# MC68705 to MM58274 RTC Interface Analysis

## Complete Communication Protocol and Register Mapping

## MM58274 Real-Time Clock Integration

Based on the MM58274 datasheet and MC68705 code analysis, here's the complete interface implementation:

### Hardware Connection Summary

```
MC68705          MM58274 RTC
-------          -----------
PB1  (/WRCLK) → /WR (Write Enable)
PB2  (/ROCLK) → /RD (Read Enable)
PC[3:0]       → AD[3:0] (Address/Data Bus)
PA[7:0]       ↔ Data Bus (via multiplexed I/O)
```

### Port B Control Signal Mapping

| MC68705 Pin | Signal | MM58274 Pin | Function |
|---|---|---|---|
| PB0 | WMM_n Control | - | IDB Latch Control (not RTC) |
| PB1 | /WRCLK | /WR | MM58274 Write Enable (Active Low) |
| PB2 | /ROCLK | /RD | MM58274 Read Enable (Active Low) |
| PB3 | WMM Strobe | - | IDB Data Strobe (not RTC) |
| PB4 | Command Ready | - | System Control |
| PB5 | Sync Control | - | Timer Synchronization |
| PB6 | RTC Mode | - | RTC Operation Mode |
| PB7 | Reserved | - | Unused |

### Port C Address/Data Interface

| MC68705 Pin | Signal | MM58274 Pin | Function |
|---|---|---|---|
| PC[3:0] | RTC Address/Data | AD[3:0] | Multiplexed Address/Data Bus |
| PC[7:4] | Control/Status | - | Internal MC68705 control |

## MM58274 Register Map and MC68705 Memory Mapping

### MM58274 Internal Registers (from datasheet)

| Address | Register | MC68705 RAM | Function | Access |
|---|---|---|---|---|
| 0x0 | Control Register | - | Clock control | R/W (Split) |
| 0x1 | Tenths of Seconds | 0x20 | 0.1 second counter | Read Only |
| 0x2 | Units Seconds | 0x21 | Seconds (0-9) | R/W |
| 0x3 | Tens Seconds | 0x22 | Seconds tens (0-5) | R/W |
| 0x4 | Units Minutes | 0x23 | Minutes (0-9) | R/W |
| 0x5 | Tens Minutes | 0x24 | Minutes tens (0-5) | R/W |
| 0x6 | Units Hours | 0x25 | Hours (0-9) | R/W |
| 0x7 | Tens Hours | 0x26 | Hours tens (0-2) | R/W |
| 0x8 | Units Days | RAM_0017 | Days (0-9) | R/W |
| 0x9 | Tens Days | RAM_0018 | Days tens (0-3) | R/W |
| 0xA | Units Months | RAM_0014 | Months (1-9) | R/W |
| 0xB | Tens Months | - | Months tens (0-1) | R/W |
| 0xC | Units Years | RAM_0015 | Years (0-9) | R/W |
| 0xD | Tens Years | RAM_0016 | Years tens (0-9) | R/W |
| 0xE | Day of Week | - | Day (1-7) | R/W |
| 0xF | Clock Setting/Interrupt | - | Control/Status | R/W |

## RTC Communication Protocol Implementation

### MM58274 Initialization Sequence

```c
void Initialize_MM58274_RTC_Chip(void) {
    // Reset RTC by clearing /ROCLK (PB2 = 0)
    PORTB &= 0xFB;  // Clear PB2 (/ROCLK low = reset)

    // Clear all MM58274 register buffers in MC68705 RAM
    MM58274_Tens_Hours = 0;      // 0x26
    MM58274_Tenths_Seconds = 0;  // 0x20
    MM58274_Units_Seconds = 0;   // 0x21
    MM58274_Units_Minutes = 0;   // 0x22
    MM58274_Tens_Seconds = 0;    // 0x23
    MM58274_Tens_Minutes = 0;    // 0x24
    MM58274_Units_Hours = 0;     // 0x25
    segment_display.segment_data[0] = 0;  // 0x52

    PORTA = 0;      // Clear data bus
    PORTB |= 0x04;  // Set PB2 (/ROCLK high = enable RTC)
    DDRA = 0;       // Port A as input
}
```

