

HIL-OS V9.5

Complete Reference Manual & Maker Guide

Hardware-In-The-Loop Operating System for Arduino

Sherif Rashwan

Version 9.5

August 2026

HIL-OS V9.5 -- Complete Reference Manual & Maker Guide

Hardware-In-The-Loop Operating System for Arduino

Version 9.5 -- Sherif Rashwa

This document is the definitive reference for every aspect of the HIL-OS toolchain. It is written for engineers, makers, students, and anyone who wants to use HIL LISP to control real hardware or simulate circuits. This manual covers everything from basic LISP syntax to advanced topics like PID control, I2C communication, and building complete embedded projects

Table of Contents

- [Introduction](#1-introduction)
 - [Getting Started](#2-getting-started)
 - [Arduino Hardware Reference](#3-arduino-hardware-reference)
 - [HIL LISP Language Reference](#4-hil-lisp-language-reference)
 - [Sensors & Input Devices](#5-sensors--input-devices)
 - [Actuators & Output Devices](#6-actuators--output-devices)
 - [Communication Protocols](#7-communication-protocols)
 - [CLI Command Reference](#8-cli-command-reference)
 - [Firmware & Deployment](#9-firmware--deployment)
 - [CRUMB Circuit Simulation](#10-crumb-circuit-simulation)
 - [Serial Wire Protocol](#11-serial-wire-protocol)
 - [JSON IPC & Node.js API](#12-json-ipc--nodejs-api)
 - [VS Code Extension](#13-vs-code-extension)
 - [Transpiler Deep Dive](#14-transpiler-deep-dive)
 - [Advanced Topics](#15-advanced-topics)
 - [Complete Projects](#16-complete-projects)
 - [Troubleshooting & FAQ](#17-troubleshooting--faq)
 - [Appendix](#18-appendix)
-

1. Introduction

What is HIL-OS?

HIL-OS is a complete hardware-in-the-loop toolchain for Arduino-based embedded systems. It provides a custom LISP programming language that lets you write scripts to control real hardware -- read sensors, drive motors, light LEDs, play sounds, run PID controllers, and more -- all from a simple, expressive scripting language

The key insight behind HIL-OS is that embedded systems programming should be fast, interactive, and forgiving. Traditional Arduino development requires writing C/C++ code, compiling it, flashing it to the board, and then

discovering that your LED is on the wrong pin. HIL-OS lets you write a script, run it immediately, see the results in real time, and change anything on the fly -- all without reflashing the board

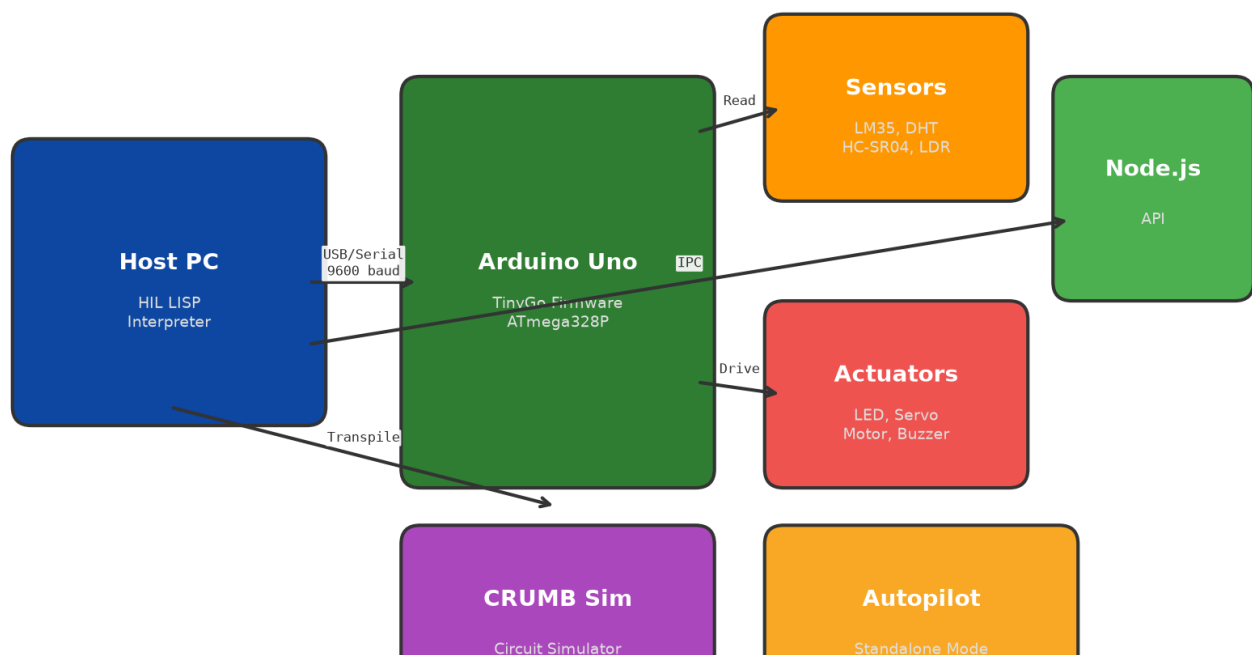
How It Works

L-OS has three execution mode

1. Host-Controlled Mode (Live REPL)

host computer runs the LISP interpreter and communicates with an Arduino over a serial connection. The Arduino firmware acts as a hardware abstraction layer, executing commands received from the host. This is ideal for rapid prototyping, data logging, and situations where you need the power of a host computer alongside real hardware

HIL-OS System Architecture



HIL-OS System Architecture

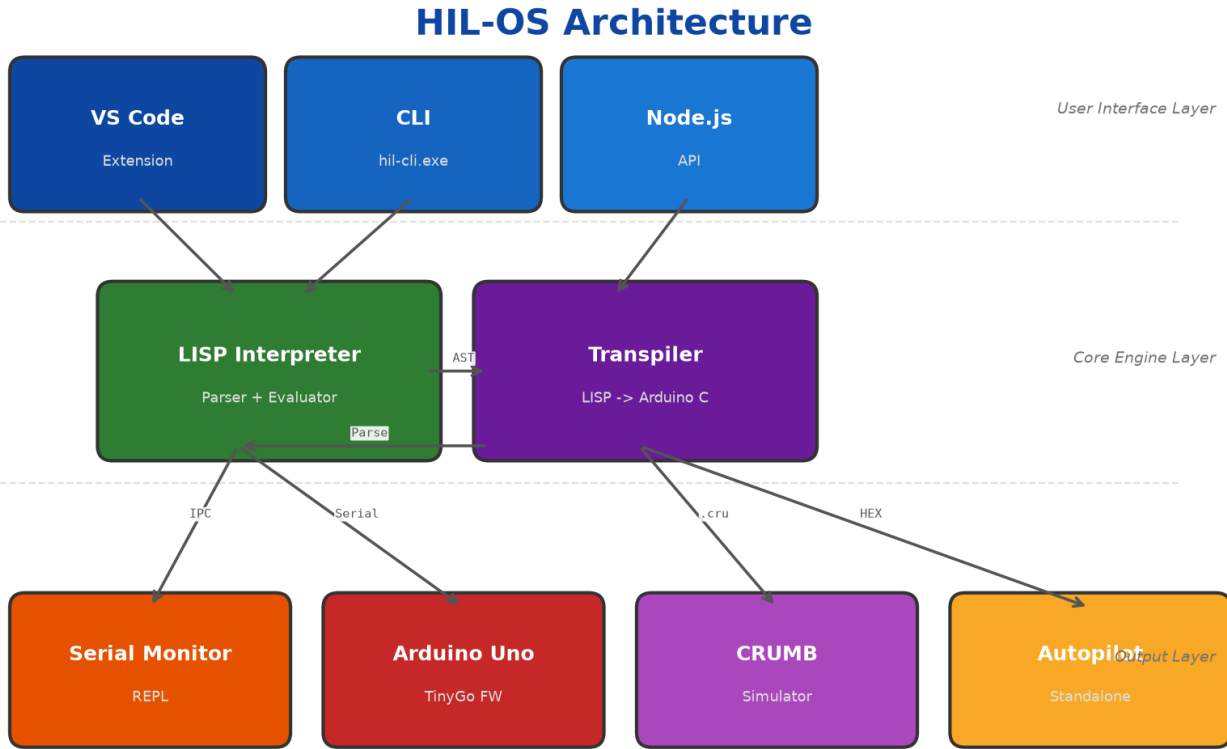
Figure 1: HIL-OS System Architecture -- showing the three execution modes and their data flow

2. Standalone Autopilot Mode

The host compiles a LISP script into a sequence of native hardware commands that are stored in the Arduino's RAM. The Arduino then runs these commands autonomously, without any host connection. This is ideal for deployed systems that must operate independently -- once the autopilot is programmed, you can disconnect the USB cable and run the Arduino from a battery

3. CRUMB Circuit Simulation Mode

The LISP script is transpiled into Arduino C code and injected into a CRUMB circuit simulator. You can test and debug your logic in a virtual circuit before touching any real hardware. The generated C code uses standard Arduino API calls and also works on real hardware if compiled with the Arduino IDE or TinyG



HIL-OS Architecture

figure 2: HIL-OS layered architecture -- User Interface, Core Engine, and Output layers

Why HIL-OS?

Feature	Traditional Arduino	HIL-OS
Language	C/C++	LISP
Edit-compile-flash cycle	10-30 seconds	Instant
Interactive debugging	Limited	Real-time REPL
Circuit simulation	Separate tools	Built-in (CRUMB)
Sensor libraries	Manual setup	One-line commands
Code reuse	Copy-paste	Macros & packages
Deploy to hardware	Flash once	Autopilot mode
Host integration	None	Node.js API

Who is this for?

- **Makers and hobbyists** who want a higher-level way to program Arduino projects
- **Students** learning about embedded systems, sensors, and control theory
- **Engineers** prototyping cyber-physical systems who want rapid iteration
- **Educators** teaching programming, robotics, or control systems
- **Anyone** who wants to bridge software and hardware without writing C/C++

- **Developers** who want to integrate hardware into software systems via Node.js

2. Getting Started

System Requirements

Component	Requirement	Notes
Operating System	Windows 10+, macOS 12+, Linux (Ubuntu 22.04+)	Tested on all three
Go	1.26 or later	For building the toolchain
Node.js	18 or later	Optional, for the Node.js wrapper
Arduino board	Uno, Nano, Mega, or ATmega328P/ATmega2560	Recommended for beginners
USB cable	Type A to Type B (Uno) or Micro-USB (Nano)	For connecting the Arduino
VS Code	Latest version	Optional, for the IDE extension

Step 1: Install Go

you don't have Go installed, download it from <https://go.dev/dl/>. Verify your installatio

```
go version
# Should print: go version go1.26.x windows/amd64
```

Step 2: Build the Toolchain

vigate to the HIL-OS repository and build both component

```
# Build the host monitor (the LISP interpreter)
cd go-serial-monitor
go build -o hil-orchestrator.exe .

# Build the CLI (project manager and toolchain)
cd ../hil-cli
go build -o hil-cli.exe .
```

Step 3: Verify Everything

n the diagnostic command to check your installatio

```
.\hil-cli.exe doctor
```

is will check: - Go toolchain availability - Node.js installation (optional) - Monitor executable - Node wrapper (optional) - CRUMB simulator (optional) - Transpiler functionality - CRUMB generator functionali

Step 4: Find Your Serial Port

e the interactive port scanner to find your Arduin

```
.\hil-cli.exe ports
```

is scans all available serial ports and presents an interactive menu. On Windows, Arduino boards typically appear a

COM3

COM4

or higher. On Linux, they appear a

/dev/ttyACM0

r similar. On macOS, they appear a

/dev/cu.usbmodem*

/dev/cu.usbserial*

Step 5: Create Your First Project

```
# Create a new project with the blink template
.\hil-cli.exe init my-first-project --template blink

# Or create a project with the full development environment
.\hil-cli.exe init my-sensor-project --template basic
```

Step 6: Run Your First Script

```
# Run the blink script
.\hil-cli.exe run src/main.hil

# Or run it headlessly (no hardware needed)
.\hil-cli.exe simulate src/main.hil
```

Step 7: Create Your First Project Manually

create a new file called

hello.hil

```
; Hello World in HIL LISP
; Blinks the built-in LED on pin 13

(LOG "Hello, HIL-OS!")
(LOG "Starting LED blink...")

(SET pin 13)
(SET count 0)

(WHILE (< count 5)
  (DO
    (WRITE D pin 1)
    (LOG "LED ON")
    (DELAY 500)
    (WRITE D pin 0)
    (LOG "LED OFF")
    (DELAY 500)
    (SET count (+ count 1))))

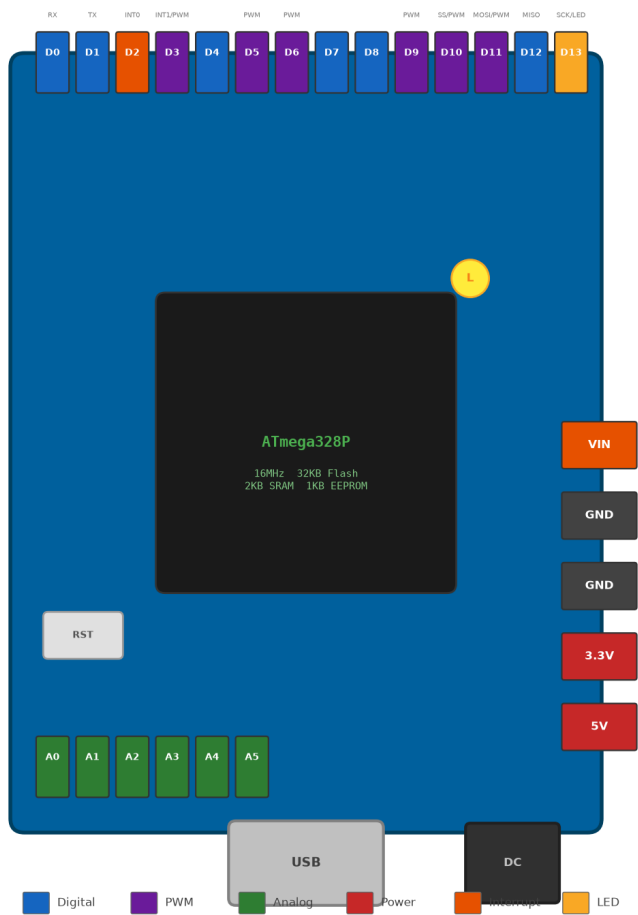
(LOG "Done! Blink complete.")
```

n i

```
.\hil-cli.exe run hello.hil
```

3. Arduino Hardware Reference

Arduino Uno R3 Pinout



Arduino Uno R3 Pinout

figure 3: Arduino Uno R3 complete pinout diagram showing digital, analog, PWM, and power pins

Pin Functions

Pin	Type	Functions	Max Current	Notes
D0	Digital	RX (UART)	20mA	Used for serial communication
D1	Digital	TX (UART)	20mA	Used for serial communication
D2	Digital	INT0, PWM	20mA	External interrupt 0
D3	Digital	INT1, PWM	20mA	External interrupt 1
D4	Digital	--	20mA	General purpose
D5	Digital	PWM	20mA	PWM capable
D6	Digital	PWM	20mA	PWM capable
D7	Digital	--	20mA	General purpose
D8	Digital	--	20mA	General purpose
D9	Digital	PWM	20mA	PWM capable, common servo pin
D10	Digital	PWM, SS	20mA	SPI slave select
D11	Digital	PWM, MOSI	20mA	SPI MOSI, passive buzzer in CRUMB

D12	Digital	MISO	20mA	SPI MISO
D13	Digital	SCK, LED	20mA	Built-in LED connected here
A0	Analog	ADC0	20mA	Analog input (0-1023)
A1	Analog	ADC1	20mA	Analog input
A2	Analog	ADC2	20mA	Analog input
A3	Analog	ADC3	20mA	Analog input
A4	Analog	SDA (I2C)	20mA	I2C data line
A5	Analog	SCL (I2C)	20mA	I2C clock line
5V	Power	--	500mA total	Regulated 5V output
3.3V	Power	--	50mA	Regulated 3.3V output
GND	Power	--	--	Ground
VIN	Power	--	--	External voltage input (7-12V)

Arduino Nano Pinout



Arduino Nano Pinout

Figure 4: Arduino Nano pinout -- compact form factor with the same ATmega328P chip as the Uno. The Arduino Mega 2560 has more pins and more memory (256KB flash, 8KB SRAM). It uses the ATmega2560 chip and provides 54 digital I/O pins, 16 analog inputs, and 15 PWM outputs. The pinout follows the same naming convention as the Uno but with extended pin ranges (D0-D53, A0-A15).

