

# Theory of Operation

## Hardware Analysis

The ESD 8086 board was modified to accommodate fly-by direct memory access transfers. The two main additions to the board are a Direct Memory Access Controller (DMAC, part 82C37A), and two 256 x 8 First In First Out (FIFO, part IDT7200) devices.

With the addition of a DMA controller, the system has become a multi-master system. In a multi-master system, there is more than one device that is capable of driving the system-wide address bus, data bus, and control signals. The first device is the 8086 processor already present on the ESD board. Unfortunately, when moving large amounts of data between two devices (such as moving an image from memory to a laser marking engine), the processor is extremely inefficient. The processor must first perform a memory read, taking a total of four clock cycles, then perform an I/O write, taking another four clock cycles for each 16 bit transfer.

Since the processor is significantly slower at moving data between two devices in the system than the memory is capable of supplying data, this is clearly a poor use of the system bus for this application. DMA Fly-By transfers are one good solution to this problem. A DMA Controller, in a standard operating mode can move data directly between two devices in three clock cycles (as opposed to the 8 clock cycles required by the processor). The only stipulation is that the two devices must consist of one memory device and one I/O device for a fly-by transfer.

To perform a fly-by transfer, the DMA controller obtains control of the system address bus and control lines from the processor. The appropriate memory device (as selected by decoding the address) views the bus cycle as it would any read cycle, and drives the data found at that address on to the system wide data bus.

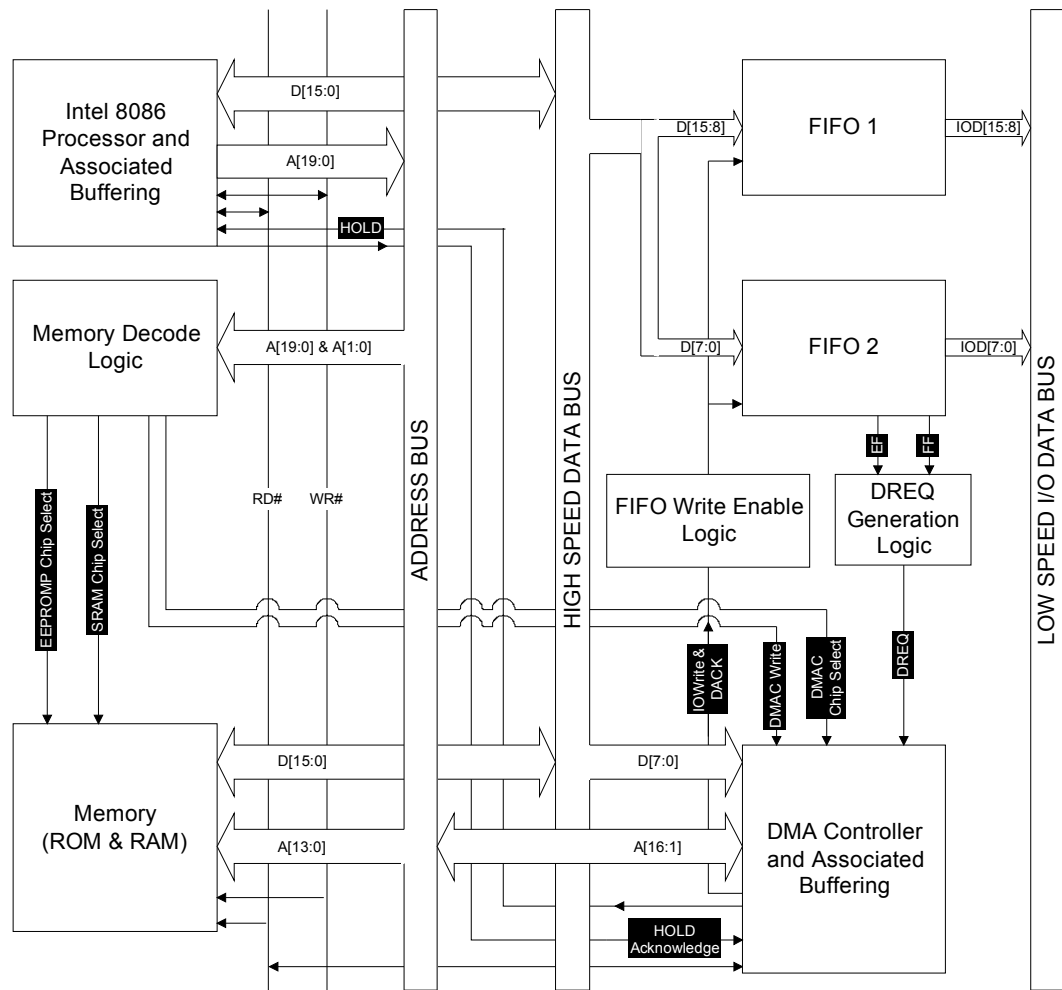
The DMA controller drives separate read and write lines that are connected to the I/O device, so that while it is driving the read line connected to the memory device, it can also drive the write line connected to the I/O device. Typically, an I/O device does not need to do much address decoding when being written to for large data transfers, so the system must be designed so that the I/O device ignores the address lines, and is selected through different means during a DMA transfer.