## MM58274 Read Operation

```c
uint8_t Read_MM58274_Register(uint8_t address) {
    // Step 1: Set address on PC[3:0]
    PORTC = (PORTC & 0xF8) | (address & 0x0F);

    // Step 2: Assert /ROCLK (PB2 = 0) to start read cycle
    PORTB &= ~0x04;  // Clear PB2 (/RD low)

    // Step 3: Read data from AD[3:0] via PC[3:0]
    uint8_t data = PORTC & 0x0F;

    // Step 4: Deassert /ROCLK (PB2 = 1) to end read cycle
    PORTB |= 0x04;   // Set PB2 (/RD high)

    return data;
}
```

## MM58274 Write Operation

```c
void Write_MM58274_Register(uint8_t address, uint8_t data) {
    // Step 1: Set address on PC[3:0]
    PORTC = (PORTC & 0xF8) | (address & 0x0F);

    // Step 2: Set data on PC[3:0] (address/data multiplexed)
    PORTC = (PORTC & 0xF0) | (data & 0x0F);

    // Step 3: Assert /WRCLK (PB1 = 0) to start write cycle
    PORTB &= ~0x02;  // Clear PB1 (/WR low)

    // Step 4: Deassert /WRCLK (PB1 = 1) to latch data
    PORTB |= 0x02;   // Set PB1 (/WR high)
}
```

---

## MC68705 RTC Interface Functions Analysis

### Function: Read_MM58274_Time_Date_Output_To_DGA

This function handles PANC read operations (when command bit 3 = 0):

c

```
void Read_MM58274_Time_Date_Output_To_DGA(void) {
    uint8_t command_flags = received_command;
    uint8_t original_portb = PORTB;

    // Extract register address from command (bits 2-0)
    uint8_t rtc_address = command_flags & 0x07;

    // Set MM58274 address on Port C
    PORTC = (PORTC & 0xF8) | rtc_address;

    // Check if this requires special handling
    if ((command_flags & 0x04) == 0) {
        return;  // No RTC operation needed
    }

    PORTB &= 0xDF;  // Clear control bit

    // Invert address bits for MM58274 addressing
    rtc_address = (command_flags & 0x03) ^ 0x03;

    if ((command_flags & 0x20) == 0) {
        // Standard read operation
        PORTB = original_portb & 0x9F;  // Clear read control bits

        // Read data from MM58274
        uint8_t rtc_data = Strobe_WMM_Read_IDB_Data();

        // Store in buffer for later output
        *(uint8_t*)(rtc_address + 0x47) = rtc_data;

        // Output to DGA via IDB
        Write_RegA_To_PortA_And_Latch_to_IDB(rtc_data);

        PORTB |= 0x20;  // Set completion flag

        // Special handling for address 0 (control register)
        if (rtc_address == 0) {
            Copy_RTC_Buffer_To_Display();
            Write_Complete_DateTime_To_DGA();
        }
    } else {
        // RTC write operation
        PORTB |= 0x40;  // Set write mode

        // Read data to write from IDB
        uint8_t write_data = Strobe_WMM_Read_IDB_Data();
        // Special case: reinitialize RTC if writing to control register
        if (rtc_address == 3) {
            Initialize_MM58274_RTC_Chip();
            Calculate_RTC_Display_Format();
        }

        // Write data to MM58274
        Write_MM58274_Register(rtc_address, write_data);

        // Output confirmation to DGA
        Write_RegA_To_PortA_And_Latch_to_IDB(write_data);

        PORTB |= 0x20;  // Set completion flag
    }
}
```

## Clock Setting and Data-Changed Flag

The MM58274 has special features utilized by the MC68705:

### Clock Setting Pulse (Bit 2 of Control Register)

- **Purpose**: Synchronizes all time registers
- **MC68705 Usage**: Used during time updates to ensure atomic changes
- **Implementation**: Set during write operations, cleared after completion

