# P2 Single-Step Debugger User's Manual

**Version**: 1.0  
**Target SPIN2 Version**: v51+  
**Last Updated**: August 2025

## Introduction

The P2 Single-Step Debugger provides sophisticated PASM-level debugging capabilities integrated directly into the SPIN2 development environment. This manual covers practical workflows for debugging P2 assembly code, multi-COG applications, and real-time systems.

## Quick Start

### Basic Debugging Setup

PUB main() | result  
 DEBUG("Program started") ' Set initial breakpoint  
   
 result := calculate(42)  
 DEBUG("Result: ", DEC(result)) ' Monitor results  
   
 DEBUG("Program complete")  
  
PRI calculate(value) : result  
 DEBUG("Entering calculate with: ", DEC(value))  
   
 ' PASM section with debugging  
 org  
 debug ' PASM-level breakpoint  
 mov result, value  
 add result, #10  
 debug ' Another breakpoint  
 end

### Essential Commands

* **Step Into**: Execute next instruction and stop
* **Step Over**: Execute next instruction, stepping over calls
* **Continue**: Resume until next breakpoint
* **Break**: Interrupt execution at current instruction

## Core Debugging Features

### PASM-Level Debugging

Set breakpoints in assembly code using the debug instruction:

org  
 mov a, b  
 debug ' Execution pauses here  
 add a, #1  
 debug ' And here  
end

**Key Capabilities**: - Instruction-level stepping through PASM code - Register and memory inspection at each step - Multiple breakpoints with independent control - Integration with SPIN2 high-level debugging

### Multi-COG Debugging

Each COG can be debugged independently:

PUB start\_system()  
 DEBUG("Starting sensor COG")  
 cognew(sensor\_cog(), @sensor\_stack)  
   
 DEBUG("Starting display COG")   
 cognew(display\_cog(), @display\_stack)  
   
 DEBUG("Main coordination loop")  
 coordination\_loop()  
  
PRI sensor\_cog()  
 DEBUG("Sensor COG operational")  
 repeat  
 DEBUG("Reading sensors")  
 ' Sensor code with independent breakpoints  
  
PRI display\_cog()  
 DEBUG("Display COG operational")   
 repeat  
 DEBUG("Updating display")  
 ' Display code with independent breakpoints

**Multi-COG Features**: - Independent breakpoints per COG - Parallel debugging sessions - Non-destructive COG state monitoring - Coordinated breakpoints across COGs

## DEBUG SCOPE Displays

### Real-Time Signal Monitoring

PUB monitor\_system() | temperature, voltage  
  
 ' Setup scope for real-time plotting  
 DEBUG(`SCOPE\_XY samples temperature voltage $FF0000)  
 DEBUG(`SCOPE TRIGGER temperature AUTO 100)  
   
 repeat  
 temperature := read\_sensor(TEMP\_PIN)  
 voltage := read\_sensor(VOLT\_PIN)  
   
 ' Update scope display  
 DEBUG(`SCOPE\_XY\_UPDATE temperature voltage)  
   
 waitms(50)

**SCOPE Capabilities**: - **Real-time waveform display**: Monitor changing values as plots - **Multi-channel support**: Display multiple signals simultaneously  
- **Auto-scaling**: Automatic range adjustment for optimal viewing - **Trigger system**: Auto-trigger on signal conditions - **Color coding**: Different colors for different signals

### Trigger System

' Auto-trigger when temperature exceeds threshold  
DEBUG(`SCOPE TRIGGER temperature AUTO {offset})  
  
' Manual trigger control  
DEBUG(`SCOPE TRIGGER MANUAL)

**Trigger Features** (v41+): - Automatic triggering on signal conditions - Channel selection for trigger source - Offset control within display window - Edge detection for rising/falling triggers

## Advanced Debugging Workflows

### Performance Analysis

PUB timing\_critical\_loop() | start\_time, elapsed  
  
 DEBUG("Starting performance analysis")  
   
 start\_time := CNT  
   
 repeat 1000  
 ' Critical code section  
 org  
 debug ' Time measurement point  
 ' Assembly operations  
 debug ' Another measurement point  
 end  
   
 elapsed := CNT - start\_time  
 DEBUG("Total cycles: ", DEC(elapsed))  
 DEBUG("Per-iteration: ", DEC(elapsed/1000))

### Hardware Interface Debugging

PUB debug\_smart\_pin(pin\_num) | config, result  
  
 DEBUG("Configuring Smart Pin ", DEC(pin\_num))  
   
 ' Configure Smart Pin with debugging  
 config := P\_ADC | P\_ADC\_100X  
 WRPIN(config, pin\_num)  
 WXPIN(1000, pin\_num) ' Set timing  
 WYPIN(0, pin\_num) ' Start conversion  
 DIRH(pin\_num) ' Enable pin  
   
 DEBUG("Smart Pin configured: $", HEX(config))  
   
 repeat  
 result := RDPIN(pin\_num)  
 DEBUG("ADC Reading: ", DEC(result))  
 waitms(100)

