ND-100/120 Panel Controller - Complete Command Analysis

1. Summary Section

1.1 Command Format Structure

Input Command Format (PORTA bits 5-0):

```
Bit Layout:
  7   6   5   4   3   2   1   0
┌─┬─┬─┬─┬─┬─┬─┬─┐
│S│C│      COMMAND      │
└─┴─┴───────────────────┘
  │ │         │
  │ │         └── 6-bit command code (0x00-0x3F)
  │ └──────────── Button change flag (0x40)
  └────────────── Status/control bit (0x80)
```

Command Processing Steps:

1. Extract command: `command = PORTA & 0x3F`
2. Select lookup table: `table = (DisplayControlFlags & 0x10) ? 0x8B : 0x80`
3. Lookup dispatch: `dispatch_code = table[command + 1] << 1`
4. Execute handler: `switch(dispatch_code)` with cases 0x00-0xFE (even numbers)

Response Format (PORTB output):

```
Bit Layout:
  7   6   5   4   3   2   1   0
┌─┬─┬─┬─┬─┬─┬─┬─┐
│S│      RESPONSE DATA      │
└─┴─────────────────────────┘
  │          │
  │          └── 7-bit response data
  └───────────── Status bit (combined with DisplayControlFlags)
```

Response Protocol:

```
c

PORTC = PORTC & 0xFE;           // Clear strobe
PORTB = (data | status) & 0x7F; // Set response + status
PORTC = PORTC | 1;              // Set strobe (data valid)
```

1.2 Command Summary Table

Dispatch	Handler	Command Name	Response	Address	Notes
0x00	caseD_34	Display Update	Status	0x01DB	Display control operation
0x02	caseD_2a	Conditional Update	Status	0x01D1	Conditional display operation
0x04	caseD_46	Multi-stage Update	Status	0x01ED	Complex display sequence
0x06	caseD_56	*Unknown*	Status	0x01FD	Handler not analyzed
0x08	caseD_5a	*Unknown*	Status	0x0201	Handler not analyzed
0x0A	caseD_5e	*Unknown*	Status	0x0205	Via indirect jump
0x0C	caseD_6c	*Unknown*	Status	0x0213	Handler not analyzed
0x10-0x18	Input Polling	Button Input	None	0x01B7	Waits for button release
0x90-0x94	Direct Output	Data Output	Data	0x023C	Direct PORTB output
0xA2	Special Return	Status Return	Combined	-	Returns combined status
0xEE-0xFE	Serial Data	Data Reception	None	0x02DE	Serial input processing

1.3 Lookup Table Structure

Primary Table (0x80): Used when `DisplayControlFlags & 0x10 == 0` **Secondary Table (0x8B)**: Used when `DisplayControlFlags & 0x10 != 0`

```
Table Entry Format:
[command_code][dispatch_flags]
```

2. Detailed Command Analysis

2.1 Command 0x00 (Dispatch 0x00) — Display Update

Handler: `caseD_34` at address `0x01DB`

Processing Logic:

```c
if ((ButtonStateBuffer & 0x80) == 0) {
    OutputToDisplayDriver(2);        // Send data=2 with status
    OutputToDisplayDriver();         // Send status only
    caseD_a4();                     // Execute display update sequence
    CompleteCommandProcessing(4);    // Complete with response=4
} else {
    WriteToDisplayPort(DisplayControlFlags); // Direct status output
    caseD_10();                     // Execute handler 0x10
    UpdateTimersAndWait();          // Update timers and wait
}
```

Response: Status byte via `CompleteCommandProcessing(4)` **Internal State Changes**: Updates DisplayControlFlags, executes display sequences **Subroutines**: `OutputToDisplayDriver`, `caseD_a4`, `CompleteCommandProcessing`

2.2 Command 0x01 (Dispatch 0x02) — Conditional Update

Handler: `caseD_2a` at address `0x01D1`

Processing Logic:

```
c

if (((DisplayControlFlags & 0x10) == 0) && ((SerialInputData & 8) != 0)) {
    WriteToDisplayPort(DisplayControlFlags); // Output status
    caseD_10();                          // Execute handler
    UpdateTimersAndWait();               // Update and wait
} else {
    CompleteCommandProcessing(1);     // Complete with response=1
}
```

