# Storage & DMA Integration - 8086 Microprocessor Project

# Giovanny Garcia

This document outlines the implementation of the 8272 floppy disk controller, DMA-based high-speed data transfer using the 8237 controller, USB transfers via DMA, and system integration for the 8086 microprocessor project.

## 1. 8272 Floppy Disk Controller

### Components used:

* 8272 Floppy Disk Controller (FDC)
* Floppy disk drive interface
* DMA integration for high-speed transfers

### Block Diagram:

┌─────────────┐  
| |  
| RAM | ┌─────────────┐  
| Buffer | | 8272 |  
| @0xC0900 | | FDC |  
| | | @0xC0900 |  
└─────────────┘ └─────────────┘  
 ^ |  
 | |  
 | ▼  
┌──────────────┐ ┌──────────────┐  
| | | |  
| 8086 CPU |<──────────────| FLOPPY DISK |  
| | | DRIVE |  
└──────────────┘ └──────────────┘  
 |  
 |  
┌──────────────┐  
| 8237 |  
| DMA |  
| CONTROLLER |  
└──────────────┘

### Description:

#### 8272 Floppy Disk Controller

* Manages read/write operations to floppy disks (360KB/720KB/1.44MB formats).
* Handles sector formatting, track positioning, and data transfer.
* Supports multiple drives (up to 4 drives).
* Uses DMA for high-speed data transfer to minimize CPU overhead.
* Provides status monitoring for drive ready, write protect, and track positioning.

### Address Mapping:

| Register | Address |
| --- | --- |
| Status Register | 0xC0900 |
| Command Register | 0xC0901 |
| Data Register | 0xC0902 |
| Control Register | 0xC0903 |

### PSEUDOCODE

FUNCTION Initialize8272FDC()  
BEGIN  
 // Reset the 8272 controller  
 WritePort(0xC0903, 0x00) // Reset control register  
 Wait(10 milliseconds)  
 WritePort(0xC0903, 0x0C) // Enable controller  
   
 // Wait for controller ready  
 WAIT\_FDC\_READY:  
 status = ReadPort(0xC0900)  
 IF (status AND 0x80 == 0) THEN  
 GOTO WAIT\_FDC\_READY  
 EndIF  
   
 // Configure drive parameters  
 WritePort(0xC0901, 0x03) // Specify command  
 WritePort(0xC0902, 0xCF) // Step rate=3ms, head unload=240ms  
 WritePort(0xC0902, 0x02) // Head load=4ms, DMA mode  
   
 // Recalibrate drive 0  
 WritePort(0xC0901, 0x07) // Recalibrate command  
 WritePort(0xC0902, 0x00) // Drive 0  
   
 // Wait for recalibration complete  
 WAIT\_RECALIBRATE:  
 status = ReadPort(0xC0900)  
 IF (status AND 0x20 == 0) THEN  
 GOTO WAIT\_RECALIBRATE  
 EndIF  
   
 SHOW "8272 FDC initialized successfully"  
 RETURN SUCCESS  
END  
  
FUNCTION ReadSector(drive, track, head, sector, buffer)  
BEGIN  
 // Setup DMA for read operation  
 CallFunction ConfigureDMAForRead(buffer, 512)  
   
 // Send read command to 8272  
 WritePort(0xC0901, 0x46) // Read data command  
 WritePort(0xC0902, (head << 2) | drive) // Head and drive  
 WritePort(0xC0902, track) // Track number  
 WritePort(0xC0902, head) // Head number  
 WritePort(0xC0902, sector) // Sector number  
 WritePort(0xC0902, 0x02) // 512 bytes per sector  
 WritePort(0xC0902, sector) // End of track  
 WritePort(0xC0902, 0x1B) // Gap length  
 WritePort(0xC0902, 0xFF) // Data length  
   
 // Wait for operation completion  
 WAIT\_READ\_COMPLETE:  
 status = ReadPort(0xC0900)  
 IF (status AND 0x10 == 0) THEN  
 GOTO WAIT\_READ\_COMPLETE  
 EndIF  
   
 // Read result bytes  
 result1 = ReadPort(0xC0902) // ST0  
 result2 = ReadPort(0xC0902) // ST1  
 result3 = ReadPort(0xC0902) // ST2  
   
 IF (result1 AND 0xC0 == 0x00) THEN  
 RETURN SUCCESS  
 ELSE  
 RETURN ERROR  
 EndIF  
END  
  
FUNCTION WriteSector(drive, track, head, sector, buffer)  
BEGIN  
 // Setup DMA for write operation  
 CallFunction ConfigureDMAForWrite(buffer, 512)  
   
