

ND-100 Panel Controller Integration Analysis

MC68705 Panel/Calendar Controller with 3202D CBP Board and Delilah DGA CPU

Introduction

The Norsk Data ND-100 computer system employs a sophisticated three-tier architecture for panel control and system monitoring. At the heart of this system is the MC68705 microcontroller (chip 44A) which serves as a dedicated Panel/Calendar Controller, interfacing between the main Delilah DGA (Delilah Gate Array) CPU and the 3202D CBP (Control Board Panel) to provide real-time system status, CPU performance monitoring, and calendar/time management functions.

This document provides a comprehensive analysis of how these components integrate to form a complete system monitoring and control solution, including detailed signal analysis, command protocols, and data flow patterns.

Executive Summary

The MC68705 Panel/Calendar Controller represents a critical subsystem that offloads time-sensitive operations from the main ND-100 CPU while providing operators with essential system monitoring capabilities. Key functions include:

Primary Functions:

- **Real-time CPU performance monitoring** at 1200Hz sampling rate
- **PANC (Panel Control) command processing** from the Delilah DGA CPU
- **MM58274 Real-Time Clock management** for system time/date operations
- **7-segment display control** for operator panel indicators
- **System status monitoring** including interrupt, memory protection, and cache performance

Integration Benefits:

- **Reduced CPU overhead** by handling panel operations independently
 - **Real-time responsiveness** for critical system monitoring
 - **Operator visibility** into CPU utilization, system load, and hardware status
 - **Centralized time management** independent of main CPU operations
-

System Architecture Overview

Component Hierarchy

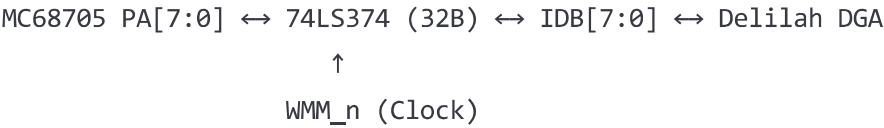
ND-100 SYSTEM
Delilah DGA (CPU) <ul style="list-style-type: none">IDB[15:0] (Internal Data Bus)/EMP_n (Enable Memory Protection)CPU Status (Ring, Level, PONI, IONI)PANC Command Generation
3202D CBP (Control Board Panel) <ul style="list-style-type: none">74LS374 Latch (32B) - IDB[7:0] InterfaceWMM_n Clock SignalPA[7:0] Bidirectional Data PathPanel Display Drivers
MC68705 Panel Controller (44A) <ul style="list-style-type: none">CPU Performance Monitoring (1200Hz)PANC Command ProcessingMM58274 RTC InterfaceDisplay Control (DISP1-5, STAT[4:0])System Status Integration

Hardware Integration Details

MC68705 Port Configuration and Signal Mapping

Port A (PA[7:0]) - IDB Data Interface

- Direction:** Bidirectional (controlled by DDRA)
- Function:** Primary data path to/from Delilah DGA via 3202D CBP
- Connection:** PA[7:0] → 74LS374 Latch → IDB[7:0]
- Protocol:** PANC command data exchange



Port B (PB[7:0]) - Control and RTC Interface

- Direction:** Output (DDRB = 0xFF)
- Initial Value:** 0x2F (0010 1111)

Pin	Signal	Function	Notes
PB0	WMM_n Control	IDB Latch Strobe	Controls 74LS374 output enable
PB1	Reserved	-	-
PB2	RTC Reset/Enable	MM58274 Control	0=Reset, 1=Enable
PB3	IDB Read Strobe	Data Clock	Strobes PA data reads
PB4	Command Ready	Status Signal	Cleared during /EMP_n wait
PB5	Sync Control	Timer Sync	Cleared during timer sync
PB6	Mode Control	System Mode	Set during RTC operations
PB7	Reserved	-	-

Port C (PC[7:0]) - RTC Data Interface

- **Direction:** Output (DDRC = 0xFF)
- **Initial Value:** 0xF8 (1111 1000)
- **Function:** Direct communication with MM58274 RTC chip
- **Protocol:** RTC command and data transmission