Now that the inefficiency of the processor performing large move operations between memory and I/O has been addressed, the next major bottleneck on the system must be examined. Usually, I/O devices are quite slow in relation to the rest of a computer system (often by at least an order of magnitude). Therefore, if the DMA transfer is being restricted to the speed of a typical I/O device, there is not much of a performance gain found when using DMA transfers. To find a solution to this problem, the I/O device must be physically separated from the system wide data bus. This is accomplished through the use of FIFOs.

FIFOs are usually extremely fast devices (at least as fast as the memory devices in the system). They consist of some number of data bus connections on an A and B side, and some number of internal registers to temporarily store data. In the case of the system under examination, the FIFOs in use have 256 internal registers, and are 8 bits wide. By using FIFOs, a fast memory device can burst data into the FIFOs quickly, and be emptied slowly by the I/O device. While the FIFOs are being emptied, the processor can once again regain control of the system busses, and is therefore interrupted as little as possible. Once a solution of this type has been implemented, the system busses and memory devices are being used as efficiently as possible to move data to slow I/O devices.

The method that is used to control DMA transfers to the FIFOs is extremely important. When the FIFOs get full, if any more writes are done to them before there is space made inside, that data will be lost. This results in distorted images, or broken data being read from the FIFOs on the I/O side. Therefore, choosing

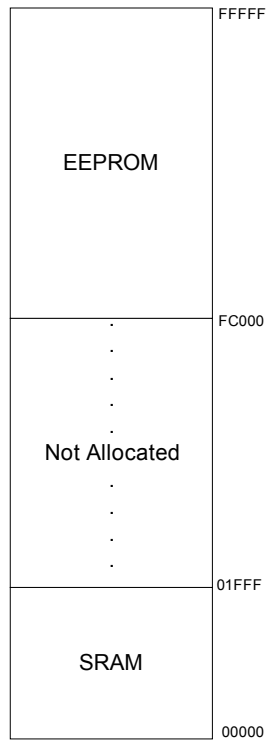
how to stop transfers to the FIFOs is critical. Also of great importance is the method used to choose when to begin writing. If a new transfer is started as soon as space is made available inside the FIFOs, there will only be one or two spaces open, and the DMA transfer will be extremely short. If the system is used in this manner, the DMAC will be continuously requesting the system bus from the processor and releasing it, which is an inefficient use of the system, as shown in the Results section of this report and in figure 1-5.



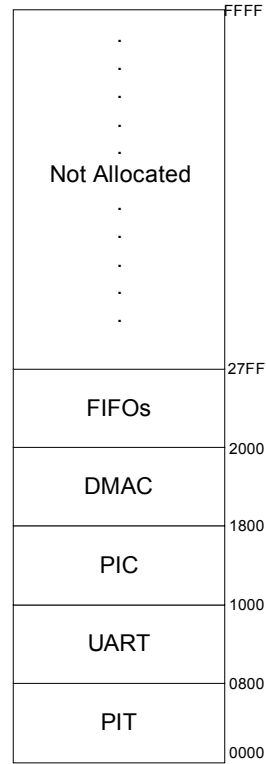
**Figure 1-1 – System Block Diagram including FIFOs and DMAC**

Figure 1-1 shows the block diagram for the changes implemented on the ESD 86 board. As discussed previously, the FIFOs separate the system wide data bus

from the low speed I/O bus, and are controlled through a separate means of decode and write enable from the rest of the system.