Response: Status byte via `CompleteCommandProcessing(1)` or direct output **Internal State Changes**: Conditional based on control flags and input data **Subroutines**: `WriteToDisplayPort`, `caseD_10`, `CompleteCommandProcessing`

2.3 Command 0x02 (Dispatch 0x04) — Multi-stage Update

Handler: `caseD_46` at address `0x01ED`

Processing Logic:

```
c

OutputToDisplayDriver(2);            // Send data=2 with status
caseD_10();                         // Execute handler 0x10
OutputToDisplayDriver();            // Send status only
caseD_a4();                        // Execute display sequence
WriteToDisplayPort(DisplayControlFlags); // Output final status
caseD_10();                        // Execute handler again
UpdateTimersAndWait();             // Complete operation
```

Response: Multiple outputs via `OutputToDisplayDriver` and `WriteToDisplayPort` **Internal State Changes**: Complex multi-stage display operation **Subroutines**: Multiple display and control functions

2.4 Command 0x08-0x0C (Dispatch 0x10-0x18) — Button Input Polling

Handler: Code at `0x01B7`

Processing Logic:

```c
do {
    bVar4 = PORTA;                  // Read input
    bVar4 = bVar4 & 0x3f;          // Mask to command bits
    bVar8 = bVar4 == 0;            // Check if zero
} while (!bVar8);                   // Wait until input clear
return bVar4;                       // Return final value
```

Response: Returns button state value **Internal State Changes**: None (polling only) **Usage**: Waits for button release, used in button processing

2.5 Command 0x48-0x4A (Dispatch 0x90-0x94) — Direct Data Output

Handler: Code at `0x023C`

Processing Logic:

```c
bVar4 = (bVar4 | DisplayControlFlags) & 0x7f; // Combine data + status
PORTC = PORTC & 0xfe;              // Clear strobe
PORTB = bVar4;                     // Set output data
PORTC = PORTC | 1;                 // Set strobe
// Timing delay loop
```

Response: Direct 7-bit data output via PORTB **Internal State Changes**: None (direct output only) **Usage**: Raw data transmission to CPU

2.6 Command 0x51 (Dispatch 0xA2) — Special Status Return

Handler: Inline code in switch statement

Processing Logic:

```c
CountdownTimer2 = 0x50;            // Set timer value
CommandParameter = bVar3;          // Store command
return bVar4 | *(byte *)(ushort)bVar7; // Return combined value
```

Response: Combined status and table data **Internal State Changes**: Updates timer and command storage **Usage**: Special command completion with enhanced status

2.7 Command 0x77-0x7F (Dispatch 0xEE-0xFE) — Serial Data Reception

Handler: Code at `0x02DE` (complex serial processing)

Processing Logic:

```c
// Multi-byte serial data reception loop
bVar7 = 8;                         // 8 bytes to receive
do {
    // 8-bit reception loop per byte
    do {
        PORTC = PORTC & 0xfd;      // Clear clock
        PORTC = PORTC | 2;         // Set clock
        SerialInputData = PORTA;    // Read input

        // Shift data into 3 registers
        ShiftRegister1 = ShiftRegister1 >> 1;
        ShiftRegister2 = ShiftRegister2 >> 1;
        ShiftRegister3 = ShiftRegister3 >> 1;

        // Input active-low data
        if ((PORTA & 1) == 0) ShiftRegister1 |= 0x80;
        if ((PORTA & 2) == 0) ShiftRegister2 |= 0x80;
        if ((PORTA & 4) == 0) ShiftRegister3 |= 0x80;

        BitCounter++;
    } while (BitCounter != 8);

    // Store completed bytes in buffers
    *(byte *)(bVar7 + 0x2d) = ShiftRegister1; // TimeDataBuffer
    *(byte *)(bVar7 + 0x35) = ShiftRegister2; // TimeDisplayBuffer
    *(byte *)(bVar7 + 0x3d) = ShiftRegister3; // StatusDataBuffer

    bVar7--;
} while (bVar7 != 0);

// Process received data for display
// (Complex character decoding and display logic follows)
```