Pin	Signal	Function	RTC Interface
PC[2:0]	RTC Address	Register Select	MM58274 register addressing
PC[7:3]	RTC Data/Control	Data Lines	MM58274 data and control signals

Port D (PD[7:0]) - System Status Monitoring

- **Direction:** Input (DDRD = 0x00)
- **Function:** Real-time monitoring of Delilah DGA CPU status
- **Sampling Rate:** 1200Hz via timer interrupt

Pin	Signal	Source	Function
PD7	/EMP_n	Delilah DGA	Enable Memory Protection (Command Ready)
PD6	LHIT	Cache Controller	Load Hit (Cache Hit Indicator)
PD5	Reserved	-	-
PD4	Reserved	-	-
PD3	IONI	Interrupt Controller	Interrupt System ON
PD2	PONI	Memory Controller	Memory Protection ON
PD[1:0]	PCR Ring	CPU PCR Register	Current Ring Level (0-3)

Signal Flow and Data Paths

1. Command Processing Data Flow

Delilah DGA CPU
↓ (Generate PANC Command)
IDB[15:0] Bus
↓ (IDB[7:0] to Panel Controller)
74LS374 Latch (32B)
↓ (WMM_n strobe)
MC68705 Port A
↓ (Process Command)
MC68705 Internal Processing
↓ (Generate Response)
MC68705 Port A
↓ (WMM_n latch)
74LS374 Output
↓ (Back to CPU)
IDB[7:0] → Delilah DGA

2. Real-Time Monitoring Data Flow

Delilah DGA CPU Status
↓ (Ring, Level, PONI, IONI, Cache)
MC68705 Port D
↓ (1200Hz sampling)
Timer Interrupt Handler
↓ (Accumulate Statistics)
CPU Utilization Calculation
↓ (Format for Display)
Port B/C Control
↓ (Display Updates)
DISP[5:1] + STAT[4:0] Output
↓ (To Operator Panel)
3202D CBP Display Drivers

3. RTC Interface Data Flow

MM58274 RTC Chip
↔ (Control/Data Lines)
MC68705 Port C
↔ (Time/Date Operations)
Internal RTC Buffer
↔ (PANC Commands)
MC68705 Port A
↔ (To/From DGA)
Delilah DGA CPU

PANC Command Protocol Analysis

Command Byte Format






Bit 7-4: Command Flags and Options
Bit 3: Read/Write Direction (0=Write to CPU, 1=Read from CPU)
Bit 2-0: Command Type (0-7)

Complete Command Set with Detailed Analysis

Command 0: Set Complete Time and Date

- **Command Byte:** $(0x08 \mid 0x00) = 0x08$ (Read operation)
- **Purpose:** Set full RTC time and date from DGA CPU
- **Data Sequence:** 5 bytes (Seconds, Hours, Minutes, Month, Day)

Internal MC68705 Operations:

-  **Reads PA:** 5 sequential reads from Port A
-  **Writes 68705 RAM:** Updates internal time variables (0x14-0x18)
-  **Updates Display:** Sets display to "PEXM" (normal) or "PT###" (test mode)
-  **RTC Operations:** No direct MM58274 write (processed later)
-  **STAT/DISP Update:** Display chars set, but not output yet

c






```
// Command 0 Processing
rtc_datetime.seconds = Read_PA_Data(); // 0x14
rtc_datetime.hours   = Read_PA_Data(); // 0x16
rtc_datetime.minutes = Read_PA_Data(); // 0x15
rtc_datetime.month   = Read_PA_Data(); // 0x18
rtc_datetime.day      = Read_PA_Data(); // 0x17

// Display mode selection
if (system_config & 0x04) {
    display = "PT##"; // Test mode with calculated digits
} else {
    display = "PEXM"; // Normal panel mode
}
```

Command 1: Set Time Only (No Date)

- **Command Byte:** $(0x08 \mid 0x01) = 0x09$
- **Purpose:** Set only time components, preserve date
- **Data Sequence:** 4 bytes (Month, Day, Hours, Minutes)





Internal MC68705 Operations:

-  **Reads PA:** 4 sequential reads from Port A
-  **Writes 68705 RAM:** Updates time variables, zeros seconds
-  **Updates Display Control:** Sets control flags (bit 4 = 1)
-  **RTC Operations:** No immediate MM58274 access
-  **STAT/DISP Update:** Control flags updated only