### Data-Changed Flag (Bit 3 of Control Register)

- **Purpose**: Indicates time data has been updated since last read
- **MC68705 Usage**: Polled to determine if time display needs refresh
- **Implementation**: Cleared by reading control register

## 12-Hour vs 24-Hour Mode

The MM58274 supports both 12-hour and 24-hour operation:

```c
// Hours register format in 12-hour mode
// Bit 0 of clock setting register: 0=24hr, 1=12hr
// Tens hours register: bit 1 = AM/PM (0=AM, 1=PM)

void Set_RTC_12Hour_Mode(bool enable_12hr, bool pm_flag) {
    uint8_t control_reg = Read_MM58274_Register(0x0);

    if (enable_12hr) {
        control_reg |= 0x01;  // Enable 12-hour mode
        Write_MM58274_Register(0x0, control_reg);

        // Set AM/PM flag in tens hours register
        uint8_t tens_hours = Read_MM58274_Register(0x7);
        if (pm_flag) {
            tens_hours |= 0x02;  // Set PM
        } else {
            tens_hours &= ~0x02; // Set AM
        }
        Write_MM58274_Register(0x7, tens_hours);
    } else {
        control_reg &= ~0x01; // Enable 24-hour mode
        Write_MM58274_Register(0x0, control_reg);
    }
}
```

---

## Time/Date Update Sequences

### Complete Time/Date Set (PANC Command 0)

c

```
// Command 0: Set complete time and date
void Set_Complete_RTC_DateTime(void) {
    // Read time/date from DGA via IDB
    uint8_t seconds = Strobe_WMM_Read_IDB_Data();
    uint8_t hours   = Strobe_WMM_Read_IDB_Data();
    uint8_t minutes = Strobe_WMM_Read_IDB_Data();
    uint8_t month   = Strobe_WMM_Read_IDB_Data();
    uint8_t day     = Strobe_WMM_Read_IDB_Data();

    // Set clock setting pulse to synchronize updates
    uint8_t control = Read_MM58274_Register(0x0) | 0x04;
    Write_MM58274_Register(0x0, control);

    // Update MM58274 registers
    Write_MM58274_Register(0x2, seconds % 10);      // Units seconds
    Write_MM58274_Register(0x3, seconds / 10);      // Tens seconds
    Write_MM58274_Register(0x4, minutes % 10);      // Units minutes
    Write_MM58274_Register(0x5, minutes / 10);      // Tens minutes
    Write_MM58274_Register(0x6, hours % 10);        // Units hours
    Write_MM58274_Register(0x7, hours / 10);        // Tens hours
    Write_MM58274_Register(0x8, day % 10);          // Units days
    Write_MM58274_Register(0x9, day / 10);          // Tens days
    Write_MM58274_Register(0xA, month % 10);        // Units months
    Write_MM58274_Register(0xB, month / 10);        // Tens months

    // Clear clock setting pulse to start counting
    control = Read_MM58274_Register(0x0) & ~0x04;
    Write_MM58274_Register(0x0, control);

    // Update display based on mode
    if (system_config & 0x04) {
        Set_Display_Test_Mode();
    } else {
        Set_Display_Normal_Mode("PEXM");
    }
}
```

### Time-Only Update (PANC Command 1)

```c
// Command 1: Set time only, preserve date
void Set_RTC_Time_Only(void) {
    uint8_t month   = Strobe_WMM_Read_IDB_Data();
    uint8_t day     = Strobe_WMM_Read_IDB_Data();
    uint8_t hours   = Strobe_WMM_Read_IDB_Data();
    uint8_t minutes = Strobe_WMM_Read_IDB_Data();

    // Set clock setting pulse
    uint8_t control = Read_MM58274_Register(0x0) | 0x04;
    Write_MM58274_Register(0x0, control);

    // Update only time registers, zero seconds
    Write_MM58274_Register(0x2, 0);               // Units seconds = 0
    Write_MM58274_Register(0x3, 0);               // Tens seconds = 0
    Write_MM58274_Register(0x4, minutes % 10);   // Units minutes
    Write_MM58274_Register(0x5, minutes / 10);   // Tens minutes
    Write_MM58274_Register(0x6, hours % 10);     // Units hours
    Write_MM58274_Register(0x7, hours / 10);     // Tens hours

    // Clear clock setting pulse
    control = Read_MM58274_Register(0x0) & ~0x04;
    Write_MM58274_Register(0x0, control);

    // Set time-only display mode
    display_control_flags |= 0x10;
}
```