 // Send write command to 8272  
 WritePort(0xC0901, 0x45) // Write data command  
 WritePort(0xC0902, (head << 2) | drive) // Head and drive  
 WritePort(0xC0902, track) // Track number  
 WritePort(0xC0902, head) // Head number  
 WritePort(0xC0902, sector) // Sector number  
 WritePort(0xC0902, 0x02) // 512 bytes per sector  
 WritePort(0xC0902, sector) // End of track  
 WritePort(0xC0902, 0x1B) // Gap length  
 WritePort(0xC0902, 0xFF) // Data length  
   
 // Wait for operation completion  
 WAIT\_WRITE\_COMPLETE:  
 status = ReadPort(0xC0900)  
 IF (status AND 0x10 == 0) THEN  
 GOTO WAIT\_WRITE\_COMPLETE  
 EndIF  
   
 // Read result bytes  
 result1 = ReadPort(0xC0902) // ST0  
 result2 = ReadPort(0xC0902) // ST1  
 result3 = ReadPort(0xC0902) // ST2  
   
 IF (result1 AND 0xC0 == 0x00) THEN  
 RETURN SUCCESS  
 ELSE  
 RETURN ERROR  
 EndIF  
END

## 2. 8237 DMA Controller for High-Speed Data

### Components used:

* 8237 DMA Controller
* Multiple DMA channels for different operations
* Memory buffers for data transfer

### Block Diagram:

┌─────────────┐  
| |  
| RAM | ┌─────────────┐  
| Buffer | | USB |  
| @0x40000 | | INTERFACE |  
| | | @0xC0700 |  
└─────────────┘ └─────────────┘  
 ^ ^  
 | |  
 | |  
┌──────────────┐ ┌──────────────┐  
| | | 8237 |  
| 8086 CPU |<──────────────| DMA |  
| | | CONTROLLER |  
└──────────────┘ └──────────────┘  
 | |  
 | ▼  
┌──────────────┐ ┌──────────────┐  
| 8272 | | FLOPPY |  
| FDC | | DRIVE |  
| @0xC0900 | └──────────────┘  
└──────────────┘

### Description:

#### 8237 DMA Controller

* Provides high-speed data transfer between memory and peripherals.
* Supports 4 DMA channels (0-3) for concurrent operations.
* Reduces CPU overhead during data transfer operations.
* Handles memory-to-peripheral and peripheral-to-memory transfers.

### Address Mapping:

| Component | Address Range |
| --- | --- |
| DMA Controller | 0x0000-0x000F |
| DMA Page Registers | 0x0080-0x008F |
| DMA Buffer | 0x40000-0x4FFFF |

### PSEUDOCODE

FUNCTION Initialize8237DMA()  
BEGIN  
 // Reset DMA controller  
 WritePort(0x000D, 0x00) // Master clear  
 WritePort(0x000C, 0x00) // Clear byte pointer  
   
 // Configure DMA channels  
 CallFunction ConfigureDMAChannel0() // Reserved  
 CallFunction ConfigureDMAChannel1() // Floppy disk  
 CallFunction ConfigureDMAChannel2() // USB transfers  
 CallFunction ConfigureDMAChannel3() // System integration  
   
 SHOW "8237 DMA Controller initialized"  
 RETURN SUCCESS  
END  
  
FUNCTION ConfigureDMAForRead(buffer, size)  
BEGIN  
 // Configure DMA Channel 1 for floppy read  
 WritePort(0x000A, 0x05) // Mask channel 1  
 WritePort(0x000C, 0x00) // Clear byte pointer  
 WritePort(0x000B, 0x46) // Single mode, read, channel 1  
   
 // Set buffer address  
 address = buffer  
 WritePort(0x0002, address AND 0xFF) // Address low byte  
 WritePort(0x0002, (address >> 8) AND 0xFF) // Address high byte  
 WritePort(0x0083, (address >> 16) AND 0xFF) // Page register  
   
 // Set transfer count  
 count = size - 1  
 WritePort(0x0003, count AND 0xFF) // Count low byte  
 WritePort(0x0003, (count >> 8) AND 0xFF) // Count high byte  
   
 // Enable DMA channel  
 WritePort(0x000A, 0x01) // Unmask channel 1  
   
 RETURN SUCCESS  
END  
  
FUNCTION ConfigureDMAForWrite(buffer, size)  
BEGIN  
 // Configure DMA Channel 1 for floppy write  
 WritePort(0x000A, 0x05) // Mask channel 1  
 WritePort(0x000C, 0x00) // Clear byte pointer  
 WritePort(0x000B, 0x4A) // Single mode, write, channel 1  
   
 // Set buffer address  
 address = buffer  
 WritePort(0x0002, address AND 0xFF) // Address low byte  
 WritePort(0x0002, (address >> 8) AND 0xFF) // Address high byte  
 WritePort(0x0083, (address >> 16) AND 0xFF) // Page register  
   