Electrical Characteristics

Parameter	Value	Notes
Operating voltage	5V	USB or external power
Input voltage	7-12V (recommended)	6-20V (absolute limits)
Digital I/O pins	14 (Uno) / 54 (Mega)	6 PWM capable (Uno) / 15 (Mega)
Analog input pins	6 (Uno) / 16 (Mega)	10-bit ADC (0-1023)
DC current per I/O	20mA	Absolute maximum
DC current for 5V pin	500mA	Total from all sources
Flash memory	32KB (Uno) / 256KB (Mega)	0.5KB used by bootloader
SRAM	2KB (Uno) / 8KB (Mega)	For variables and stack
EEPROM	1KB (Uno) / 4KB (Mega)	Persistent storage
Clock speed	16MHz	Crystal oscillator

Power Supply Options

USB Power (5V, 500mA):

he simplest way to power your Arduino. Just plug in the USB cable. This provides 5V at up to 500mA, which is enough for the board itself and most sensors/LED

External Power (7-12V via VIN):

se a 9V battery, wall adapter, or solar panel. The onboard voltage regulator steps it down to 5V. Do not exceed 12V -- the regulator will overhea

Direct 5V via 5V Pin:

f you have a regulated 5V supply, you can connect it directly to the 5V pin. This bypasses the voltage regulator. Do not use this if you are also using USB power -- choose one or the othe

Wiring Best Practices

- ****Always use current-limiting resistors with LEDs.**** A typical LED needs a 220-330 ohm resistor. Without one, you will burn out the LED or damage the Arduino pin.
- ****Never connect more than 20mA to a single digital pin.**** If you need to drive a relay, motor, or other high-current device, use a transistor or relay module.
- ****Use pull-up or pull-down resistors for buttons.**** Without one, the pin will "float" and give random readings. HIL-OS provides software pull-ups via the firmware.
- ****Keep wires short and organized.**** Long wires act as antennas and pick up electrical noise. Use a breadboard for prototyping and solder for permanent connections.
- ****Separate analog and digital signals.**** Run analog sensor wires away from digital switching wires (like servo or motor wires) to reduce noise.
- ****Use a common ground.**** All devices must share the same ground reference. Connect all GND pins together.

4. HIL LISP Language Reference

Overview

L LISP is a minimalist, integer-only LISP dialect designed for embedded hardware control. Every value in HIL LISP is an integer. There are no floating-point numbers, no strings (except as literal constants i

LOG

n

INCLUDE

and no complex data structures beyond nested list

is simplicity is by design. Embedded systems have limited resources -- the ATmega328P has only 2KB of RAM and runs at 16MHz. By keeping the language simple, HIL-OS ensures fast parsing, low memory usage, and deterministic behavior

Syntax Rules

- **S-expressions:** Everything is a parenthesized expression: `(operator arg1 arg2 ...)`
- **Case-sensitive:** `SET` and `set` are different. Use uppercase for operators.
- **Integers only:** All values are integers. No floats, no strings (except in `LOG`).
- **No type errors:** If you pass a string where a number is expected, the result is 0.
- **Semicolons for comments:** `;` starts a comment that runs to end of line.

Comments

Comments start with a semicolon

;

and run to the end of the line. Hash

#

is also recognized as a comment delimiter for compatibility with some tool

```
; This is a comment that explains what the next line does
(WRITE D 13 1) ; Turn on the LED on pin 13

# This is also a comment
# Multiple comment lines are fine
```

Variables

Variables are created and assigned using

SET

Once set, they can be referenced by their bare name. Variable names are case-sensitive. There is no need to declare variables before using them -

SET

creates them automatically

```
(SET pin 13)           ; Create variable 'pin' with value 13
(SET brightness 255)   ; Create variable 'brightness'
(SET sensorPin A0)     ; Create variable 'sensorPin' with value of A0
(WRITE D pin 1)        ; Use 'pin' -- writes HIGH to pin 13
(WRITE A pin brightness) ; Use both variables
```

Variables propagate to parent scopes. If a variable is defined inside

WHILE

loop and also exists in the outer scope, then

SET

updates the outer variable. This is important for loop counters and state flag

```
(SET x 0)
(WHILE (< x 10)
  (DO
    (SET x (+ x 1)))) ; Updates the outer 'x'
(LOG "Final x=" x)    ; Prints "Final x=10"
```

Variables have no type -- they can hold any integer value at any time. You can store a pin number in a variable one moment and an analog reading in the same variable the next

```
(SET myVar 13)         ; myVar holds pin number 13
(WRITE D myVar 1)      ; Use it as a pin number
(SET myVar (READ A 0)) ; Now myVar holds an analog reading (0-1023)
```

Arithmetic Operators

Integer arithmetic is integer-based. Division truncates toward zero (e.g.,

$7/2 = 3$

not 3.5). Division by zero returns 0 as a safety feature -- it does not crash the program

```
(+ 3 4)      ; Addition: 3 + 4 = 7
(- 10 3)     ; Subtraction: 10 - 3 = 7
(* 6 7)      ; Multiplication: 6 * 7 = 42
(/ 10 3)     ; Integer division: 10 / 3 = 3
(/ 10 0)     ; Safe division: 10 / 0 = 0 (no crash)
```

You can nest arithmetic expressions to build complex calculations

```
; Calculate PWM value from analog reading
; Formula: output = (input * 255) / 1023
(SET pwmVal (/ (* (READ A 0) 255) 1023))
(WRITE A 9 pwmVal)
```

Comparison Operators

Comparisons return

1

or true and

0

or false. There is no boolean type -- integers serve as booleans (0 is false, anything else is true)

```
(== 5 5)    ; Equal: 1 (true)
(!= 5 3)    ; Not equal: 1 (true)
(> 10 5)    ; Greater than: 1 (true)
(< 3 7)     ; Less than: 1 (true)
(>= 5 5)    ; Greater or equal: 1 (true)
(<= 3 7)    ; Less or equal: 1 (true)
```

Comparison results can be used directly in

IF

condition

```
(IF (> (READ A 0) 512)
  (WRITE D 13 1) ; If analog reading > 512, turn LED on
  (WRITE D 13 0)) ; Otherwise, turn LED off
```

Logical Operators

There are no explicit

AND

OR

operators. Instead, use arithmetic

```
; AND: multiply two conditions
(IF (* (> temp 25) (< temp 35))
  (LOG "Temperature is in range"))

; OR: add two conditions (use min to clamp)
(IF (or (> temp 35) (< temp 10))
  (LOG "Temperature is out of range"))
```

Note: HIL LISP does not have built-in

AND

OR

operators. The multiplication trick works because

$1 * 1 = 1$

true) and

$1 * 0 = 0$

false). For OR, you can check if either condition is non-zero

Flow Control

DO -- Sequence of Operations

```
(DO expr1 expr2 ...)
```

evaluates each form in order and returns the last value. Use it to group multiple operations together

```
(DO
  (SET a 1)
  (SET b 2)
  (+ a b)) ; Returns 3
```

DO

s essential insid

WHILE

n

IF

locks when you need to execute multiple statements. Without it, only the first expression would be evaluate

```
; WRONG -- only the first expression is in the WHILE
(WHILE (< i 5)
  (SET i (+ i 1)))

; CORRECT -- use DO to group multiple expressions
(WHILE (< i 5)
  (DO
    (LOG "i=" i)
    (SET i (+ i 1))))
```

IF -- Conditional Branching

(IF condition then-expr else-expr)

ranches on a nonzero condition. If the condition is nonzero (true)

then-expr

s evaluated. If it is zero (false)

else-expr

s evaluate

```
(IF (> temp 30)
  (WRITE D 13 1) ; Turn on LED if hot
  (WRITE D 13 0)) ; Otherwise turn it off
```

else

ranch is optiona

```
(IF (> temp 30)
  (WRITE D 13 1)) ; Only turn on, don't turn off
```

u can nes

IF

tatements to creat

else-if

hain

```
(IF (> temp 35)
  (LOG "HOT!")
  (IF (> temp 25)
    (LOG "Warm")
    (IF (> temp 15)
      (LOG "Cool")
      (LOG "Cold")))))
```

WHILE -- Loop

(WHILE condition body...)

oops while the condition is nonzero. The condition is checked before each iteration, so if it starts as false, the loop body never execute

Critical rule:

ver

WHILE

oop MUST contain

DELAY

r a condition that eventually becomes false. Without this, the loop runs as fast as the hardware allows and appears to hang. The Arduino will stop responding to serial commands because it is stuck in the loo

```
(SET i 0)
(WHILE (< i 5)
  (DO
    (LOG "i=" i)
    (DELAY 1000) ; Pause for 1 second -- required!
    (SET i (+ i 1))))
```

r infinite loops (like a main program loop), us

(WHILE (== 1 1) ...)

nd include

DELAY

f at least 10m

```
; Main program loop -- runs forever
(WHILE (== 1 1)
  (DO
    (SET temp (LM35 A0))
    (LOG "Temperature: " temp)
    (IF (> temp 30)
      (WRITE D 13 1)
      (WRITE D 13 0))
    (DELAY 1000))) ; Check every second
```

AWAIT -- Polling with Timeout

(AWAIT condition timeout-ms)

olls the condition every 50 milliseconds up to the specified timeout. Return

1

f the condition became true

0

n timeout. This is useful for waiting for a button press or sensor readin

```
; Wait for button press on pin 2, timeout after 5 seconds
(SET pressed (AWAIT (== (READ D 2) 1) 5000))
(IF (== pressed 1)
  (LOG "Button was pressed!")
  (LOG "Timeout -- no button press"))
```

EXIT -- Halt Execution

(EXIT)

tops all script execution immediately. The Arduino stops processing commands. You must send a new script to restart

```
(IF (== (READ D 2) 1)
  (DO
    (LOG "Emergency stop!")
    (EXIT)))
```

Delay

(DELAY ms)

locks for the given number of milliseconds. The Arduino cannot do anything else during a delay -- it cannot read sensors, respond to serial commands, or update outputs. Keep delays short (under 1000ms) and use them sparingly in loop

```
(WRITE D 13 1) ; LED on
(DELAY 500) ; Wait 500ms
(WRITE D 13 0) ; LED off
(DELAY 500) ; Wait 500ms
```

r delays longer than 1 second, consider breaking them into smaller chunks so the system remains responsiv

```
; Wait 5 seconds but remain responsive
(SET i 0)
(WHILE (< i 5)
  (DO
    (DELAY 1000)
    (SET i (+ i 1))))
```

Logging

(LOG arg1 arg2 ...)

rints arguments to the serial monitor. Strings are printed verbatim; numbers are printed as integers. Multiple arguments are printed on the same line without spaces (unless you include spaces in strings

```
(LOG "Hello, world!")
(LOG "Temperature: " temp " C")
(LOG "x=" x " y=" y " z=" z)
```


LOG

ommand is your primary debugging tool. Use it liberally during development to understand what your script is doin

```
(SET reading (READ A 0))
(LOG "Raw ADC reading: " reading)
(SET voltage (/ (* reading 5000) 1023))
(LOG "Voltage: " voltage " mV")
(SET temp (/ voltage 10))
(LOG "Temperature: " temp " C")
```

Math Utilities

MAP -- Range Remapping

```
(MAP value inMin inMax outMin outMax)
```

emaps a value from one range to another using integer math. This is one of the most useful functions in HIL-OS, because Arduino analog readings are 0-1023 but most actuators need 0-25

```
; Map analog reading (0-1023) to PWM (0-255)
(SET pwmVal (MAP (READ A 0) 0 1023 0 255))
(WRITE A 9 pwmVal)

; Map temperature (0-50C) to servo angle (0-180)
(SET angle (MAP temp 0 50 0 180))
(V-SERVO 9 angle)
```

the input range is degenerate (inMax == inMin)

MAP

eturn

outMin

s a safe default. This prevents division-by-zero crashe

CONSTRAIN -- Clamping

```
(CONSTRAIN value min max)
```

lamps a value to a range. If the value is belo

min

it return

min

If abov

max

it return

max

This is essential for protecting hardware from out-of-range value

```
(SET safe (CONSTRAIN (READ A 0) 0 255))
(WRITE A 9 safe)

; Prevent servo from going beyond physical limits
(SET angle (CONSTRAIN angle 0 180))
(V-SERVO 9 angle)
```

RANDOM -- Random Numbers

```
(RANDOM min max)
```

returns a random integer in the inclusive range [min, max]. Use it for games, randomized patterns, or testing

```
; Random LED blink interval
(SET delay (RANDOM 100 1000))
(DELAY delay)

; Random PWM output
(SET r (RANDOM 0 255))
(WRITE A 9 r)
```

Macros (DEFUN)

```
(DEFUN name (arg1 arg2 ...) body...)
```

defines a reusable macro. Arguments are call-by-name -- they are substituted into the body on each call. Macros return integer values (the value of the last expression in the body)

```
(DEFUN BLINK-N (pin count delay-ms)
  (BLINK pin count delay-ms))

; Call the macro
(BLINK-N 13 5 500) ; Blink pin 13, 5 times, 500ms each
```

Macros can call other macros and can be recursive

```
(DEFUN FACTORIAL (n)
  (IF (<= n 1)
    1
    (* n (FACTORIAL (- n 1)))))

(SET result (FACTORIAL 5))
(LOG "5! = " result) ; Prints "5! = 120"
```