## Real-Time Data Monitoring

### Continuous Time Reading for Display

The MC68705 periodically reads the MM58274 to update displays:

```c
void Update_RTC_Display_Data(void) {
    // Read current time from MM58274
    uint8_t tenths_sec  = Read_MM58274_Register(0x1);
    uint8_t units_sec   = Read_MM58274_Register(0x2);
    uint8_t tens_sec    = Read_MM58274_Register(0x3);
    uint8_t units_min   = Read_MM58274_Register(0x4);
    uint8_t tens_min    = Read_MM58274_Register(0x5);
    uint8_t units_hour  = Read_MM58274_Register(0x6);
    uint8_t tens_hour   = Read_MM58274_Register(0x7);

    // Store in MC68705 buffer for display processing
    MM58274_Tenths_Seconds = tenths_sec;
    MM58274_Units_Seconds = units_sec;
    MM58274_Tens_Seconds = tens_sec;
    MM58274_Units_Minutes = units_min;
    MM58274_Tens_Minutes = tens_min;
    MM58274_Units_Hours = units_hour;
    MM58274_Tens_Hours = tens_hour;

    // Check data-changed flag
    uint8_t control = Read_MM58274_Register(0x0);
    if (control & 0x08) {
        // Data has changed, update displays
        Format_Time_For_7Segment_Display();
        Update_Panel_Time_Display();

        // Clear data-changed flag by reading control register again
        Read_MM58274_Register(0x0);
    }
}
```

## Integration with ND-100 Time Base

### ND-100 Time Format Conversion

The ND-100 uses a specific time format (half-days since 1979-01-01):

```c
void Convert_ND100_To_MM58274_Format(uint16_t nd100_days, uint16_t nd100_seconds) {
    // ND-100 format: half-days since 1979-01-01, seconds since midnight/noon

    // Convert half-days to actual days
    uint16_t actual_days = nd100_days / 2;
    bool past_noon = (nd100_days & 1) != 0;

    // Add noon offset if in PM half-day
    uint32_t total_seconds = nd100_seconds;
    if (past_noon) {
        total_seconds += 12 * 3600;  // Add 12 hours
    }

    // Convert to hours, minutes, seconds
    uint8_t hours = (total_seconds / 3600) % 24;
    uint8_t minutes = (total_seconds / 60) % 60;
    uint8_t seconds = total_seconds % 60;

    // Calculate date from days since 1979-01-01
    // (Simplified - real implementation would handle leap years)
    uint16_t year = 1979 + (actual_days / 365);
    uint16_t day_of_year = actual_days % 365;

    // Convert day of year to month/day (simplified)
    uint8_t month, day;
    Convert_DayOfYear_To_MonthDay(day_of_year, &month, &day);

    // Update MM58274 with converted values
    Set_Complete_RTC_DateTime_Internal(hours, minutes, seconds, month, day, year % 100);
}
```

---

## Summary

The MC68705 implements a sophisticated interface to the MM58274 RTC that provides:

1. **Complete time/date management** independent of the main ND-100 CPU
2. **Precise timing synchronization** using clock setting pulses
3. **Bidirectional communication** for both reading current time and setting new values
4. **Format conversion** between ND-100 time format and standard date/time
5. **Real-time display updates** with automatic change detection
6. **Integration with panel displays** for operator visibility

The interface efficiently handles the MM58274's multiplexed address/data bus and leverages advanced features like the data-changed flag for optimal performance.