 // Set transfer count  
 count = size - 1  
 WritePort(0x0003, count AND 0xFF) // Count low byte  
 WritePort(0x0003, (count >> 8) AND 0xFF) // Count high byte  
   
 // Enable DMA channel  
 WritePort(0x000A, 0x01) // Unmask channel 1  
   
 RETURN SUCCESS  
END

## 3. USB Transfers Using DMA

### Components used:

* USB interface (external controller)
* 8237 DMA Controller
* Memory buffers for USB data

### Block Diagram:

┌─────────────┐  
| |  
| RAM | ┌─────────────┐  
| Buffer | | USB |  
| @0x41000 | | CONTROLLER |  
| | | @0xC0700 |  
└─────────────┘ └─────────────┘  
 ^ ^  
 | |  
 | |  
┌──────────────┐ ┌──────────────┐  
| | | 8237 |  
| 8086 CPU |<──────────────| DMA |  
| | | CONTROLLER |  
└──────────────┘ | Channel 2 |  
 └──────────────┘  
 |  
 ▼  
 ┌──────────────┐  
 | USB |  
 | DEVICE |  
 └──────────────┘

### Description:

#### USB Interface via DMA

* Handles USB data transfers using DMA Channel 2.
* Supports bulk data transfer for high-speed USB operations.
* Manages USB protocol through external USB controller.
* Provides efficient data movement between USB devices and system memory.

### Address Mapping:

| Component | Address Range |
| --- | --- |
| USB Controller | 0xC0700-0xC07FF |
| USB Buffer | 0x41000-0x41FFF |

### PSEUDOCODE

FUNCTION InitializeUSBDMA()  
BEGIN  
 // Configure DMA Channel 2 for USB operations  
 WritePort(0x000A, 0x06) // Mask channel 2  
 WritePort(0x000C, 0x00) // Clear byte pointer  
   
 // Set USB buffer address  
 address = 0x41000  
 WritePort(0x0004, address AND 0xFF) // Address low byte  
 WritePort(0x0004, (address >> 8) AND 0xFF) // Address high byte  
 WritePort(0x0081, (address >> 16) AND 0xFF) // Page register  
   
 SHOW "USB DMA initialized"  
 RETURN SUCCESS  
END  
  
FUNCTION USBTransferToMemory(size)  
BEGIN  
 // Configure DMA for USB read operation  
 WritePort(0x000A, 0x06) // Mask channel 2  
 WritePort(0x000B, 0x42) // Single mode, read, channel 2  
   
 // Set transfer count  
 count = size - 1  
 WritePort(0x0005, count AND 0xFF) // Count low byte  
 WritePort(0x0005, (count >> 8) AND 0xFF) // Count high byte  
   
 // Enable DMA channel  
 WritePort(0x000A, 0x02) // Unmask channel 2  
   
 // Start USB transfer  
 WritePort(0xC0700, 0x01) // Start USB read  
   
 // Wait for DMA completion  
 WAIT\_USB\_DMA:  
 status = ReadPort(0x0008)  
 IF (status AND 0x04 == 0) THEN  
 GOTO WAIT\_USB\_DMA  
 EndIF  
   
 RETURN SUCCESS  
END  
  
FUNCTION USBTransferFromMemory(size)  
BEGIN  
 // Configure DMA for USB write operation  
 WritePort(0x000A, 0x06) // Mask channel 2  
 WritePort(0x000B, 0x46) // Single mode, write, channel 2  
   
 // Set transfer count  
 count = size - 1  
 WritePort(0x0005, count AND 0xFF) // Count low byte  
 WritePort(0x0005, (count >> 8) AND 0xFF) // Count high byte  
   
 // Enable DMA channel  
 WritePort(0x000A, 0x02) // Unmask channel 2  
   
 // Start USB transfer  
 WritePort(0xC0700, 0x02) // Start USB write  
   
 // Wait for DMA completion  
 WAIT\_USB\_DMA:  
 status = ReadPort(0x0008)  
 IF (status AND 0x04 == 0) THEN  
 GOTO WAIT\_USB\_DMA  
 EndIF  
   
 RETURN SUCCESS  
END

## 4. System Integration

### Components used:

* All I/O controllers and interfaces
* Unified memory management
* Interrupt coordination