Important:

Macros are expanded at parse time, not at runtime. This means: - Arguments are text-substituted, not evaluated - You cannot pass expressions as arguments -- they are used literally - Recursive macros work but can cause stack overflow on deep recursion

practical macro example -- a function to read a sensor with averaging

```

; Read analog pin 5 times and return average
(DEFUN READ-AVG (pin)
  (DO
    (SET total 0)
    (SET i 0)
    (WHILE (< i 5)
      (DO
        (SET total (+ total (READ A pin)))
        (SET i (+ i 1))
        (DELAY 10)))
      (/ total 5)))

; Use it
(SET avgTemp (READ-AVG A0))
(LOG "Average reading: " avgTemp)

```

Includes

```
(INCLUDE "path")
```

loads and evaluates another HIL file. The path is relative to the current file. Circular includes are detected and rejected to prevent infinite loop

```

(INCLUDE "lib/cyber.hil")    ; Load the cyber-physical library
(INCLUDE "lib/sensors.hil") ; Load sensor utilities

```

e includes to organize your project into modules. Put reusable macros i

lib/

iles and include them from your main scrip

```

my-project/
  src/
    main.hil      ; Main script
  lib/
    cyber.hil     ; Cyber-physical macros
    sensors.hil   ; Sensor reading utilities

```

Structs

```
(STRUCT name (field1 field2 ...))
```

declares a struct layout

```
(NEW var StructName val1 val2 ...)
```

creates an instance. Structs are useful for grouping related configuration value

```

(STRUCT Motor (pin speed direction))
(NEW myMotor Motor 9 200 1)

; Access fields as dot-notation variables
(SET motorPin myMotor.pin)
(SET motorSpeed myMotor.speed)
(WRITE D myMotor.direction 1)
(WRITE A myMotor.pin myMotor.speed)

```

Sensor Tuple Access

me operations return tuples (nested lists). Us

FIRST

n

SECOND

o access element

```
; LM35 returns temperature as (temp)
(SET reading (LM35 A0))
(SET temp (FIRST reading))
(LOG "Temperature: " temp)

; DHT returns temperature and humidity as (temp humidity)
(SET reading (DHT-READ 4 22))
(SET temp (FIRST reading))
(SET humidity (SECOND reading))
(LOG "Temp: " temp " Humidity: " humidity)
```

Batching

(BATCH-BEGIN)

uppresses per-command ACK waits, allowing commands to stream at maximum speed

(BATCH-END)

ushes the batch. This is useful when sending many commands quickl

```
(BATCH-BEGIN)
(WRITE D 13 1)
(WRITE D 12 1)
(WRITE D 11 1)
(WRITE D 10 1)
(BATCH-END)
```

thout batching, eac

WRITE

ommand waits for a

ACK

rom the Arduino before proceeding. With batching, all four writes are sent as a burst, and the Arduino processes them as fast as possible. This can be 4-10x faster for bulk operations like driving NeoPixel strip

Autopilot

(AUTOPILOT-BEGIN)

tarts recording hardware commands to the Arduino's RAM

(AUTOPILOT-END)

ommits the recording. The Arduino can then replay the sequence standalon

```
(AUTOPILOT-BEGIN)
(WRITE D 13 1)
(DELAY 500)
(WRITE D 13 0)
(DELAY 500)
(AUTOPILOT-END)
```

ter committing, flash the autopilot to boot memory wit

`hil-cli flash`

The autopilot buffer holds 64 commands (512 bytes). Each command is an 8-byte packed record. Delays are encoded as two records and support values up to 32,767 millisecond

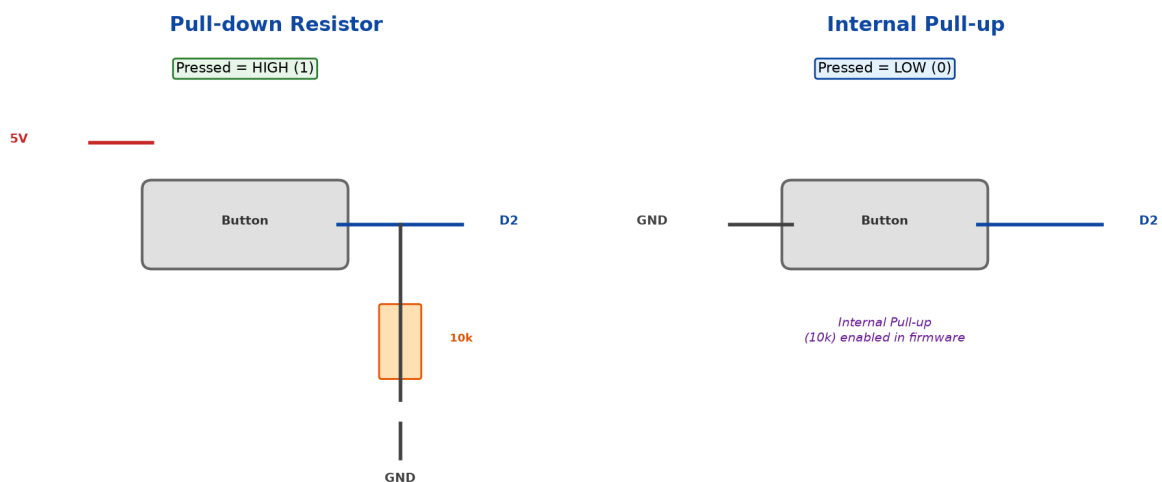
5. Sensors & Input Devices

Digital Sensors

Push Button / Toggle Switch

simple momentary push button or toggle switch. When pressed, it connects the pin to GND (active-low) or 5V (active-high)

Push Button Wiring Options



Push Button Wiring Options

figure 8: Push button wiring -- pull-down resistor (left) vs internal pull-up (right). The internal pull-up is simpler (no external resistor needed) and is enabled by the HIL-OS firmware

```
; Read button on D2 (active-high)
(SET button (READ D 2))
(IF (== button 1) ; 1 = pressed
  (LOG "Button pressed!"))
```

Reed Switch / Hall Effect Sensor

etects magnetic fields. Useful for door/window sensors, proximity detection, and speed measuremen

Wiring:

```

5V ----[Reed Switch]---- Arduino Pin D2
                        |
                        [10k ohm]
                        |
                        GND

; Detect magnet proximity
(WHILE (== 1 1)
  (DO
    (IF (== (READ D 2) 0) ; Reed switch closes when magnet near
      (LOG "Magnet detected!"))
    (DELAY 100)))

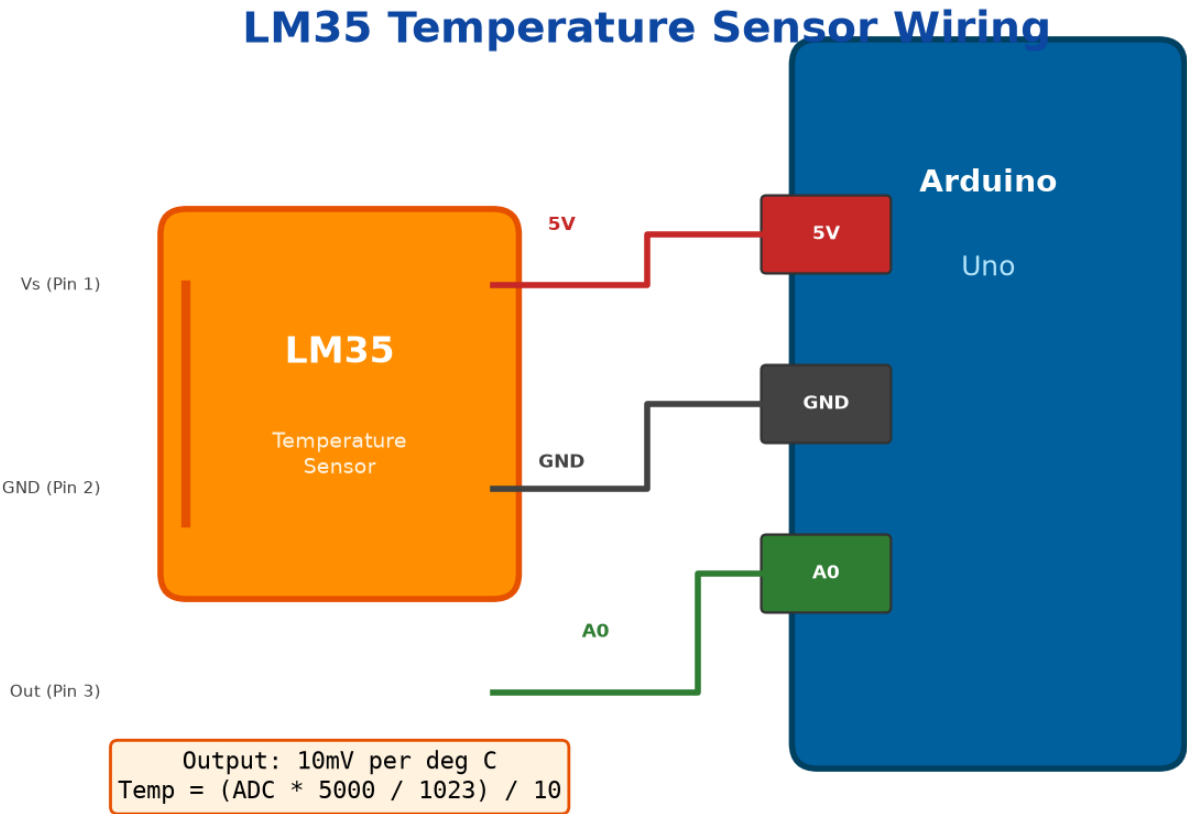
```

Analog Sensors

LM35 Temperature Sensor

The LM35 is a precision centigrade temperature sensor. It outputs 10mV per degree Celsius, so at 25C it outputs 250mV. The Arduino's ADC reads 0-1023 for 0-5V, so the formula is

```
temp = (analogRead * 5000 / 1023) / 10
```



LM35 Temperature Sensor Wiring

Figure 5: LM35 temperature sensor wiring diagram. The sensor outputs 10mV per degree Celsius

LM35 Pin	Arduino Pin	Description
1 (Vs)	5V	Power supply
2 (GND)	GND	Ground
3 (Out)	A0	Analog output

Wiring:

```
5V --- LM35 Pin 1
GND -- LM35 Pin 2
A0 --- LM35 Pin 3

; Read temperature from LM35 on A0
(SET temp (LM35 A0))
(LOG "Temperature: " temp " C")

; HIL-OS provides the LM35 macro which handles:
; 1. Reading the analog pin
; 2. Converting ADC value to millivolts
; 3. Dividing by 10 to get Celsius
; Internally: map(analogRead(pin), 0, 1023, 0, 500)
```

Accuracy tips:

Add a 0.1uF ceramic capacitor between Vs and GND, close to the sensor - Keep wire runs short (under 30cm)

to reduce noise - Take multiple readings and average them for better accuracy

Photoresistor (LDR)

light-dependent resistor changes resistance based on light intensity. In bright light, resistance is low (200-500 ohms); in darkness, it is high (1-10M ohms)

Wiring (Voltage Divider):



The voltage divider converts the changing resistance to a changing voltage that the Arduino can read. In bright light, the voltage at A0 is high (near 5V); in darkness, it is low (near 0V)

```
; Read light level from LDR on A0
(SET light (READ A 0))
(LOG "Light level: " light) ; 0 = dark, 1023 = bright

; Control LED based on light level
(IF (< light 200)
  (WRITE D 13 1) ; Dark -- turn on LED
  (WRITE D 13 0)) ; Bright -- turn off LED
```

Inverse wiring (for dark=high behavior):



Potentiometer (Variable Resistor)

potentiometer provides a manually adjustable voltage from 0-5V. It has three pins: two fixed ends and one wiper. Pinout:



Pot Pin	Arduino Pin	Description
1 (Left)	GND	Ground
2 (Middle/Wiper)	A0	Analog output
3 (Right)	5V	Power supply


```

; Read potentiometer position
(SET position (READ A 0)) ; 0-1023

; Use pot to control servo angle
(SET angle (MAP position 0 1023 0 180))
(V-SERVO 9 angle)

; Use pot to control LED brightness
(SET brightness (MAP position 0 1023 0 255))
(WRITE A 9 brightness)

```

Flex Sensor

flex sensor changes resistance when bent. Straight = ~10k ohm, bent = ~20-30k ohm. Useful for robotic grippers and gesture detectio

Wiring (same voltage divider as LDR):

```

5V ---[Flex Sensor]---+--- Arduino Pin A0
                      |
                      [10k ohm]
                      |
                      GND

; Read flex sensor
(SET flex (READ A 0))
(SET bendAngle (MAP flex 0 1023 0 90))
(LOG "Bend angle: " bendAngle " degrees")

```

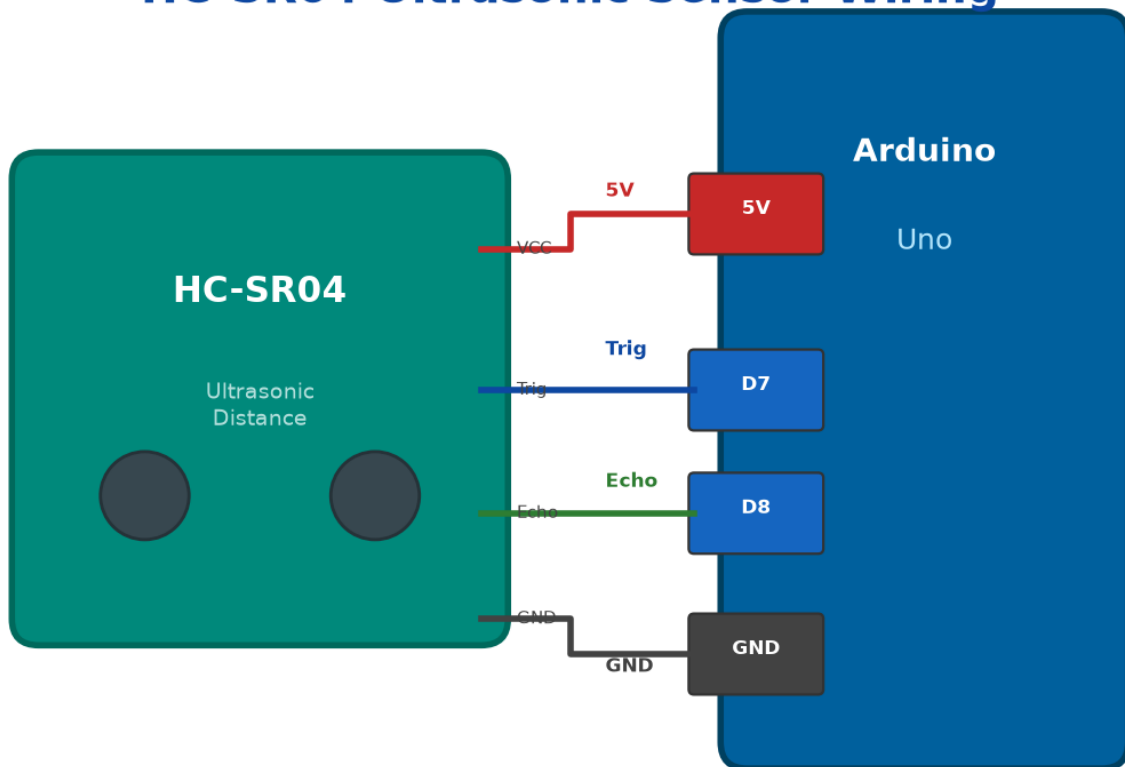
Distance Sensors

HC-SR04 Ultrasonic Distance Sensor

The HC-SR04 measures distance by sending ultrasonic pulses and timing the echo. It has four pins