### Error Handling and Diagnostics

PUB safe\_operation() | status, error\_code  
  
 DEBUG("Beginning safe operation")  
   
 status := attempt\_operation()  
   
 if status < 0  
 error\_code := get\_error\_details()  
 DEBUG("Error occurred: ", DEC(error\_code))  
   
 ' Debug error state  
 org  
 debug ' Examine error condition  
 ' Error recovery code  
 debug ' Verify recovery  
 end  
 else  
 DEBUG("Operation successful: ", DEC(status))

## Integration and Setup

### Development Environment Integration

**Clock Frequency Adaptation** (v36+): - Debugger automatically adapts to runtime clock changes - Uses P63 for frequency storage and tracking - Maintains accuracy across CLKSET operations - Supports all P2 clock modes seamlessly

**Serial Communication Setup**:

' Control debug baud rate  
DEBUG\_BAUD = 2\_000\_000 ' 2M baud for fast transfer  
  
' Control debug window behavior  
DEBUG\_WINDOWS\_OFF = true ' Disable automatic window opening

### Flash Programming Debug Support

**Production Debugging**: - Flash-programmed code retains debug capability - Command-line DEBUG-only mode available - Debug data preserved in flash memory - Post-programming debug sessions supported

## Troubleshooting

### Common Issues

**Serial Communication Problems**: - Check pull-up resistor compatibility (addressed in v35v) - Verify baud rate configuration - Ensure proper hardware connections

**Performance Issues**: - Use conditional debugging with \_\_DEBUG\_\_ symbol - Minimize scope display update rates - Implement selective breakpoint activation

**Multi-COG Coordination**: - Verify independent COG debug setup - Check for resource conflicts between COGs - Use coordinated breakpoints for synchronization issues

# Appendix A: Version History and Evolution

## SPIN2 Debugger Evolution Timeline

### v35u (August 26, 2022) - **PASM-Level Debugger Introduction**

**Major Milestone**: First introduction of PASM-level debugging capabilities

**New Features**: - **PASM-level debugger**: Step through assembly instructions line by line - **Breakpoint support**: Set breakpoints in PASM code using debug instruction  
- **Single-stepping functionality**: Step Into, Step Over, Continue, Break commands - **Basic DEBUG output**: Text-based debugging output to terminal

**Impact**: Revolutionized P2 development by enabling instruction-level debugging for the first time.

### v35v (September 11, 2022) - **Hardware Compatibility Enhancement**

**Focus**: Early P2 Edge module support

**Improvements**: - **Serial pull-up compatibility**: Enhanced support for early P2 Edge modules without built-in serial pull-ups - **Communication reliability**: Improved serial communication stability - **Hardware adaptation**: Better support for different P2 hardware configurations

**Impact**: Made debugging accessible across all P2 hardware variants, including early development boards.

### v35g (Date TBD) - **Stability and Reliability**

**Focus**: Bug fixes and mathematical robustness

**Fixes**: - **Floating-point exception handling**: Fixed line-clipping routine floating-point exceptions - **Memory access protection**: Prevention of memory-access violations - **Scope display stability**: More robust scope display operations - **Mathematical operations**: Improved stability in debug mathematical calculations

**Impact**: Significantly improved debugger reliability for production use.

### v36 (September 18, 2022) - **Clock Management Revolution**

**Major Enhancement**: Dynamic clock frequency adaptation

**New Features**: - **Automatic clock frequency adaptation**: Debugger automatically tracks runtime clock changes - **Serial receive pin utilization**: Uses P63 for frequency storage and coordination - **CLKSET compatibility**: Maintains debugging accuracy across CLKSET operations - **Clock mode support**: Full support for all P2 clock modes and frequencies - **PASM-only program support**: Automatic clock-setter prepended to PASM-only programs - **ASMCLK instruction obsolescence**: No longer requires ASMCLK instruction at program start

**Impact**: Eliminated clock-related debugging issues and harmonized debugger operation across all P2 clock configurations.

## Version Compatibility Matrix

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Feature | v35u | v35v | v35g | v36 | v41 | v51+ |
| PASM Debugging | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| Serial Pull-up Support | ❌ | ✅ | ✅ | ✅ | ✅ | ✅ |
| Exception Handling | ❌ | ❌ | ✅ | ✅ | ✅ | ✅ |
| Auto Clock Adaptation | ❌ | ❌ | ❌ | ✅ | ✅ | ✅ |
| Auto SCOPE Triggering | ❌ | ❌ | ❌ | ❌ | ✅ | ✅ |
| Flash Debug Support | ❌ | ❌ | ❌ | ✅ | ✅ | ✅ |
| Multi-COG Advanced | ❌ | ❌ | ❌ | ✅ | ✅ | ✅ |
| Production Ready | ❌ | ❌ | ⚠️ | ✅ | ✅ | ✅ |

**Legend**: - ✅ Full support - ⚠️ Limited/partial support  
- ❌ Not available

**Minimum Version Requirements**: - **For Basic PASM Debugging**: v35u or later - **For Hardware Compatibility**: v35v or later  
- **For Production Use**: v36 or later - **For Advanced Features**: v41 or later - **For Complete Feature Set**: v51 or later