### Block Diagram:

┌─────────────┐  
| |  
| SYSTEM | ┌─────────────┐  
| MEMORY | | KEYBOARD |  
| @0x00000 | | CONTROLLER |  
| | | @0xC0400 |  
└─────────────┘ └─────────────┘  
 ^ ^  
 | |  
 | |  
┌──────────────┐ ┌──────────────┐  
| | | DISPLAY |  
| 8086 CPU |<──────────────| CONTROLLER |  
| | | @0xC0500 |  
└──────────────┘ └──────────────┘  
 | |  
 | ▼  
┌──────────────┐ ┌──────────────┐  
| 8237 | | FLOPPY |  
| DMA | | 8272 |  
| CONTROLLER | | @0xC0900 |  
└──────────────┘ └──────────────┘  
 | |  
 | ▼  
┌──────────────┐ ┌──────────────┐  
| USB | | SERIAL |  
| CONTROLLER | | @0xC0200 |  
| @0xC0700 | └──────────────┘  
└──────────────┘

### Description:

#### System Integration

* Coordinates all I/O operations through unified addressing.
* Manages data flow between different subsystems.
* Provides high-level functions for complex operations.
* Handles error conditions and system recovery.

### PSEUDOCODE

FUNCTION IntegrateAllSystems()  
BEGIN  
 // Initialize all controllers  
 CallFunction Initialize8272FDC()  
 CallFunction Initialize8237DMA()  
 CallFunction InitializeUSBDMA()  
   
 // Configure system memory buffers  
 CallFunction ConfigureSystemBuffers()  
   
 // Setup interrupt handlers  
 CallFunction SetupSystemInterrupts()  
   
 SHOW "All systems integrated successfully"  
 RETURN SUCCESS  
END  
  
FUNCTION ConfigureSystemBuffers()  
BEGIN  
 // Assign memory buffers for different operations  
 // Floppy buffer: 0x40000-0x407FF (2KB)  
 // USB buffer: 0x41000-0x417FF (2KB)  
 // System buffer: 0x42000-0x427FF (2KB)  
   
 // Clear all buffers  
 CallFunction FillMemory(0x40000, 0x800, 0x00) // Floppy buffer  
 CallFunction FillMemory(0x41000, 0x800, 0x00) // USB buffer  
 CallFunction FillMemory(0x42000, 0x800, 0x00) // System buffer  
   
 RETURN SUCCESS  
END  
  
FUNCTION DataFlowOperation(source, destination, size)  
BEGIN  
 // High-level data flow between different systems  
   
 IF (source == "FLOPPY" AND destination == "USB") THEN  
 // Read from floppy to memory  
 CallFunction ReadSector(0, 0, 0, 1, 0x40000)  
   
 // Transfer from memory to USB  
 CallFunction CopyBlock(0x40000, 0x41000, size)  
 CallFunction USBTransferFromMemory(size)  
   
 ELSE IF (source == "USB" AND destination == "FLOPPY") THEN  
 // Read from USB to memory  
 CallFunction USBTransferToMemory(size)  
   
 // Transfer from memory to floppy  
 CallFunction CopyBlock(0x41000, 0x40000, size)  
 CallFunction WriteSector(0, 0, 0, 1, 0x40000)  
   
 EndIF  
   
 RETURN SUCCESS  
END  
  
FUNCTION SystemHealthCheck()  
BEGIN  
 // Check all system components  
 floppyStatus = CallFunction CheckFloppyStatus()  
 usbStatus = CallFunction CheckUSBStatus()  
 dmaStatus = CallFunction CheckDMAStatus()  
   
 IF (floppyStatus == SUCCESS AND usbStatus == SUCCESS AND dmaStatus == SUCCESS) THEN  
 SHOW "All systems operational"  
 RETURN SUCCESS  
 ELSE  
 SHOW "System component failure detected"  
 RETURN ERROR  
 EndIF  
END

## 5. Integration Notes

### Memory Buffer Assignment:

* Floppy Data Buffer: 0x40000-0x407FF (2KB)
* USB Transfer Buffer: 0x41000-0x417FF (2KB)
* System Integration Buffer: 0x42000-0x427FF (2KB)

### DMA Channel Assignment:

* Channel 0: Reserved for system use
* Channel 1: Floppy disk operations (8272)
* Channel 2: USB transfers
* Channel 3: System integration and bulk transfers

### Address Coordination:

* Floppy Controller: 0xC0900-0xC09FF
* USB Controller: 0xC0700-0xC07FF
* All addresses coordinated with team assignments
* No conflicts with other subsystems

### Interrupt Integration:

* Floppy operations use IRQ6 (managed by 8259A)
* USB operations use IRQ5 (managed by 8259A)
* DMA completion signals handled through polling and interrupts

## Conclusion:

This document provides a complete implementation of storage and DMA integration systems for the 8086 microprocessor. The implementation focuses on efficient data transfer using DMA controllers, reliable floppy disk operations, and seamless USB integration. All components are designed to work together as part of the larger system architecture while maintaining compatibility with othe