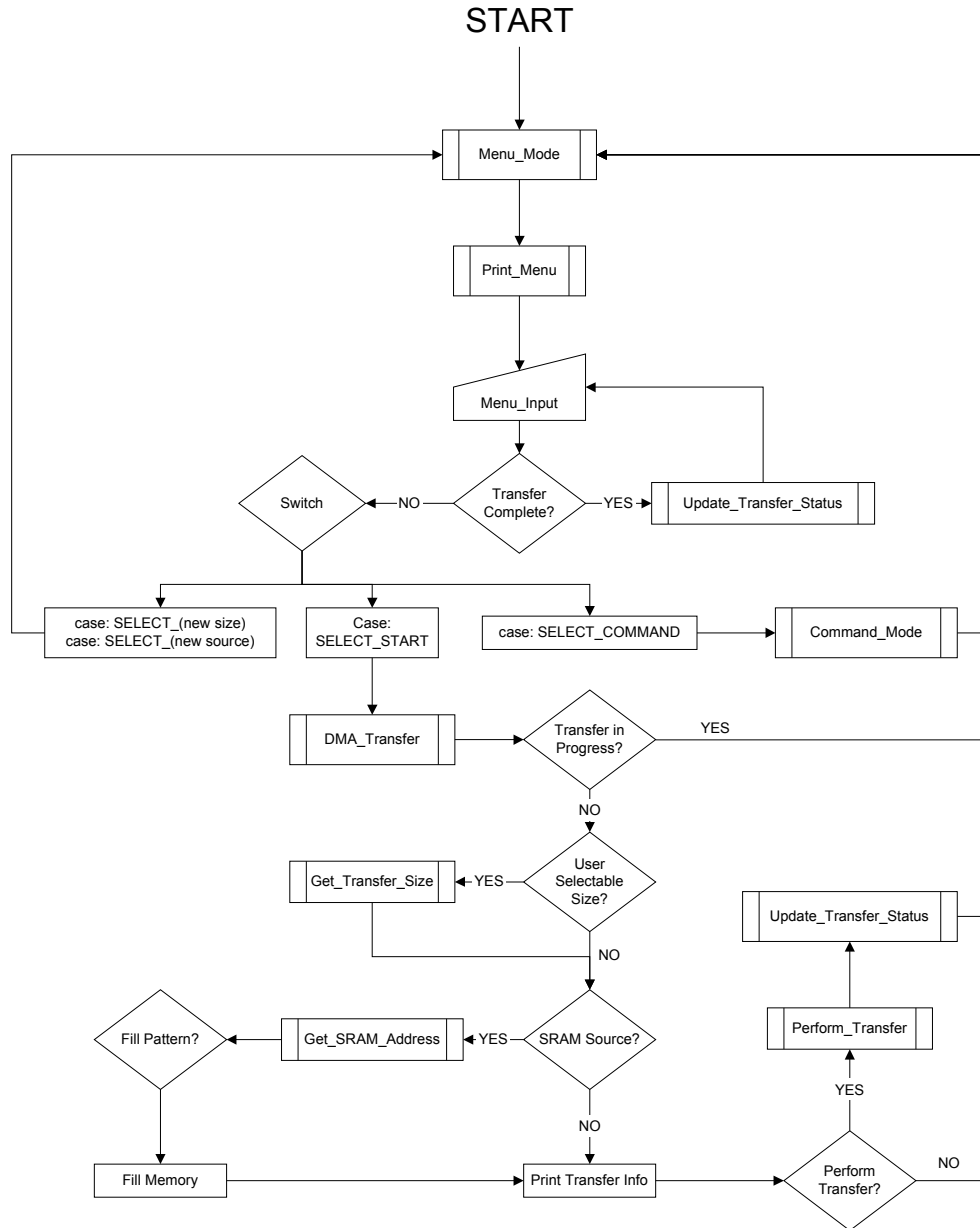
**Figure 1-2 – Memory Map**



**Figure 1-3 – I/O Map**

Figure 1-2 shows the current memory map for the ESD 86 board. This has not been modified with the addition of the DMAC and FIFOs. Figure 1-3 is the current I/O map for the ESD 86 board. The only changes in the I/O map are the addition of the DMAC and FIFOs at address 1800 and 2000 respectively.

## Software Analysis



**Figure 1-4 – Software Flow Chart**

The flow chart shown in figure 1-4 contains the basic flow and logic used in the DMA transfer menu that controls DMA transfers via a serial port connection to a

VT100 terminal (or terminal emulator). All of the C code that is loaded in to the on board EEPROMs is included in the appendix.