Pin	Arduino Pin	Description
VCC	5V	Power supply
TRIG	D7	Trigger pulse input
ECHO	D8	Echo pulse output
GND	GND	Ground

HC-SR04 Ultrasonic Sensor Wiring



HC-SR04 Ultrasonic Sensor Wiring

Figure 7: HC-SR04 ultrasonic distance sensor wiring. TRIG on D7, ECHO on D8

How it works:

1. Arduino sends a 10-microsecond HIGH pulse on TRIG
2. The sensor sends 8 ultrasonic pulses at 40kHz
3. The ECHO pin goes HIGH for a duration proportional to the distance
4. Arduino measures the pulse width using `pulseIn()`

. Distance = (pulse width * speed of sound) /

speed of sound = 343 m/s = 0.0343 cm/microsecond

```

; Read distance from HC-SR04
(SET dist (ULTRASONIC 7 8))
(LOG "Distance: " dist " cm")

; Avoidance behavior
(WHILE (== 1 1)
  (DO
    (SET dist (ULTRASONIC 7 8))
    (IF (< dist 20)
      (DO
        (LOG "Obstacle! Stopping.")
        (WRITE D 9 0) ; Stop motor
        (WRITE D 10 1) ; Turn)
      (ELSE
        (WRITE D 9 1) ; Continue forward
        (DELAY 100)))
  )
)

```

Accuracy tips:

The sensor has a minimum range of about 2cm - Maximum range is about 400cm - Accuracy is +/- 3mm - The beam angle is about 15 degrees - Keep the sensor level and pointed at the target - Avoid measuring soft or angled surfaces that absorb sound

IR Proximity Sensor

infrared proximity sensor detects nearby objects using reflected IR light. It has analog and digital output

Wiring:

```

5V --- Sensor VCC
GND -- Sensor GND
D2 --- Sensor DIGITAL (optional, for simple proximity)
A0 --- Sensor ANALOG (optional, for distance measurement)

; Digital proximity detection
(SET proximity (READ D 2))
(IF (== proximity 0) ; Most IR sensors are active-low
  (LOG "Object detected!"))

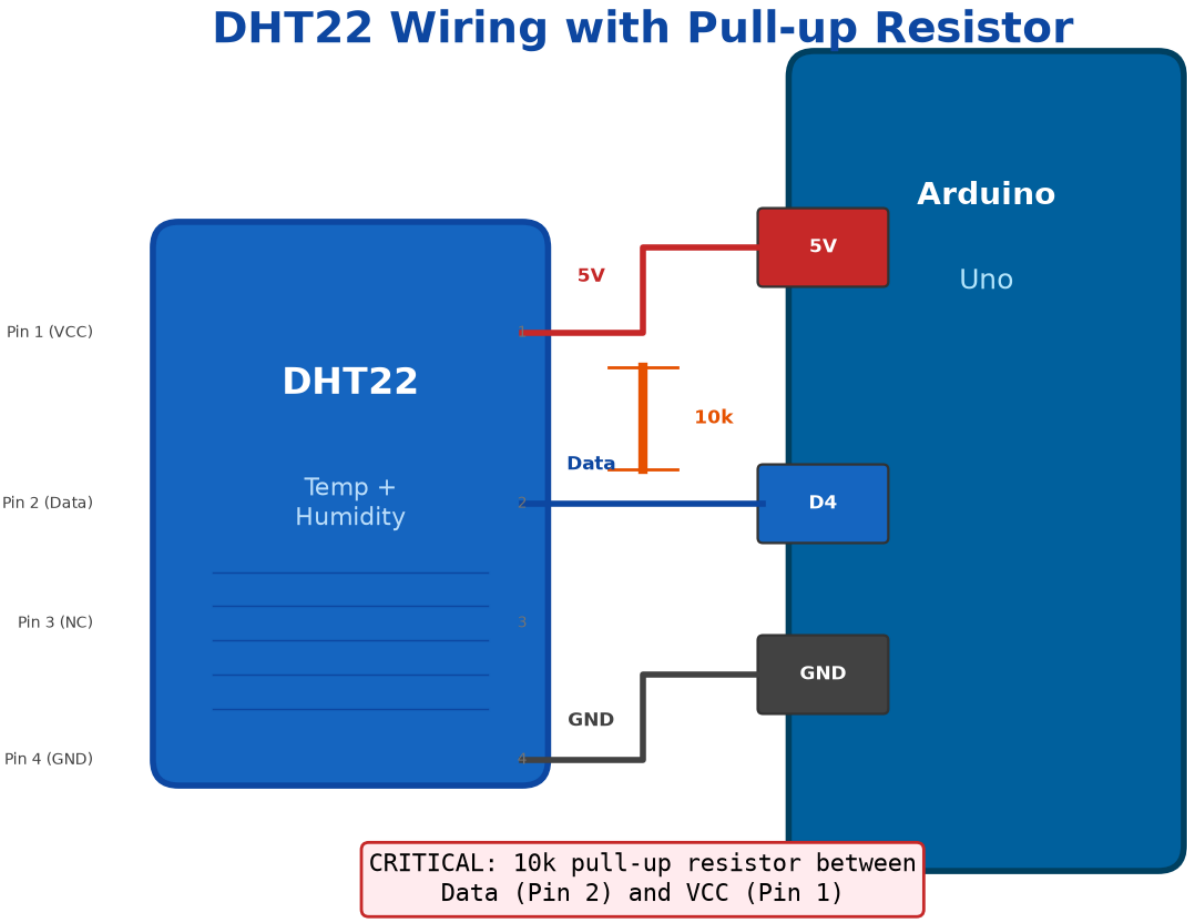
; Analog distance measurement (closer = higher value)
(SET distance (READ A 0))
(LOG "IR distance value: " distance)

```

Temperature/Humidity Sensors

DHT11 / DHT22

The DHT11 measures temperature (0-50C) and humidity (20-80%). The DHT22 (AM2302) is more accurate and wider range (temp: -40-80C, humidity: 0-100%)



DHT22 Wiring with Pull-up Resistor

figure 6: DHT22 wiring diagram showing the critical 10k pull-up resistor between Data and VCC

```
; Read DHT22 on pin 4
(DHT-READ 4 22) ; Returns (temperature humidity) tuple

; Store and log values
(SET reading (DHT-READ 4 22))
(SET temp (FIRST reading))
(SET humidity (SECOND reading))
(LOG "Temperature: " temp " C")
(LOG "Humidity: " humidity " %")
```

DHT11 vs DHT22:

Parameter	DHT11	DHT22
Temperature range	0-50C	-40-80C
Temperature accuracy	+/-2C	+/-0.5C
Humidity range	20-80%	0-100%
Humidity accuracy	+/-5%	+/-2-5%
Sampling rate	1Hz	0.5Hz
Price	\$1-2	\$4-8

6. Actuators & Output Devices

LEDs

Single LED

Wiring:

```
Arduino Pin D13 ---[330 ohm]---[LED Anode (long leg)]---[LED Cathode (short leg)]--- GND
```

Resistor selection:

Red LED: $V_f=2V$, $I=20mA$, $R = (5-2)/0.02 = 150 \text{ ohm}$ (use 150-330 ohm) - **Green LED:** $V_f=2.1V$, $I=20mA$, $R = (5-2.1)/0.02 = 145 \text{ ohm}$ - **Blue/White LED:** $V_f=3.2V$, $I=20mA$, $R = (5-3.2)/0.02 = 90 \text{ ohm}$

```
; Blink LED
(BLINK 13 5 500) ; Pin 13, 5 times, 500ms on/off

; PWM brightness control
(WRITE A 9 128) ; 50% brightness
(WRITE A 9 255) ; Full brightness
(WRITE A 9 0) ; Off
```

RGB LED (Common Cathode)

RGB LED has four pins: common cathode (longest leg) and three anodes (R, G, B

Pinout:

```
RGB LED (flat side facing you, longest leg = GND):
```

```

  | |
  | R G B |
  | |
  | | | |
  R G B G
  e r l N
  d e u D
```

Wiring:

```
Arduino D9 ---[220 ohm]--- Red Anode
Arduino D10 --[220 ohm]--- Green Anode
Arduino D11 --[220 ohm]--- Blue Anode
GND --- Common Cathode
```

```

; Set RGB LED color (0-255 for each)
(DEFUN SET-COLOR (r g b)
  (DO
    (WRITE A 9 r)      ; Red
    (WRITE A 10 g)     ; Green
    (WRITE A 11 b))) ; Blue

; Cycle through colors
(WHILE (== 1 1)
  (DO
    (SET-COLOR 255 0 0) ; Red
    (DELAY 1000)
    (SET-COLOR 0 255 0) ; Green
    (DELAY 1000)
    (SET-COLOR 0 0 255) ; Blue
    (DELAY 1000)
    (SET-COLOR 255 255 0) ; Yellow
    (DELAY 1000)
    (SET-COLOR 0 255 255) ; Cyan
    (DELAY 1000)
    (SET-COLOR 255 0 255) ; Magenta
    (DELAY 1000)))

```

Buzzer / Speaker

Passive Buzzer

passive buzzer requires a frequency signal to produce sound. It can play different notes at different frequencies. This is the type used in CRUMB (on pin D11

Wiring:

```

Arduino Pin D11 ---[Buzzer +]---[Buzzer -]--- GND

```

```

; Play a 1kHz tone for 200ms
(TONE 11 1000 200)

; Play a melody
(DEFUN NOTE (freq duration)
  (TONE 11 freq duration))

; Play C major scale
(NOTE 262 200) ; C4
(NOTE 294 200) ; D4
(NOTE 330 200) ; E4
(NOTE 349 200) ; F4
(NOTE 392 200) ; G4
(NOTE 440 200) ; A4
(NOTE 494 200) ; B4
(NOTE 523 200) ; C5

```

Common note frequencies:

Note	Frequency (Hz)	Note	Frequency (Hz)
C4	262	A4	440

D4	294	B4	494
E4	330	C5	523
F4	349	D5	587
G4	392	E5	659

Active Buzzer

active buzzer has an internal oscillator -- just apply HIGH and it beeps. It cannot play different notes. Note `TONE( )`

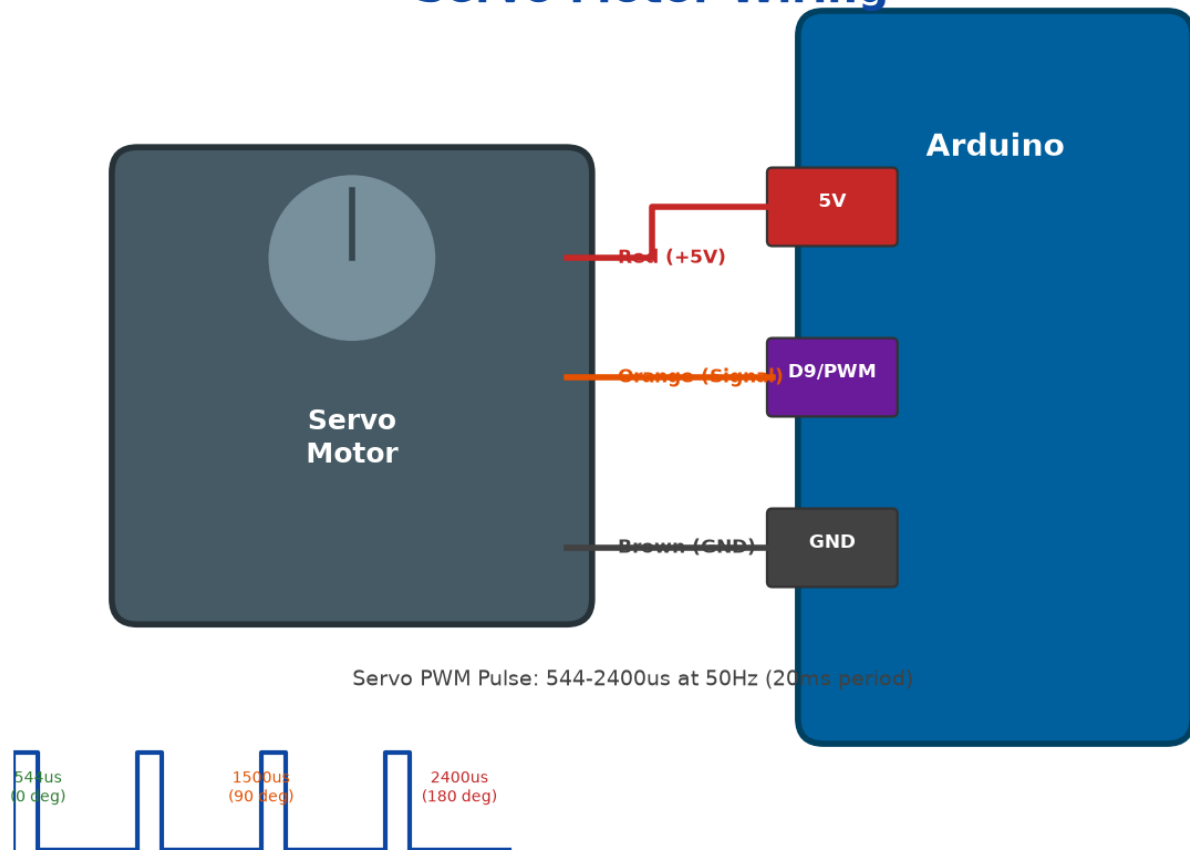
oes not work with active buzzers (it needs a passive buzzer to generate the frequency

```
; Active buzzer -- just turn on/off
(WRITE D 11 1)    ; Beep
(DELAY 200)
(WRITE D 11 0)    ; Stop
```

Servo Motors

rvo motors are controlled by PWM signals. The position is determined by the pulse width: 544ms = 0 degrees, 1500ms = 90 degrees, 2400ms = 180 degree

Servo Motor Wiring



Servo Motor Wiring

Figure 9: Servo motor wiring with PWM pulse width diagram. The servo position is controlled by pulse width: 544us = 0 degrees, 1500us = 90 degrees, 2400us = 180 degrees

```
; Set servo angle
(V-SERVO 9 90) ; Move to 90 degrees (center)
(V-SERVO 9 0) ; Move to 0 degrees
(V-SERVO 9 180) ; Move to 180 degrees

; Sweep servo from 0 to 180
(SET angle 0)
(WHILE (< angle 180)
  (DO
    (V-SERVO 9 angle)
    (DELAY 15)
    (SET angle (+ angle 1))))
```