Response: None (data reception only) **Internal State Changes**: Updates time, display, and status buffers **Subroutines**: `DecodeCharacterFromTable`, `ShowSystemStatusDisplay`, display functions

3. Response Generation Functions

3.1 OutputToDisplayDriver(data)

Address: `0x0238` **Function**: Combines data with status and outputs via strobe protocol

```c
PORTC = PORTC & 0xfe;              // Clear strobe
PORTB = (data | DisplayControlFlags) & 0x7f; // Combine data + status
PORTC = PORTC | 1;                 // Set strobe
// Timing delay (32 × 256 cycles)
```

3.2 WriteToDisplayPort(data)

Address: `0x023C`

Function: Direct data output via strobe protocol

```c
PORTC = PORTC & 0xfe;              // Clear strobe
PORTB = data;                      // Set data
PORTC = PORTC | 1;                 // Set strobe
// Timing delay (32 × 256 cycles)
```

3.3 CompleteCommandProcessing(response_code)

Address: `0x01C0` **Function**: Standard command completion sequence

```c
OutputToDisplayDriver();           // Send status
OutputToDisplayDriver();           // Send status again
WriteToDisplayPort(DisplayControlFlags); // Send final status
caseD_10();                       // Execute handler
UpdateTimersAndWait();            // Complete with timing
```

4. Control Flags and State Variables

4.1 DisplayControlFlags (0x14)

Usage: Primary control register for command processing

```
Bit 7: CPU communication status
Bit 6: Additional display control
Bit 5: Display enable flag (0x20)
Bit 4: Command table select (0=0x80, 1=0x8B)
Bit 3-0: Display mode and addressing
```

4.2 Command Processing Variables

- **CommandParameter (0x16)**: Current command code (PORTA & 0x3F)

- **ButtonStateBuffer (0x12)**: Current button/input state
- **SerialInputData (0x20)**: Raw input data from PORTA
- **ButtonChangeFlags (0x17)**: Detected input changes

5. Ambiguities and Unknowns

5.1 Unanalyzed Command Handlers

Commands requiring further analysis:

- **caseD_56 (0x01FD)**: Handler not decompiled
- **caseD_5a (0x0201)**: Handler not decompiled
- **caseD_6c (0x0213)**: Handler not decompiled
- **caseD_10**: Referenced frequently but not located
- **caseD_a4**: Display sequence handler not analyzed

5.2 Lookup Table Contents

Unknown table entries:

- Complete contents of command_lookup_table_primary (0x80)
- Complete contents of command_lookup_table_secondary (0x8B)
- Mapping between command codes and dispatch values
- Table termination and bounds checking

5.3 Response Format Details

Partially understood:

- Exact bit meanings in status responses
- CPU interpretation of strobe timing
- Error or fault response codes
- Response data encoding for specific commands

5.4 Serial Data Protocol

Unknown aspects:

- Complete data packet structure (192 bits total)
- Synchronization and framing
- Error detection/correction
- Timing requirements and tolerances

6. Implementation Notes

6.1 Command Dispatch Mechanism

The firmware uses a sophisticated two-level dispatch system:

1. **Command extraction**: `PORTA & 0x3F` gives 6-bit command (0-63)
2. **Table selection**: DisplayControlFlags bit 4 selects primary/secondary table
3. **Lookup**: `table[command+1] << 1` gives dispatch code (even numbers 0x00-0xFE)
4. **Switch execution**: Massive switch statement with 128+ cases

6.2 Response Timing

All responses use identical strobe protocol:

- Clear PORTC bit 0 (setup)
- Set PORTB data (valid data)
- Set PORTC bit 0 (strobe)
- Fixed delay loop (8192 cycles typical)

6.3 State Machine Architecture

The firmware implements a complex state machine where:

- Commands can modify DisplayControlFlags
- Flag changes affect subsequent command interpretation
- Multiple response stages possible per command
- Timer coordination maintains CPU synchronization

This analysis covers all identifiable commands in the ProcessData switch statement. Further investigation required for complete handler decompilation and lookup table contents.