Command 2: Read System Status

- **Command Byte:** $(0x08 \mid 0x02) = 0x0A$
- **Purpose:** Read PONI/IONI status and format for display
- **Data Sequence:** 2 bytes (Month, Day parameters)

Internal MC68705 Operations:

-  **Reads PA:** 2 reads from Port A
-  **Reads Port D:** Samples PONI/IONI status from hardware
-  **Writes 68705 RAM:** Formats status for display characters
-  **Updates Display:** Creates status display representation

- **✗ RTC Operations:** No RTC access
- **✓ STAT Update:** Status formatted for STAT output

c

```
// Command 2 Processing
uint8_t poni_ioni_status = PORTD & 0x3C; // Bits 5-2
display_chars.char2 = (poni_ioni_status & 0x0F) + 0x10; // IONI status
display_chars.char3 = ((poni_ioni_status & 0x30) >> 4) + 0x3A; // PONI status
```

Command 3: Set Display Mode

- **Command Byte:** $(0x08 \mid 0x03) = 0x0B$
- **Purpose:** Configure display mode and parameters
- **Data Sequence:** 3 bytes (Hours, Month, Day)

Internal MC68705 Operations:

- **✓ Reads PA:** 3 reads from Port A
- **✓ Writes 68705 RAM:** Updates display mode settings
- **✓ Updates Display Control:** Sets mode control flags (bit 3 = 1)
- **✗ RTC Operations:** No RTC access
- **✗ STAT/DISP Update:** Mode set but not yet output

Command 4: Load Display Characters from ROM

- **Command Byte:** $(0x08 \mid 0x04) = 0x0C$
- **Purpose:** Load custom display characters from lookup tables
- **Data Sequence:** 1 byte (character selection index)

Internal MC68705 Operations:

- **✓ Reads PA:** 1 read from Port A
- **✓ Reads ROM:** Accesses lookup tables at 0x0D12 and 0x0D1A
- **✓ Writes 68705 RAM:** Loads 4 display characters
- **✗ RTC Operations:** No RTC access
- **✓ Updates Display:** Custom characters loaded for panel

c






```
// Command 4 Processing - ROM Lookup Tables
uint8_t char_select = Read_PA_Data();
uint8_t index1 = (char_select & 0x03) * 2;           // Bits 1-0 → Table 1
uint8_t index2 = ((char_select & 0x18) >> 2);       // Bits 4-3 → Table 2

display_chars.char3 = ROM_TABLE_1[index1];           // 0x0D12 table
display_chars.char2 = ROM_TABLE_1[index1 + 1];
display_chars.char1 = ROM_TABLE_2[index2];           // 0x0D1A table
display_chars.char0 = ROM_TABLE_2[index2 + 1];
```

Command 5: Set System Configuration

- **Command Byte:** $(0x08 \mid 0x05) = 0x0D$
- **Purpose:** Update system configuration flags
- **Data Sequence:** 1 byte (configuration value)





Internal MC68705 Operations:

-  **Reads PA:** 1 read from Port A
-  **Writes 68705 RAM:** Updates system_config variable (0x1A)
-  **Display Update:** No immediate display change
-  **RTC Operations:** No RTC access
-  **STAT/DISP Update:** Configuration stored only

Command 6: Set Direct Display Data

- **Command Byte:** $(0x08 \mid 0x06) = 0x0E$
- **Purpose:** Directly set 4 display characters
- **Data Sequence:** 4 bytes (char1, char0, char3, char2)

Internal MC68705 Operations:






-  **Reads PA:** 4 sequential reads from Port A
-  **Writes 68705 RAM:** Direct update of display characters
-  **RTC Operations:** No RTC access
-  **Updates Display:** Immediate display character update

Command 7: System Control and Reset

- **Command Byte:** $(0x08 \mid 0x07) = 0x0F$
- **Purpose:** System control operations including soft reset

- **Data Sequence:** 2 bytes (param1, param2)

Internal MC68705 Operations:

-  **Reads PA:** 2 reads from Port A
-  **Writes 68705 RAM:** Updates control parameters
-  **Conditional Reset:** If param2[7:4] = 0x4, performs soft reset
-  **Feature Control:** If param2[7:4] = 0x1, enables features
-  **RTC Operations:** No RTC access (unless reset)

c

```
// Command 7 Processing - System Control
uint8_t param1 = Read_PA_Data();
uint8_t param2 = Read_PA_Data();





if ((param2 & 0xF0) == 0x40) {
    // Soft Reset Sequence
    MR = 0x40;
    TCR = 0x47;
    PCR = 1;
    DDRA = DDRB = DDRC = 0; // Reset all ports
    RESET(); // Jump to reset vector
}

if ((param2 & 0xF0) == 0x10) {
    utilization_control |= 0x04; // Enable feature
}
```

Read Operations (Command Bit 3 = 0)

When bit 3 of the command is 0, the MC68705 performs a **write operation to the CPU**:

Internal MC68705 Operations:

-  **Reads RTC:** Accesses MM58274 for current time/date
-  **Writes PA:** Outputs current data to Port A
-  **Updates STAT/DISP:** Formats data for panel display
-  **Writes 68705 RAM:** Updates internal state

C

```
// Read Operation Processing
void Output_RTC_Data_To_CPU(void) {
    uint8_t command_flags = received_command & 0x07;

    // Set Port C for RTC addressing
    PORTC = (PORTC & 0xF8) | command_flags;

    if (received_command & 0x20) {
        // RTC read operation
        PORTB |= 0x40; // Set RTC read mode
        current_data = Read_MM58274_Register();
        Write_PA_Data(current_data);
    } else {
        // Internal data output
        Write_PA_Data(internal_buffer[command_flags]);
    }
}
```

MM58274 Real-Time Clock Integration

RTC Command Interface (via Port C)

The MC68705 communicates with the MM58274 using a sophisticated protocol:

RTC Register Addressing

- PC[2:0] = RTC Register Address (0-7)
- PC[7:3] = RTC Control and Data Lines

RTC Register Map

Address	Register	Function	MC68705 Access
0	Seconds	Current seconds (0-59)	Read/Write via Command 0,1
1	Minutes	Current minutes (0-59)	Read/Write via Command 0,1
2	Hours	Current hours (0-23)	Read/Write via Command 0,1
3	Day of Week	Day (1-7)	Read/Write via Command 0
4	Day	Day of month (1-31)	Read/Write via Command 0
5	Month	Month (1-12)	Read/Write via Command 0
6	Year	Year (0-99)	Read/Write via Command 0
7	Control	RTC control register	Internal use only

RTC Operation Sequences

RTC Write Operation:

```
c
void Write_MM58274_Register(uint8_t address, uint8_t data) {
    PORTC = (PORTC & 0xF8) | address; // Set register address
    PORTB &= 0xDF; // Clear write enable
    PORTC = (PORTC & 0x07) | (data << 3); // Set data
    PORTB |= 0x20; // Strobe write
}
```

RTC Read Operation:

```
c
uint8_t Read_MM58274_Register(uint8_t address) {
    PORTC = (PORTC & 0xF8) | address; // Set register address
    PORTB |= 0x40; // Set read mode
    return (PORTC >> 3) & 0x1F; // Return data bits
}
```

Display and Status Output Integration

7-Segment Display Control (DISP[5:1])

The MC68705 generates display control signals for the operator panel:

Display Signal Mapping

MC68705 → Display Drivers → 7-Segment Panel		
↓	↓	↓
DISP1	Driver 1	Digit 1 (Leftmost)
DISP2	Driver 2	Digit 2
DISP3	Driver 3	Digit 3
DISP4	Driver 4	Digit 4
DISP5	Driver 5	Digit 5 (Rightmost)

Display Character Encoding

The MC68705 uses ROM lookup tables to convert ASCII characters to 7-segment patterns:

c

```
// 7-Segment Encoding Process
void Format_7Segment_Display_Output(void) {
    for (int i = 0; i < 4; i++) {
        uint8_t ascii_char = display_chars[i];
        uint8_t segment_pattern = ROM_7SEG_TABLE[ascii_char];
        segment_display.segment_data[i] = segment_pattern;
    }

    // Output to display drivers
    Update_DISP_Signals();
}
```