HIL-OS Servo Macro:

h

V-SERVO

acro handles the microsecond pulse generation internally. It converts the angle (0-180) to a pulse width (544-2400 microseconds) using the formula

$$\text{pulse} = 544 + (\text{angle} * 1856) / 180$$

Motors

DC Motor (with Transistor)

DC motor draws too much current for an Arduino pin (up to several amps). You must use a transistor or motor drive

Wiring (with NPN transistor like 2N2222 or TIP120):

```

Arduino D9 ---[1k ohm]---[Base of TIP120]
                        Collector --- Motor - terminal
                        Emitter --- GND
                        Motor + terminal --- 5V (or external supply)

Flyback diode: 1N4007 across motor terminals (cathode to +)

; Control motor speed with PWM
(WRITE A 9 200) ; ~78% speed
(DELAY 2000)
(WRITE A 9 0)   ; Stop

```

L298N Motor Driver (for 2 motors or H-bridge)

The L298N module can drive two DC motors with speed and direction control

Wiring:

```

Arduino D5 --- ENA (speed motor A)
Arduino D6 --- IN1 (direction motor A)
Arduino D7 --- IN2 (direction motor A)
Arduino D9 --- ENB (speed motor B)
Arduino D10 -- IN3 (direction motor B)
Arduino D11 -- IN4 (direction motor B)

; Drive motor A forward at full speed
(WRITE D 6 1) ; IN1 HIGH
(WRITE D 7 0) ; IN2 LOW
(WRITE A 5 255) ; ENA full speed

; Drive motor A backward at half speed
(WRITE D 6 0) ; IN1 LOW
(WRITE D 7 1) ; IN2 HIGH
(WRITE A 5 128) ; ENA half speed

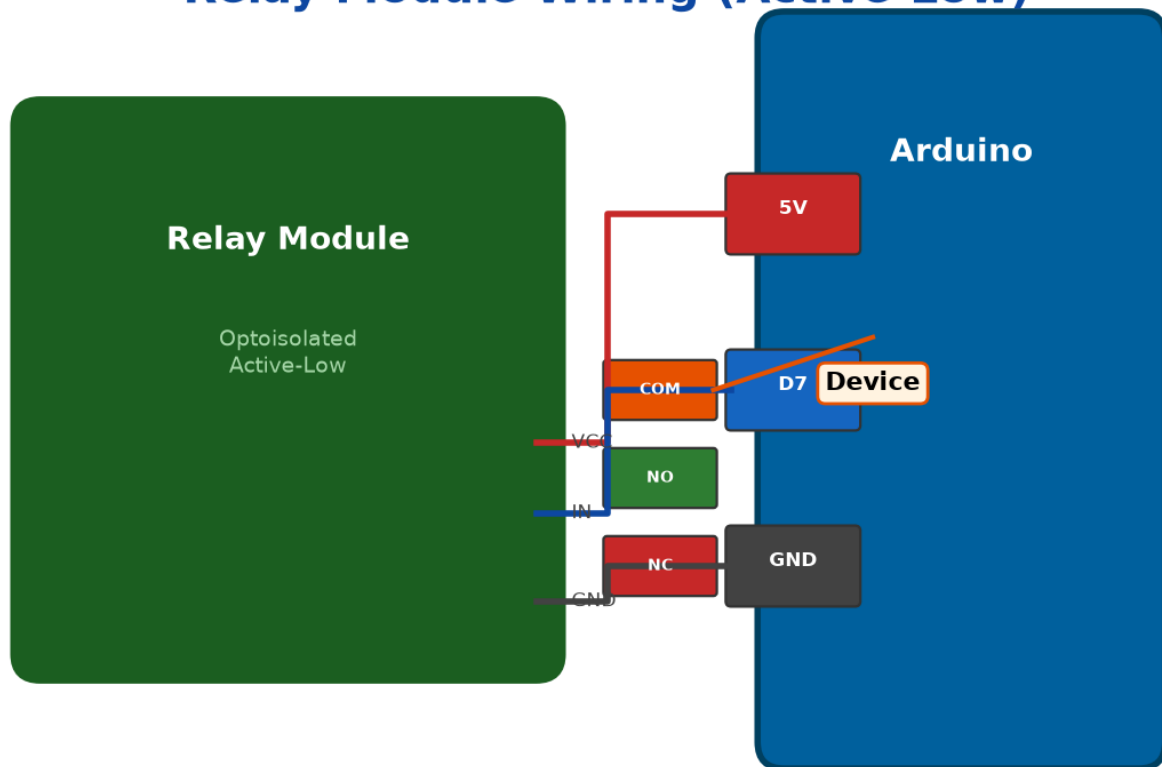
; Stop motor A
(WRITE A 5 0) ; ENA off

```

Relay Modules

Relays allow an Arduino to control high-voltage devices (lights, fans, appliances). Most relay modules are active-low and include a transistor drive

Relay Module Wiring (Active-Low)



Relay Module Wiring

Figure 10: Relay module wiring (active-low). The relay isolates the low-voltage Arduino side from the high-voltage device side

```
; Turn relay ON (device powered)
(WRITE D 7 0) ; Active-low relay

; Turn relay OFF
(WRITE D 7 1)
```

Safety warning:

Relays switching mains voltage (110V/220V) are dangerous. Never work on high-voltage circuits unless you are qualified. Use relay modules with optoisolation for safety.

NeoPixel / WS2812B LEDs

NeoPixel LEDs are individually addressable RGB LEDs. You can control hundreds of them with a single data pin. HIL-OS includes NeoPixel support via bit-bang protocol.

Wiring:

```

Arduino D6 ---[330 ohm]---[NeoPixel DIN]
5V ---[large capacitor 100-1000uF]--- NeoPixel VCC
GND --- NeoPixel GND

```

Note: For more than ~10 NeoPixels, use an external 5V supply.
Connect its GND to Arduino GND.

```

; Set pixel 0 to red
(NEO-SET 0 255 0 0)

; Set pixel 1 to green
(NEO-SET 1 0 255 0)

; Set pixel 2 to blue
(NEO-SET 2 0 0 255)

; Update all pixels
(NEO-SHOW)

; Rainbow effect
(DEFUN WHEEL (pos)
  (IF (< pos 85)
    (DO (SET r (* pos 3)) (SET g (* (- 255 pos) 3)) (SET b 0))
    (IF (< pos 170)
      (DO
        (SET pos (- pos 85))
        (SET r (* (- 255 pos) 3))
        (SET g 0)
        (SET b (* pos 3)))
      (DO
        (SET pos (- pos 170))
        (SET r 0)
        (SET g (* pos 3))
        (SET b (* (- 255 pos) 3))))))

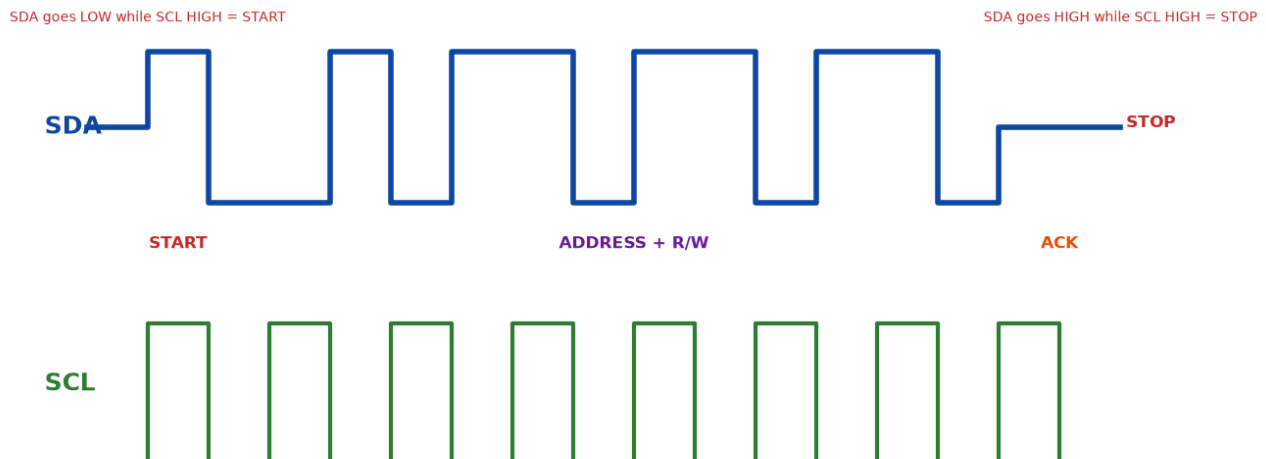
```

7. Communication Protocols

I2C (Inter-Integrated Circuit)

I2C is a two-wire serial protocol that allows multiple devices to share the same bus. It uses two lines: **SDA** (data) and **SCL** (clock). On the Arduino Uno, SDA is on A4 and SCL is on A5.

I2C Communication Protocol



I2C Timing Diagram

figure 13: I2C communication protocol -- START condition, address byte, R/W bit, ACK, and STOP

Wiring (I2C Bus):

```

Arduino A4 (SDA) ---+---+---+---+--- Device 1 SDA
                   |   |   |
Arduino A5 (SCL) ---+---+---+---+--- Device 1 SCL
                   |   |   |
                   GND GND GND (common ground)
  
```

Pull-up resistors (4.7k ohm):

5V ---[4.7k]--- A4 (SDA)

5V ---[4.7k]--- A5 (SCL)

Common I2C devices and their addresses:

Device	Address (7-bit)	Address (8-bit write)	Description
DS1307 RTC	0x68	0xD0	Real-time clock
MPU6050	0x68	0xD0	Accelerometer/gyro
BH1750	0x23	0x46	Light sensor
BMP280	0x76	0xEC	Barometric pressure
SSD1306	0x3C	0x78	OLED display
ADS1115	0x48	0x90	16-bit ADC

```

; Write a byte to an I2C device
(I2C-WRITE 0x68 0x00 0x00) ; Write 0x00 to register 0x00 on DS1307
  
```

```

; Read a byte from an I2C device
(I2C-READ 0x68 0x00) ; Read register 0x00 from DS1307
  
```

HIL-OS I2C Implementation:

he I2C functions use bit-bang implementation -- they manually toggle the SDA and SCL lines to implement the I2C protocol. This means: - Any two digital pins can be used for I2C (not just A4/A5) - The implementation is slower than hardware I2C (~100kHz vs 400kHz) - It works on all Arduino boards, not just those with hardware I

SPI (Serial Peripheral Interface)

I is a faster serial protocol than I2C. It uses four lines: MOSI (data out), MISO (data in), SCK (clock), and SS (chip select). On the Arduino Uno, these are on D11 (MOSI), D12 (MISO), D13 (SCK), and D10 (SS)

Wiring:

```
Arduino D11 (MOSI) --- Device MOSI
Arduino D12 (MISO) --- Device MISO
Arduino D13 (SCK)  --- Device SCK
Arduino D10 (SS)   --- Device CS
5V --- Device VCC
GND --- Device GND
```

Serial (UART)

rial communication is the most basic protocol. HIL-OS uses it for host-to-Arduino communication. The default baud rate is 960

Wiring:

```
Arduino D0 (RX) --- USB-to-Serial TX
Arduino D1 (TX) --- USB-to-Serial RX
GND --- Common ground
```

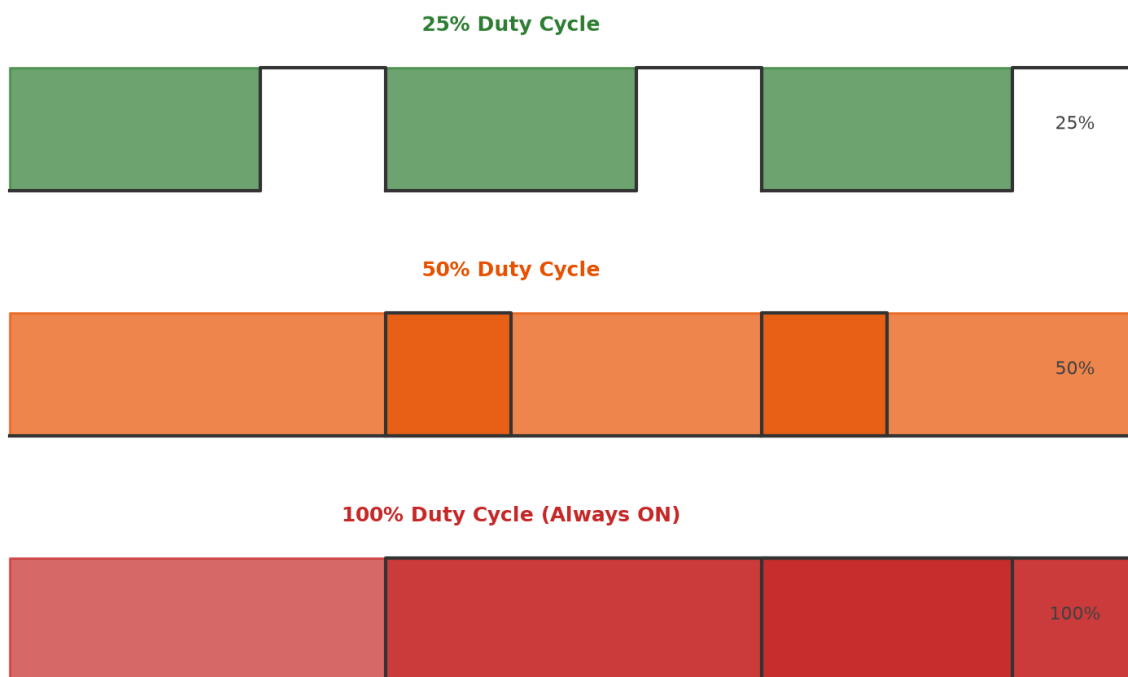
Important:

hen uploading sketches, disconnect anything on D0/D1 as they are shared with the USB-to-serial converte

Pulse Width Modulation (PWM)

M is not a communication protocol per se, but it is the primary method for controlling analog outputs on digital pins. Arduino PWM operates at ~490Hz on most pins and ~980Hz on pins D5 and D

PWM (Pulse Width Modulation) Duty Cycles



PWM Duty Cycle Diagram

Figure 12: PWM duty cycles -- 25% (dim LED), 50% (medium), 100% (full brightness)

Duty Cycle:

0% = 0V (always off) - 25% = ~1.25V - 50% = ~2.5V (half brightness for LEDs) - 75% = ~3.75V - 100% = 5V (always on)

```
; Set PWM output (0-255)
(WRITE A 9 128) ; 50% duty cycle on pin 9
(WRITE P 9 64) ; 25% duty cycle (P is alias for A in PWM mode)
```