The system uses an intuitive, menu driven interface to get from the user all options and values needed to perform a DMA transfer. After getting confirmation from the user, the transfer is programmed in to the DMA Controller. When the transfer completes, an interrupt service routine (ISR) is called. In order to avoid performing slow I/O operations with the serial port in the ISR, a memory based flag is set that is checked while the system is looping, waiting for user input. This causes the on-screen status of the DMA transfer to be updated, showing that the transfer has completed.

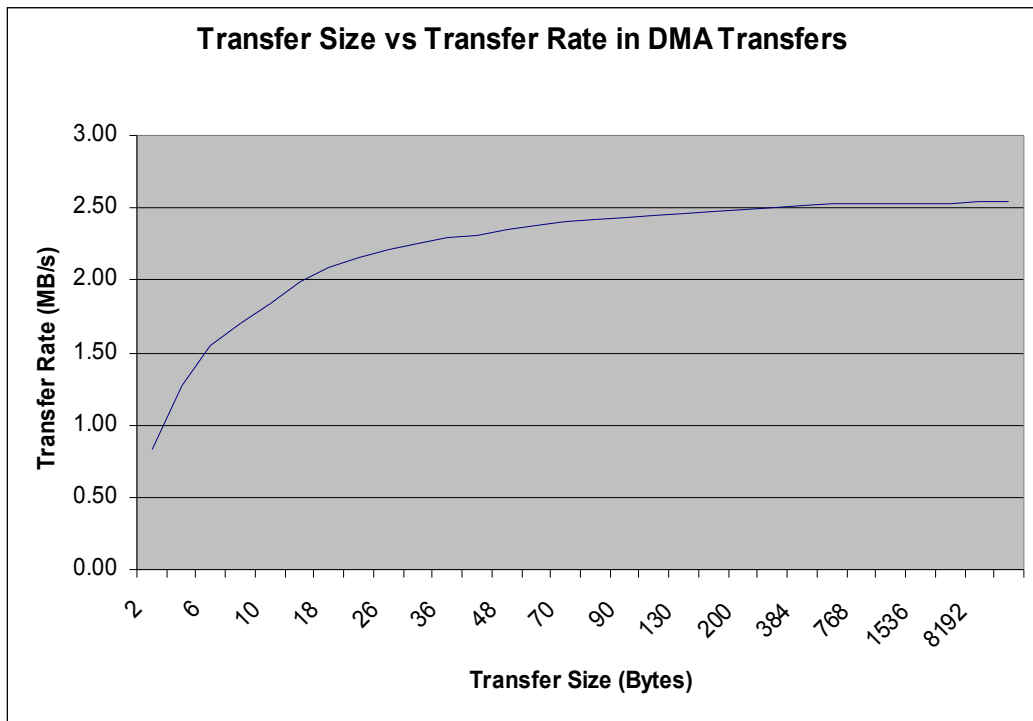


Figure 1-5 – Transfer Rate vs Transfer Size

The final throughput of this system has been evaluated, and according to the results shown in Figure 1-5, the maximum throughput is just over 2.5 MB/s, and the maximum speed has effectively been reached by transfer sizes of around 256 Bytes.