Status Output (STAT[4:0])

Status bits are output to IDB[11:8] for system monitoring:

Bit	Signal	Function	Source
STAT4	CPU Busy	CPU utilization > threshold	Ring 1-3 time percentage
STAT3	System Load	High system activity	Ring 2-3 time percentage
STAT2	RTC Valid	RTC data valid	MM58274 status
STAT1	Cache Performance	Cache hit rate > threshold	LHIT statistics
STAT0	Error Status	System error detected	Various error conditions

c

```
// Status Generation
void Update_STAT_Output(void) {
    uint8_t status = 0;

    if (cpu_utilization > 75) status |= 0x10; // STAT4: High CPU usage
    if (system_load > 50) status |= 0x08; // STAT3: High system Load
    if (rtc_valid) status |= 0x04; // STAT2: RTC operational
    if (cache_hit_rate > 90) status |= 0x02; // STAT1: Good cache performance
    if (error_detected) status |= 0x01; // STAT0: Error condition

    Output_To_IDB_High_Bits(status);
}
```

Performance Monitoring Integration

CPU Utilization Calculation

The MC68705 implements sophisticated CPU performance monitoring:

Ring Level Analysis

```
c

// 1200Hz Sampling of CPU Status
void TIMER_INTERRUPT_CPU_Monitor(void) {
    uint8_t cpu_status = PORTD;
    uint8_t current_ring = cpu_status & 0x03; // PCR Ring (0-3)

    // Accumulate time per ring level
    ring_counters[current_ring]++;
    total_samples++;

    // Track system status
    if (cpu_status & 0x08) interrupt_active_samples++; // IONI
    if (cpu_status & 0x04) memory_protected_samples++; // PONI
    if (cpu_status & 0x40) cache_hit_samples++; // LHIT

    // Calculate utilization every 200ms
    if (total_samples % 240 == 0) { // 1200Hz / 240 = 5Hz update
        Calculate_Utilization_Percentages();
    }
}

void Calculate_Utilization_Percentages(void) {
    // User time = Ring 0
    uint16_t user_time = ring_counters[0];

    // System time = Ring 1-3
    uint16_t system_time = ring_counters[1] + ring_counters[2] + ring_counters[3];

    // CPU utilization = (Total - Ring0) / Total * 100
    cpu_utilization = (system_time * 100) / total_samples;

    // System Load = (Ring2 + Ring3) / Total * 100
    system_load = ((ring_counters[2] + ring_counters[3]) * 100) / total_samples;

    // Reset counters for next period
    memset(ring_counters, 0, sizeof(ring_counters));
    total_samples = 0;
}
```

Performance Metrics Output

The calculated performance data is made available through multiple channels:

1. **PANC Commands:** CPU utilization data accessible via command protocol
 2. **STAT Signals:** Real-time status bits for immediate feedback
 3. **Display Output:** Formatted performance data on 7-segment displays
 4. **IDB Interface:** Detailed metrics available to main CPU
-

System Integration Benefits

Real-Time Performance

- **1200Hz monitoring** provides microsecond-level CPU activity tracking
- **Independent operation** ensures monitoring continues during CPU intensive tasks
- **Hardware integration** eliminates software overhead for performance measurement

Operator Interface

- **Immediate visual feedback** through panel displays and status lights
- **Historical data** through time-averaged utilization calculations
- **System health indicators** via STAT outputs and error detection

System Architecture

- **Distributed processing** reduces main CPU overhead
 - **Dedicated timing** ensures accurate time/date management
 - **Scalable monitoring** supports additional performance metrics
-

Conclusion

The MC68705 Panel/Calendar Controller represents a sophisticated integration of hardware monitoring, real-time processing, and system interface design. By offloading critical monitoring and panel control functions from the main Delilah DGA CPU, it enables the ND-100 system to provide comprehensive operator visibility while maintaining optimal performance for computational tasks.

The detailed command protocol, real-time monitoring capabilities, and hardware integration demonstrate advanced system design principles that remain relevant for modern embedded system architectures. The combination of dedicated hardware monitoring, sophisticated data processing, and comprehensive operator interface creates a robust foundation for system management and performance optimization.