8. CLI Command Reference

Project Management

`hil-cli init <name> [--template <tpl>]`

Creates a new project directory with the specified template. The template determines the initial file structure and sample code.

```
hil-cli init my-project --template basic
hil-cli new blinky --template blink
hil-cli init sensor-hub --template events
hil-cli init pid-controller --template pid
hil-cli init circuit-sim --template crumb
```

Available templates:

Template	Description	Files Created
`basic`	Full development environment	src/, lib/, .vscode/, hil.json, README.md
`blink`	Minimal LED blink	src/main.hil only
`events`	Event-driven sensors	src/main.hil with ON-CHANGE, ON-EVENT
`pid`	PID control system	src/main.hil with PID macros
`crumb`	Circuit simulator	src/main.hil with buzzer, optimized for CRUMB

`hil-cli list`

lists available project templates with description

```
hil-cli list
```

`hil-cli add <package>...`

installs a macro package into the project

lib/
directory and records it in
hil.json

```
hil-cli add cyber      ; Install cyber-physical macros
hil-cli add pid        ; Install PID tuning macros
hil-cli add cyber pid  ; Install both at once
```

Available packages:

Package	Files	Description
`cyber`	lib/cyber.hil	Alarm macros, PID initialization
`pid`	lib/pid.hil	PID tuning macros (P, I, D gain adjustment)

`hil-cli install`

installs all packages listed in

hil.json
Run this after cloning a project or after a fresh checkout

```
hil-cli install
```

`hil-cli list-pkgs`

lists all available macro packages with descriptions and file count

```
hil-cli list-pkgs
```

Running Scripts

`hil-cli run [file]`

arts the host monitor and injects the script. You can interact with the monitor's REPL. The monitor handles all hardware communicatio

```
hil-cli run src/main.hil
hil-cli run                ; Uses src/main.hil or hil.json main field
HIL_PORT=COM3 hil-cli run  ; Specify port via environment variable
HIL_BAUD=115200 hil-cli run ; Specify baud rate
```

`hil-cli simulate [file]`

ns the script headlessly with no hardware required. Uses JSON IPC mode. All hardware commands return

```
hil-cli simulate src/main.hil
```

is is useful for: - Testing logic without hardware - CI/CD pipelines - Automated testing - Debugging LISP synt

`hil-cli build [file]`

mpiles the script to an Intel HEX artifact. This creates a binary image that can be flashed to the Arduin

```
hil-cli build src/main.hil
```

`hil-cli send [file]`

jects the script into an already-running monitor via TCP localhost:9000. This is useful for sending scripts to a monitor that is already connected to hardwar

```
hil-cli send src/main.hil
```

`hil-cli stop`

nds the halt signal to the running monitor. This stops all script executio

```
hil-cli stop
```

`hil-cli flash [file]`

rns the script into the node's permanent memory (autopilot). This clears existing autopilot, streams the script, and commits to boot memor

```
hil-cli flash src/main.hil
```

Firmware (Real Hardware)

`hil-cli firmware info`

Shows available build tools and Arduino board specification

```
hil-cli firmware info
```

tpu

```
Available firmware build tools:
```

```
TinyGo:    not found
avr-gcc:   not found
avrdude:   not found
arduino-cli: not found
```

```
Arduino Uno specifications:
```

```
Chip:  ATmega328P
Flash: 32KB
SRAM:  2KB
Clock: 16MHz
```

`hil-cli firmware build [--board uno|nano|mega] [--out path]`

mpile

firmware.go

o

.hex

il

```
hil-cli firmware build --board uno
hil-cli firmware build --board nano --out my-firmware.hex
hil-cli firmware build --board mega
```

`hil-cli firmware flash [--board uno|nano|mega] [--port COM3] [--hex path] [--baud rate]`

ashes firmware to a real Arduino board. Auto-detects serial port if not specific

```
hil-cli firmware flash --board uno --port COM3
hil-cli firmware flash --board nano
hil-cli firmware flash --board mega --baud 115200
```

VS Code Extension

`hil-cli install-vscode`

ackages and installs the HIL LISP VS Code extension. Provides syntax highlighting, code snippets, and right-click action

```
hil-cli install-vscode
```

Serial Port Scanner

`hil-cli ports`

Scans all serial ports and presents an interactive selection menu

```
hil-cli ports
```

Example output:

```
Scanning serial ports...
COM1: [IN USE] - Microsoft Serial Port
COM3: [AVAILABLE] - Arduino Uno
COM5: [AVAILABLE] - Unknown device

Select a port (1-3): 2
Selected: COM3 (Arduino Uno)

Use in HIL-OS:
HIL_PORT=COM3 hil-cli run src/main.hil
```

Diagnostics

`hil-cli doctor`

Checks the entire toolchain: Go, Node.js, monitor, Node wrapper, CRUMB, transpiler, and CRUMB generator

```
hil-cli doctor
```

9. Firmware & Deployment

Architecture

The firmware

resides in `firmware.go`

It runs on ATmega328P boards (Arduino Uno/Nano). It is compiled with TinyGo and communicates with the host over a serial connection

The firmware operates in two modes

- **Slave mode:** Executes hardware commands received from the host
- **Autopilot mode:** Replays a pre-recorded command sequence from RAM

Pin Assignments

Pin	Function
D0-D13	Digital I/O, PWM, servo, events
A0-A5	Analog input, I2C (A4=SDA, A5=SCL)
D11	Passive buzzer (for CRUMB)
D9	Common servo output (PID default)

Supported Boards

Board	Chip	Flash	SRAM	Clock	TinyGo Target
Arduino Uno	ATmega328P	32KB	2KB	16MHz	`arduino`
Arduino Nano	ATmega328P	32KB	2KB	16MHz	`arduino-nano`
Arduino Mega	ATmega2560	256KB	8KB	16MHz	`arduino-mega`

Autopilot Memory

The autopilot buffer holds 64 commands (512 bytes). Each command is an 8-byte packed record

```
val1 int16 | val2 int16 | pin int8 | action byte | typ byte | pad byte
```

Layers are encoded as two records

```
A W 68 0 0
```

action='D') followed by

```
A D <ms> 0 0
```

val1=ms). This supports delays up to 32,767 millisecond

Background Tasks

The firmware runs several background tasks every 1 millisecond

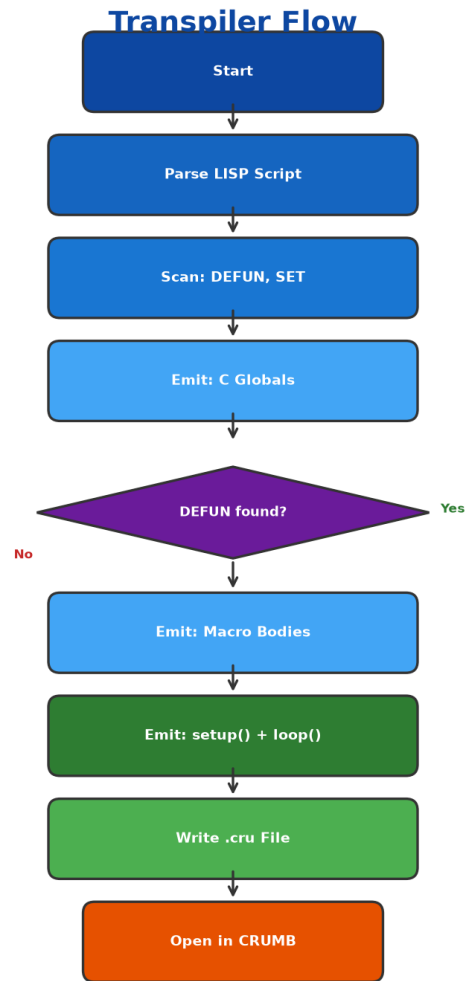
- **PID controller (10Hz):** Reads analog input, computes error, updates integral/derivative, drives output servo
- **Servo PWM (50Hz):** Generates hardware-timed servo pulses on all active pins
- **Digital event watch:** Detects state changes and emits interrupt events
- **Analog rate-of-change watch:** Monitors analog pins for threshold crossings with EMA smoothing

10. CRUMB Circuit Simulation

Overview

CRUMB is a Unity/IL2CPP circuit simulator with a virtual Arduino Nano. HIL-OS transpiles your LISP scripts into Arduino C and injects them into a CRUMB circuit file

How it works



Transpiler Flow

Figure 11: Transpiler flow -- from LISP source to CRUMB circuit file

Supported Operations

The transpiler supports all core LISP operations plus every sensor and peripheral

- **Core:** SET, DO, IF, WHILE, AWAIT, DELAY, LOG, EXIT, DEFUN, INCLUDE
- **I/O:** WRITE D/A/P, READ D/A, TOGGLE, TONE, PULSE, BLINK
- **Math:** +, -, *, /, ==, !=, >, <, MAP, CONSTRAIN, RANDOM
- **Sensors:** LM35, ULTRASONIC, DHT-READ
- **Peripherals:** SERVO, I2C-WRITE/READ, LCD-INIT/CLEAR/PRINT, PID-INIT/CLOSED-LOOP/OPEN-LOOP
- **Communication:** MP3-PLAY/VOL, SD-WRITE, NEO-SET/SHOW, RTC-READ
- **Events:** ATTENTION, ON-EVENT, ON-CHANGE
- **Advanced:** STRUCT, NEW, FIRST, SECOND, BATCH-BEGIN/END, AUTOPILOT-BEGIN/END

Implementations use standard Arduino API

`digitalWrite`

`analogRead`

micros

pulseIn

etc.) and work on both the CRUMB simulator and real Arduino hardware

Name Sanitization

L names with hyphens are converted to valid C identifiers

DUAL-BLINK

ecome

DUAL_BLINK

All user-defined names are uppercase

Pin Notation

duino pin names pass through correctly:

A0

hroug

A7

or analog pins

D0

hroug

D13

or digital pins - Hex literals lik

0x68

or I2C addresses - Float literals lik

2.0

or PID gai

Buzzer

e CRUMB template includes a passive buzzer on pin D11. Us

(TONE 11 1000 500)

o produce sound. Active buzzers will not work -

tone()

equires a passive buzze

11. Serial Wire Protocol

Frame Format

```
ACTION TYPE PIN VAL1 VAL2|CRC\r
```

elds are space-separated ASCII. The CRC is the XOR of every byte in the command part, formatted as two uppercase hex digits. Frames end with carriage return

\r

ample

W D 13 1 0|5D\r

Actions

Action	Letter	Example
Digital write	`W`	`W D 13 1 0` (pin 13 HIGH)
PWM write	`W`	`W A 9 128 0` (50% duty cycle)
Digital read	`R`	`R D 2 0 0` -> replies `D:2:1`
Analog read	`R`	`R A 0 0 0` -> replies `A:0:512`
Tone	`T`	`T D 8 1000 200` (1kHz, 200ms)
Pulse train	`S`	`S D 13 5 1000` (5 toggles, 1ms period)
Event arm	`E`	`E D 2 3 0` (arm D2 RISING)
Change arm	`C`	`C A 0 512 0` (arm A0 threshold)
I2C write	`I`	`I W 104 0 0` (write to addr 0x68)
I2C read	`I`	`I R 104 1 0` -> replies `I:104:42`
LCD	`L`	`L I 39 0 0` (init), `L P 0 0` text
Servo	`V`	`V S 9 90 0` (90 degrees on pin 9)
MP3	`M`	`M P 10 1 0` (play track 1)
SD card	`F`	`F W 10 hello`
Sensor	`H`	`H D 4 22 0` (DHT22), `H L 0 0 0` (LM35)
NeoPixel	`N`	`N S 0 255 0` (set pixel 0)
PID	`Z`	`Z I 0 0 9` (init ch0), `Z S 0 500 0` (setpoint)
Attention	`X`	`X S 0 1 0` (high-frequency polling)
Autopilot	`A`	`A C 0 0 0` (clear), `A E 1 0 0` (commit)
Batch flush	`Q`	`Q Q 0 0 0`
Ping	`P`	`P _ _ _` -> `PONG`

Replies

Reply	Meaning
`ACK`	Command accepted
`D:pin:val`	Digital read result
`A:pin:val`	Analog read result
`I:addr:val`	I2C read result
`(temp hum)`	DHT sensor reading
`(dist)`	Ultrasonic distance
`(temp)`	LM35 temperature
`EVT:D:pin:val`	Digital event fired
`EVT_A:A:pin:val`	Analog change event

`ERR:CRC`	CRC mismatch
-----------	--------------

12. JSON IPC & Node.js API

JSON IPC Protocol

The JSON IPC protocol allows headless operation without a serial port. Start the monitor with

```
-ipc -port none
```

and communicate via stdin/stdout

Monitor -> Client Messages:

```
{"type":"ready","msg":"Connected on none"}
{"type":"hardware_event","pin":"D2","value":1}
{"type":"execution_complete"}
```

Client -> Monitor Messages:

```
(LISP FORM 1)
(LISP FORM 2)
...
__HIL_END__
```

Scripts are terminated by a line containing exactly

```
__HIL_END__
```

Multiple scripts can be sent sequentially

Node.js API

Installation

```
npm install ./hil-node
```

Usage

```

const HIL_OS = require('./hil-node/hil.js');

const hil = new HIL_OS();

// Event listeners
hil.on('log', line => console.log(line));
hil.on('interrupt', ({ pin, value }) => console.log(`pin ${pin} = ${value}`));
hil.on('idle', () => hil.shutdown());

// Connect (headless, no hardware)
await hil.connect('none', 9600);

// Execute a script
hil.execute(`
  (SET n 0)
  (WHILE (< n 3)
    (DO (LOG "n=" n) (SET n (+ n 1))))
  (LOG "done")
`);

```

API Reference

Method	Description
<code>`new HIL_OS(path?)`</code>	Create instance. Optional Go binary path.
<code>`async connect(port, baud)`</code>	Connect to monitor. Use <code>`none`</code> for headless.
<code>`execute(lispScript)`</code>	Send a LISP script for execution.
<code>`shutdown()`</code>	Kill the monitor process.

