination) that requested the interrupt.

Remote Direct Memory Access

Remote Direct Memory Access (**RDMA**) is a concept whereby two or more Computers communicate via Direct Memory Access directly from the main memory of one system to the main memory of another. As there is no CPU, cache, or context switching overhead needed to perform the transfer, and transfers can continue in parallel with other system operations, this is particularly useful in applications where high throughput, low latency networking is needed such as in massively parallel Linux clusters. The most common RDMA implementation is over Infiniband Although RDMA over Infiniband is technologically superior to most alternatives; it faces an uncertain commercial future.

This also has the advantage that software-based RDMA emulation will be possible, allowing interoperation between systems with dedicated RDMA hardware and those without. One example of this might be the use of a server with a hardware-equipped RDMA to serve a large number of clients with software-emulated RDMA implementations.

Hardware and software requirement

In a computer with DMA Hardware the software is relieved from the task of performing the IO operations, However, It is the software's responsibility to tell the DMA Hardware What types of an I/O Operation is required and to signal the DMA Hardware when it should begin performing an IO operation. Once this is done by the CPU software, the DMA hardware performs the data transfer without the help of the CPU. After all the data bytes are transferred, the DMA hardware reports the 'completed' information to the CPU. Figure illustrates the protocol between the DMA hardware and software. The DMA parameters should basically include the following information:

1. The I/O device to be involved in the data transfer.
2. The indirection of data transfer –whether an input operation or an output operation.
3. The number of bytes to be transferred between the memory and the device.

4. The start address of the memory area from where the data has to be read (for an output operation) or in which the data has to be stored (for an input operation).

   The method of supplying the DMA parameters to the DMA hardware varies with the architecture of the computer. In a large system the DMA parameters are not directly issued to the DMA hardware. Instead the parameters are indirectly transferred by using a portion of the memory as shown in fig. The software stores the parameters in the main memory from where the DMA hardware fetches them. On the other hand in micro- computer it is the software's responsibility to issue all DMA parameters to the DMA hardware by means of OUT instructions this is shown in figure.

The DMA hardware store the DMA parameters in different registers .When data bytes are transferred, the DMA hardware continuously updates certain parameters. The 'byte count' parameter is decremented by one for every byte transferred whereas the 'memory address' parameter is incremented by one for every byte transferred .Once 'byte count' reaches zero, the DMA hardware terminates the data transfer.

When designing a DMA may, you should have

- Ø Address register
- Ø Byte Control register
- Ø Status register
- Ø DMA control register

Advantages:

1. DMA is fast because a dedicated piece of hardware transfers data from one computer location to another and only one or two bus read/write cycles are required per piece of data transferred.
2. DMA is usually required to achieve maximum data transfer speed, and thus is useful for high speed data acquisition devices.
3. DMA also minimizes latency in servicing a data acquisition device because the dedicated hardware responds more quickly than interrupts, and transfer time is short.

4. Minimizing latency reduces the amount of temporary storage (memory) required on an I/O device.
5. DMA also off-loads the processor, which means the processor does not have to execute any instructions to transfer data. Therefore, the processor is not used for handling the data transfer activity and is available for other processing activity. Also, in systems where the processor primarily operates out of its cache, data transfer is actually occurring in parallel, thus increasing overall system utilization.

Disadvantages

1. Cost of DMA hardware
2. DMA is useful only for DATA commands. All non-data commands have to be executed by CPU.
3. Data has to be stored in continuous locations in memory.
4. CPU's intervention is required for initializing DMA logic for every continuous data block transfer. In other words, DATA CHAINING is not possible.
5. All error conditions have to be read and analyzed by CPU.

6. These limitations have lead to the next mode of the INPUT/OUTPUT i.e. INPUT/OUTPUT channels.

Application

1. DMA has been a built-in feature of PC architecture since the introduction of the original IBM PC.
2. PC-based DMA was used for floppy disk I/O in the original PC and for hard disk I/O in later versions.
3. PC-based DMA technology, along with high-speed bus technology, is driven by data storage, communications, and graphics needs–all of which require the highest rates of data transfer between system memory and I/O devices.

Conclusion

National Instruments uses DMA hardware and software technology to achieve high throughput rates as well as to increase system utilization. These achievements are accomplished by using a background mechanism of data transfer that minimizes CPU usage. Data acquisition users are highly aware of the advantages of background data acquisition, and DMA solutions have been very popular. Lab Driver double-buffered data acquisition features are popular among users for experiments involving large amounts of data that are reduced as the data is acquired. Users appreciate the flexibility of buffering up the data and processing it in blocks, rather than having to acquire and stop for processing, or having to read individual points.

Although DMA increases system utilization and achieves high throughput, remember that interrupts do offer some advantages over DMA. A DMA controller can only transfer data from one location to another. However, it is often desirable to process data on the fly–for example, searching for a trigger condition in the data before saving it into memory, or implementing a high-speed process control loop. Interrupt-driven

acquisition is required in these applications because computation on individual points must be performed. In addition, there are usually fewer DMA channels available than interrupt channels. Lab Driver uses DMA or interrupts interchangeably so that channel availability is transparent to the application program. Another advantage is that interrupt routines can transfer data from a set of locations into different data buffers, whereas one DMA channel is required for each location and buffer. The flexibility of interrupts must be weighed against higher CPU utilization, improved service latency, and, in some cases, higher throughput when using DMA.

DMA can still be used when incoming data is slow, although the improvements in CPU utilization and latency are not significant at slow speeds. In fact, at slow speeds waiting for a block of a buffer to fill is not