Events:

Event	Payload	Description
<code>`log`</code>	string	LOG output from the engine
<code>`interrupt`</code>	<code>`{pin, value}`</code>	Hardware event (digital or analog)
<code>`idle`</code>	none	Script execution complete
<code>`error`</code>	string	Engine error or stderr
<code>`disconnect`</code>	string	Engine process exited

13. VS Code Extension

Installation

```

hil-cli install-vscode

```

Features

- **Syntax highlighting** for `.hil`` and `.lisp`` files
- **Language configuration** -- auto-closing parentheses, bracket matching, foldable S-expressions

- **20+ code snippets** -- type a prefix and press Tab to insert common patterns
- **Right-click actions** -- Run, Simulate, Transpile to CRUMB

Snippets

Prefix	Description
<code>`blink`</code>	Blink an LED
<code>`while`</code>	While loop
<code>`if`</code>	If/else conditional
<code>`set`</code>	Set a variable
<code>`defun`</code>	Define a macro
<code>`write-d`</code>	Digital write
<code>`write-a`</code>	Analog write (PWM)
<code>`read-d`</code>	Digital read
<code>`read-a`</code>	Analog read
<code>`map`</code>	Map a value range
<code>`servo`</code>	Set servo angle
<code>`ultrasonic`</code>	Read ultrasonic distance
<code>`dht`</code>	Read DHT sensor
<code>`lm35`</code>	Read LM35 temperature
<code>`lcd-print`</code>	Initialize and print to LCD
<code>`i2c-write`</code>	Write to I2C device
<code>`i2c-read`</code>	Read from I2C device
<code>`neo-set`</code>	Set NeoPixel color
<code>`mp3-play`</code>	Play MP3 track
<code>`pid`</code>	Initialize PID controller
<code>`include`</code>	Include another HIL file
<code>`log`</code>	Print to serial monitor

14. Transpiler Deep Dive

Overview

The transpiler converts HIL LISP scripts into Arduino C code. It operates in two passes

- **Scan pass:** Registers all ``DEFUN`` macros and collects all ``SET`` variable targets
- **Emit pass:** Generates C globals, helper functions, macro prototypes and bodies, and the ``setup()``/``loop()`` entry points

Generated Code Structure

```

// Global variables
static int LED = 0;
static int TEMP = 0;
static int _dht_t = 0, _dht_h = 0;

// Helper functions
int hilReadD(int pin){ return digitalRead(pin); }
int hilReadA(int pin){ return analogRead(pin); }
void hilWriteD(int pin, int val){ digitalWrite(pin, val); }
void hilWriteA(int pin, int val){ analogWrite(pin, val); }
// ... 200+ lines of real Arduino C ...

// Forward declarations
int MY_MACRO(int ARG1);

// Macro bodies
int MY_MACRO(int ARG1){
    digitalWrite(ARG1, HIGH);
    delay(500);
    digitalWrite(ARG1, LOW);
    return 0;
}

// Setup
void setup(){
    Serial.begin(9600);
}

// Run-once guard
static int _hil_ran = 0;

// Main loop
void loop(){
    if (!_hil_ran){
        _hil_ran = 1;
        MY_MACRO(13);
    }
}

```

Helper Functions

e transpiler injects a comprehensive set of helper functions that use only standard Arduino AP

Category	Functions
Digital/Analog I/O	`hilWriteD`, `hilWriteA`, `hilReadD`, `hilReadA`,
I2C bit-bang	`hilI2CSetPins`, `hilI2CWriteByte`, `hilI2CReadByt
DHT sensor	`hilDHTRead` (full bit-bang protocol with checksum
Ultrasonic	`hilUltrasonicCm` (trigger pulse + `pulseIn`)
Servo	`hilServoWrite` (PWM pulse via `micros`)
LCD 4-bit	`hilLCDInit`, `hilLCDCommand`, `hilLCDChar`, `hilL
PID	`hilPIDInit`, `hilPIDCompute` (proportional-integr
NeoPixel	`hilNeoSetByte`, `hilNeoSet`, `hilNeoShow` (WS2812
MP3	`hilMP3Cmd`, `hilMP3Play`, `hilMP3Vol` (DFPlayer s

RTC	`hilRTCRead`, `hilRTCWrite` (DS1307 via I2C)
SD	`hilSDInit`, `hilSDWrite` (SPI transfer)

Strict Mode

t

```
--strict
```

the transpiler refuses to generate output if any warnings occur. Withou

```
--strict
```

warnings are printed but generation continue

15. Advanced Topics

Power Management

Powering from USB:

SB provides 5V at up to 500mA. This is enough for the Arduino and most sensors, but not for motors or high-power LED

Powering from VIN:

se a 7-12V wall adapter or 9V battery. The onboard regulator provides 5V at up to 500mA. The input voltage minus 5V is dissipated as heat, so at 12V input, the regulator dissipates $7V \times 0.5A = 3.5W$ -- it will get hot. Use a heatsink or keep input voltage lo

External 5V for high-power devices:

or servos, motors, and NeoPixels, use a separate 5V supply. Connect its GND to the Arduino GND but do NOT connect its 5V to the Arduino 5V pin. Power the high-power devices directly from the external suppl

Battery operation:

or portable projects, use a 9V battery or a 2-cell LiPo (7.4V). The autopilot mode allows the Arduino to run standalone without a computer connectio

Noise Reduction

Decoupling capacitors:

lace 0.1uF ceramic capacitors close to the power pins of every IC and sensor. This filters high-frequency noise from the power suppl

Analog sensor noise:

Keep analog wires short and away from digital wires - Add 0.1uF capacitors on sensor power pins - Use software averaging (read multiple times and average) - Use th

```
READ-AVG
```

acro for automatic averagi

Motor noise:

Add 100uF electrolytic capacitor across motor terminals - Add flyback diode (1N4007) across motor terminals - Use separate power supply for motors - Keep motor wires away from sensor wir

Multiplexing

When you need more analog inputs than the Arduino has, use an analog multiplexer like the CD4051 (8 channels) or MCP3008 (8-channel 10-bit ADC via SPI)

CD4051 Wiring:

```

Arduino A0 --- CD4051 COMMON (output)
Arduino D2 --- CD4051 S0 (channel select bit 0)
Arduino D3 --- CD4051 S1 (channel select bit 1)
Arduino D4 --- CD4051 S2 (channel select bit 2)
5V --- CD4051 VEE
GND --- CD4051 VSS
5V --- CD4051 VDD

; Read channel 0-7 from CD4051
(DEFUN MUX-READ (channel)
  (DO
    (WRITE D 2 (& channel 1))      ; S0 = bit 0
    (WRITE D 3 (& channel 2))      ; S1 = bit 1
    (WRITE D 4 (& channel 4))      ; S2 = bit 2
    (DELAY 1)                      ; Let multiplexer settle
    (READ A 0)))                  ; Read common output

```

Interrupts

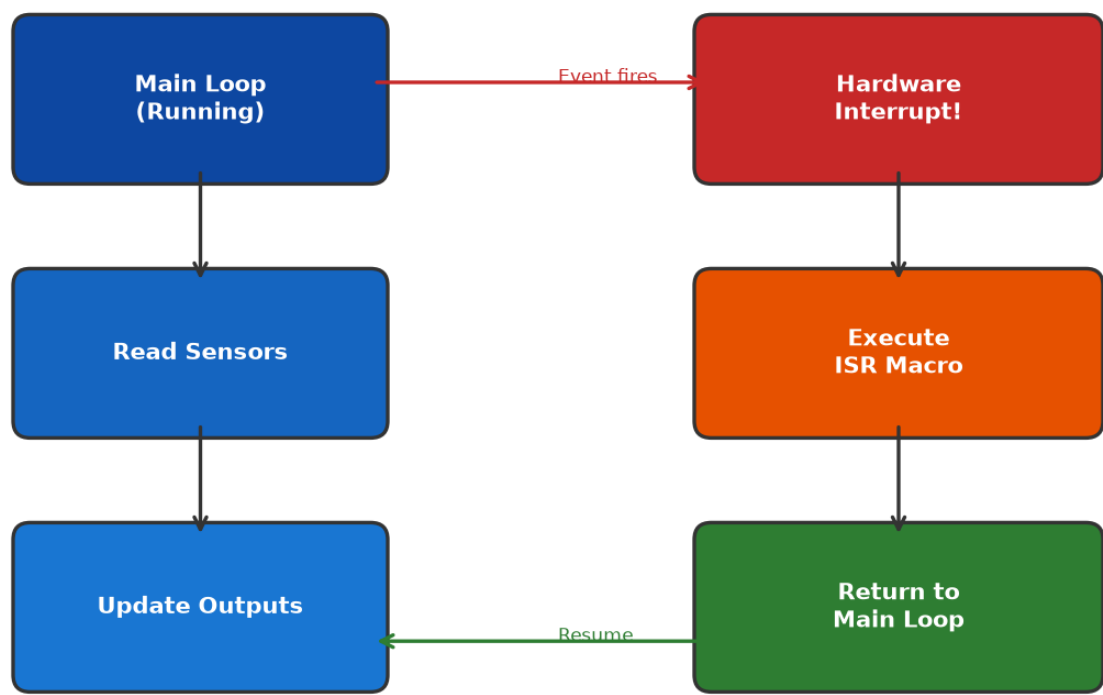
L-OS supports hardware interrupts via

ON-EVENT

n

ON-CHANGE

Interrupt-Driven Event Flow



Interrupt-Driven Event Flow

figure 15: Interrupt-driven event flow -- hardware events interrupt the main loop, execute the ISR macro, then return to the main loop

```
; Call EMERGENCY-STOP when pin D2 goes LOW (button press)
(ON-EVENT 2 0 EMERGENCY-STOP) ; Pin 2, FALLING edge

; Call SENSOR-UPDATE when analog pin A0 crosses threshold
(ON-CHANGE 0 0 512 SENSOR-UPDATE) ; Pin A0, threshold 512
```

Arduino Uno interrupt pins:

Pin	Interrupt	Notes
D2	INT0	External interrupt 0
D3	INT1	External interrupt 1

ly D2 and D3 support hardware interrupts on the Uno. The Mega has 6 interrupt pins (D2-D4, D18-D21

Timing and Precision

millis() vs delay():

DELAY

locks execution. For non-blocking timing, use a variable to track the last time an action occurs

```
(SET lastBlink 0)
(SET ledState 0)

(WHILE (== 1 1)
  (DO
    (SET now (MILLIS))
    (IF (> (- now lastBlink) 500)
      (DO
        (SET lastBlink now)
        (IF (== ledState 0)
          (DO (WRITE D 13 1) (SET ledState 1))
          (DO (WRITE D 13 0) (SET ledState 0)))))))
```

is allows the Arduino to do other things between blinks, such as reading sensors or responding to serial command

16. Complete Projects

Project 1: Automatic Night Light

light-sensitive LED that turns on when it gets dark

Hardware:

Arduino Uno - Photoresistor (LDR) - 10k ohm resistor - LED - 330 ohm resistor

Wiring:

```
5V ---[LDR]----+--- A0
                |
                [10k ohm]
                |
                GND

D13 ---[330 ohm]---[LED]--- GND
```

Code:

```

; Automatic Night Light
; Turns on LED when ambient light is below threshold

(LOG "Automatic Night Light Starting...")
(LOG "Monitoring light level on A0")

(SET threshold 300)
(SET lastState 0)

(WHILE (== 1 1)
  (DO
    (SET light (READ A 0))
    (LOG "Light level: " light)

    (IF (< light threshold)
      (DO
        (IF (== lastState 0)
          (DO
            (WRITE D 13 1)
            (SET lastState 1)
            (LOG "Night mode ON"))))
        (DELAY 100))
      (DO
        (IF (== lastState 1)
          (DO
            (WRITE D 13 0)
            (SET lastState 0)
            (LOG "Night mode OFF"))))
        (DELAY 100))))))

```

Project 2: Temperature Monitor with Alarm

monitors temperature and sounds an alarm when it gets too hot

Hardware:

Arduino Uno - LM35 temperature sensor - Passive buzzer - LED

Wiring:

```

5V --- LM35 Pin 1
GND -- LM35 Pin 2
A0 --- LM35 Pin 3

D11 --- [Buzzer] --- GND
D13 --- [330 ohm] --- [LED] --- GND

```

Code:

```

; Temperature Monitor with Alarm
; Sounds alarm and flashes LED when temperature exceeds threshold

(SET alarmTemp 35)
(SET buzzerPin 11)
(SET ledPin 13)

(LOG "Temperature Monitor Starting...")
(LOG "Alarm threshold: " alarmTemp " C")

(WHILE (== 1 1)
  (DO
    (SET temp (LM35 A0))
    (LOG "Current temperature: " temp " C")

    (IF (> temp alarmTemp)
      (DO
        (LOG "WARNING: Temperature too high!")
        (WRITE D ledPin 1)
        (TONE buzzerPin 1000 200)
        (DELAY 200)
        (WRITE D ledPin 0)
        (TONE buzzerPin 800 200)
        (DELAY 200))
      (DO
        (WRITE D ledPin 0)))

    (DELAY 1000)))

```

Project 3: Ultrasonic Parking Sensor

tects obstacles and beeps faster as they get close

Hardware:

Arduino Uno - HC-SR04 ultrasonic sensor - Passive buzzer - L

Wiring:

```

5V --- HC-SR04 VCC
GND -- HC-SR04 GND
D7 --- HC-SR04 TRIG
D8 --- HC-SR04 ECHO

D11 --- [Buzzer] --- GND
D13 --- [330 ohm] --- [LED] --- GND

```

Code:


```

; Ultrasonic Parking Sensor
; Beeps faster as obstacle gets closer

(SET trigPin 7)
(SET echoPin 8)
(SET buzzerPin 11)
(SET ledPin 13)
(SET maxDist 100)
(SET alarmDist 30)

(LOG "Parking Sensor Active")

(WHILE (== 1 1)
  (DO
    (SET dist (ULTRASONIC trigPin echoPin))
    (LOG "Distance: " dist " cm")

    (IF (< dist alarmDist)
      (DO
        ; Very close -- continuous alarm
        (WRITE D ledPin 1)
        (TONE buzzerPin 2000 100)
        (DELAY 100))
      (IF (< dist maxDist)
        (DO
          ; Getting closer -- beep faster
          (SET beepDelay (MAP dist maxDist alarmDist 500 100))
          (WRITE D ledPin 1)
          (TONE buzzerPin 1500 50)
          (DELAY beepDelay)
          (WRITE D ledPin 0)
          (DELAY beepDelay))
        (DO
          ; Safe distance -- slow beep
          (WRITE D ledPin 0)
          (DELAY 1000))))))

```

Project 4: Servo-Controlled Robot Arm

Control a servo motor with a potentiometer

Hardware:

Arduino Uno - Servo motor - Potentiometer (10k ohm)

Wiring:

```

Potentiometer:
  Pin 1 --- GND
  Pin 2 --- A0
  Pin 3 --- 5V

Servo:
  Red --- 5V (or external supply)
  Brown --- GND
  Orange --- D9

```

Code:

```

; Servo-Controlled Robot Arm
; Uses potentiometer to control servo angle

(SET potPin A0)
(SET servoPin 9)
(SET minAngle 0)
(SET maxAngle 180)

(LOG "Robot Arm Control Active")
(LOG "Turn potentiometer to move arm")

(WHILE (== 1 1)
  (DO
    (SET potValue (READ A potPin))
    (SET angle (MAP potValue 0 1023 minAngle maxAngle))
    (SET angle (CONSTRAIN angle minAngle maxAngle))
    (V-SERVO servoPin angle)
    (DELAY 15)))

```

Project 5: DHT22 Weather Station

ad temperature and humidity, display on LC

Hardware:

Arduino Uno - DHT22 temperature/humidity sensor - 16x2 LCD display (HD44780) - 10k ohm pull-up resistor for DHT

Wiring:

```

DHT22:
  Pin 1 --- 5V
  Pin 2 --- D4 ---[10k ohm]--- 5V (pull-up)
  Pin 4 --- GND

LCD (4-bit mode):
  RS --- D7
  EN --- D6
  D4 --- D5
  D5 --- D4
  D6 --- D3
  D7 --- D2
  VSS --- GND
  VDD --- 5V
  VO --- [10k pot for contrast]
  A --- 5V (backlight)
  K --- GND

```

Code:

```

; DHT22 Weather Station
; Reads temperature and humidity, displays on LCD

(LCD-INIT 7 6 5 4 3 2) ; rs, en, d4, d5, d6, d7
(LCD-CLEAR)
(LCD-PRINT 0 0 "Weather Station")
(LCD-PRINT 1 0 "Initializing..")
(DELAY 2000)
(LCD-CLEAR)

(SET dhtPin 4)
(SET dhtModel 22)

(WHILE (== 1 1)
  (DO
    (SET reading (DHT-READ dhtPin dhtModel))
    (SET temp (FIRST reading))
    (SET humidity (SECOND reading))

    (LCD-CLEAR)
    (LCD-PRINT 0 0 "Temp: " temp " C")
    (LCD-PRINT 1 0 "Hum: " humidity " %")

    (LOG "Temp: " temp " C Humidity: " humidity " %")

    (DELAY 2000)))

```

Project 6: SOS Morse Code Beacon

ansmits SOS in Morse code on LED and buzze

Hardware:

Arduino Uno - LED - 330 ohm resistor - Passive buzz

Wiring:

```

D13 ---[330 ohm]---[LED]--- GND
D11 ---[Buzzer]--- GND

```

Code:

```

; SOS Morse Code Beacon
; Transmits SOS (... --- ...) on LED and buzzer

(DEFUN DOT ()
  (DO
    (WRITE D 13 1)
    (TONE 11 750 100)
    (DELAY 100)
    (WRITE D 13 0)
    (DELAY 100)))

(DEFUN DASH ()
  (DO
    (WRITE D 13 1)
    (TONE 11 750 300)
    (DELAY 300)
    (WRITE D 13 0)
    (DELAY 100)))

(DEFUN LETTER-S ()
  (DO (DOT) (DOT) (DOT) (DELAY 200)))

(DEFUN LETTER-O ()
  (DO (DASH) (DASH) (DASH) (DELAY 200)))

(SET i 0)
(WHILE (< i 3)
  (DO
    (LOG "SOS")
    (LETTER-S)
    (LETTER-O)
    (LETTER-S)
    (DELAY 700)
    (SET i (+ i 1))))

```

Project 7: IR Remote Control Robot

Control a robot with an IR remote

Hardware:

Arduino Uno - IR receiver module (VS1838B) - L298N motor driver - 2 DC motors with wheels - Chassis

Wiring:

IR Receiver:

VS --- 5V
GND --- GND
DATA --- D2

L298N:

ENA --- D5 (PWM)
IN1 --- D6
IN2 --- D7
IN3 --- D8
IN4 --- D9
ENB --- D10 (PWM)
12V --- Battery+
GND --- Battery- --- Arduino GND

Code:

```

; IR Remote Control Robot
; Control forward/backward/turn with IR remote codes

(SET irPin 2)
(SET ena 5)
(SET in1 6)
(SET in2 7)
(SET in3 8)
(SET in4 9)
(SET enb 10)

(DEFUN MOTOR-A-FORWARD ()
  (DO (WRITE D in1 1) (WRITE D in2 0) (WRITE A ena 200)))

(DEFUN MOTOR-A-BACKWARD ()
  (DO (WRITE D in1 0) (WRITE D in2 1) (WRITE A ena 200)))

(DEFUN MOTOR-B-FORWARD ()
  (DO (WRITE D in3 1) (WRITE D in4 0) (WRITE A enb 200)))

(DEFUN MOTOR-B-BACKWARD ()
  (DO (WRITE D in3 0) (WRITE D in4 1) (WRITE A enb 200)))

(DEFUN MOTORS-STOP ()
  (DO (WRITE A ena 0) (WRITE A enb 0)))

(DEFUN FORWARD ()
  (DO (MOTOR-A-FORWARD) (MOTOR-B-FORWARD)))

(DEFUN BACKWARD ()
  (DO (MOTOR-A-BACKWARD) (MOTOR-B-BACKWARD)))

(DEFUN TURN-LEFT ()
  (DO (MOTOR-A-BACKWARD) (MOTOR-B-FORWARD)))

(DEFUN TURN-RIGHT ()
  (DO (MOTOR-A-FORWARD) (MOTOR-B-BACKWARD)))

; Main loop -- read IR and respond
(WHILE (== 1 1)
  (DO
    (SET code (READ-IR irPin))
    (IF (== code 0x18) (FORWARD))
    (IF (== code 0x52) (BACKWARD))
    (IF (== code 0x10) (TURN-LEFT))
    (IF (== code 0x11) (TURN-RIGHT))
    (IF (== code 0x45) (MOTORS-STOP))
    (DELAY 100)))

```

Project 8: Soil Moisture Monitor

Monitor soil moisture and water plants automatically

Hardware:

Arduino Uno - Soil moisture sensor (capacitive or resistive) - Relay module - Water pump - 1N4007 diode

Wiring:

Soil Sensor:

VCC --- 5V
 GND --- GND
 AOUT --- A0

Relay:

IN --- D7
 VCC --- 5V
 GND --- GND
 COM --- Battery+
 NO --- Pump+

Pump:

Pump- --- Battery-
 Flyback diode across pump terminals

Code:

```
; Soil Moisture Monitor
; Waters plants when soil is dry

(SET sensorPin A0)
(SET relayPin 7)
(SET dryThreshold 600)
(SET wetThreshold 400)
(SET wateringTime 3000)

(LOG "Soil Moisture Monitor Active")

(WHILE (== 1 1)
  (DO
    (SET moisture (READ A sensorPin))
    (LOG "Soil moisture: " moisture)

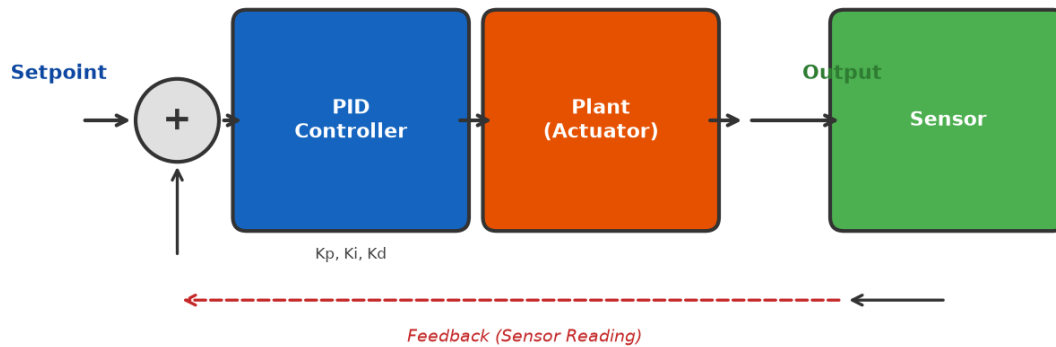
    (IF (> moisture dryThreshold)
      (DO
        (LOG "Soil is dry -- watering...")
        (WRITE D relayPin 0) ; Active-low relay ON
        (DELAY wateringTime)
        (WRITE D relayPin 1) ; Relay OFF
        (LOG "Watering complete")
        (DELAY 60000) ; Wait 1 minute before next check
        (IF (< moisture wetThreshold)
          (LOG "Soil is wet enough")))

    (DELAY 5000)))
```

Project 9: PID Temperature Controller

intain a target temperature using PID contro

PID Control Block Diagram



PID Control Block Diagram

Figure 14: PID control block diagram -- Setpoint, Error computation, PID controller, Plant (actuator), Sensor feedback

Hardware:

Arduino Uno - LM35 temperature sensor - Servo motor (as heater actuator) - LED (as status indicator)

Wiring:

```

LM35: Pin 1 --- 5V, Pin 2 --- GND, Pin 3 --- A0
Servo: Orange --- D9, Red --- 5V, Brown --- GND
LED: D13 --- [330 ohm] --- LED --- GND
  
```

Code:

```

; PID Temperature Controller
; Uses PID algorithm to maintain target temperature

(PID-INIT 0 A0 9 150 10 5) ; Channel 0: A0 in, D9 out, Kp=150, Ki=10, Kd=5
(CLOSED-LOOP 0 250) ; Target: 250 ADC units (approx 25C)

(LOG "PID Controller Active")
(LOG "Target: 25 C")

(WHILE (== 1 1)
  (DO
    (SET reading (FIRST (LM35 A0)))
    (LOG "Current temp: " reading " C")
    (DELAY 1000)))
  
```

Project 10: Multi-Sensor Data Logger

Log data from multiple sensors to the serial monitor

Hardware:

Arduino Uno - LM35 temperature sensor (A0) - LDR light sensor (A1) - DHT22 temperature/humidity sensor (D4) - HC-SR04 ultrasonic sensor (D7, D8)

Wiring:

```

LM35: 5V, GND, A0
LDR: 5V ---[LDR]---+--- A1, [10k ohm]--- GND
DHT22: 5V, GND, D4 ---[10k]--- 5V
HC-SR04: 5V, GND, D7 (TRIG), D8 (ECHO)

```

Code:

```

; Multi-Sensor Data Logger
; Logs temperature, light, humidity, and distance

(SET dhtPin 4)
(SET trigPin 7)
(SET echoPin 8)

(LOG "=== Multi-Sensor Data Logger ===")
(LOG "Timestamp | Temp | Light | Humidity | Distance")
(LOG "-----")

(SET count 0)

(WHILE (== 1 1)
  (DO
    (SET temp (LM35 A0))
    (SET light (READ A 1))
    (SET dhtReading (DHT-READ dhtPin 22))
    (SET humidity (SECOND dhtReading))
    (SET dist (ULTRASONIC trigPin echoPin))

    (LOG count " | " temp "C | " light " | " humidity "%" | " dist "cm")

    (SET count (+ count 1))
    (DELAY 5000))

```

17. Troubleshooting & FAQ

Serial / Hardware Issues

Problem: No serial port detected**s**`hil-cli ports`

o scan available ports. If no ports appear, check that your Arduino is connected and the drivers are installed. On Windows, you may need to install the CH340 or FTDI driver depending on your Arduino clon

Problem: "Monitor not running on port 9000"**h**`send``flash`**an**

stop

ommands talk to a running monitor. Start the monitor first wit

`hil-cli run`

n another termina

Problem: Script runs but nothing happens

erify the monitor is connected to the correct port. Us

`hil-cli ports`

o identify the right COM port. Check that the baud rate matches (default 9600

Problem: Garbled characters in serial monitor

IL scripts must be saved as UTF-8. Re-save your file without BOM. Ensure the baud rate matches between the monitor and the serial terminal (default 9600

CRUMB Issues

Problem: Generated circuit opens but Nano does nothing

ress the RUN button in CRUMB. Open the serial monitor (baud 9600). Check for error messages in the serial outpu

Problem: Buzzer not sounding

nsure you are using a passive buzzer on pin D11. Active buzzers just set the pin HIGH silently -

`tone()`

nly works with passive buzzer

Problem: `--strict` fails

he script uses an op that produces a warning. Review the warnings and either remove the offending ops or dro

`--strict`

Problem: CRUMB can't find the simulator

e

`HIL_CRUMB_DIR`

o the directory containin

`CRUMBWPF.exe`

or us

`--crumb-dir`

n the command lin

Firmware Issues

Problem: TinyGo not found

nstall TinyGo from <https://tinygo.org/getting-started/>. This is required to compile and flash the firmwar

Problem: avrdude not found

nstall the Arduino IDE, which bundles avrdude and avr-gcc. Alternatively, install avrdude separatel

Problem: Wrong board

nsure you specify the correct board wit

`--board uno`

`--board nano`

O

--board mega

Language Issues

Problem: "Unknown symbol" warning

variables must be created wit

SET

efore use. Check for typos and case sensitivity. Remember tha

pin

n

PIN

re different variable

Problem: "INCLUDE cycle detected"

n include chain loops back on itself. Check your include graph. If A includes B and B includes A, this is a cycl

Problem: Loop hangs forever

ver

WHILE

oop must contain

DELAY

r a condition that eventually becomes false. Without this, the loop runs as fast as the hardware allows and the system becomes unresponsiv

Problem: Dividing by zero

IL-OS returns 0 for division by zero (safe, no crash). But the result may not be what you expect. Always check your denominator

General FAQ

Q: Can I use HIL-OS without an Arduino?

: Yes! Us

hil-cli simulate

or headless execution with no hardware. All hardware commands return

Q: How do I add my own macros?

: Create

.hil

ile in you

lib/

irectory and us

(INCLUDE "lib/mylib.hil")

n your main scrip

Q: Can I use multiple sensors at once?

: Yes! Read sensors sequentially in a loop. The Arduino processes one command at a time, so you cannot read two sensors simultaneousl

Q: How do I connect to WiFi?

: HIL-OS does not currently support WiFi modules. For WiFi projects, use the Node.js API to bridge between HIL-OS and a WiFi-enabled device

Q: Can I use HIL-OS with Raspberry Pi?

: The host tools (CLI, Node.js API) work on any platform. The firmware targets Arduino/AVR boards. For Raspberry Pi GPIO, you would need to write a custom firmwar

18. Appendix

A. Complete LISP Operator Reference

Operator	Syntax	Description
SET	<code>`(SET var value)`</code>	Create or update a variable
DO	<code>`(DO expr1 expr2 ...)`</code>	Sequence of operations
IF	<code>`(IF cond then else)`</code>	Conditional branching
WHILE	<code>`(WHILE cond body...)`</code>	Loop while condition is true
AWAIT	<code>`(AWAIT cond timeout)`</code>	Poll with timeout (returns 0 or 1)
EXIT	<code>`(EXIT)`</code>	Halt execution
DELAY	<code>`(DELAY ms)`</code>	Pause for milliseconds
LOG	<code>`(LOG arg1 arg2 ...)`</code>	Print to serial monitor
DEFUN	<code>`(DEFUN name (args) body)`</code>	Define a macro
INCLUDE	<code>`(INCLUDE "path")`</code>	Load another file
STRUCT	<code>`(STRUCT name (fields))`</code>	Define a struct
NEW	<code>`(NEW var Type vals...)`</code>	Create struct instance
FIRST	<code>`(FIRST list)`</code>	Get first element of tuple
SECOND	<code>`(SECOND list)`</code>	Get second element of tuple
WRITE D	<code>`(WRITE D pin val)`</code>	Digital write
WRITE A	<code>`(WRITE A pin val)`</code>	Analog write (PWM)
WRITE P	<code>`(WRITE P pin val)`</code>	PWM alias
READ D	<code>`(READ D pin)`</code>	Digital read
READ A	<code>`(READ A pin)`</code>	Analog read
TOGGLE	<code>`(TOGGLE pin)`</code>	Toggle digital pin
TONE	<code>`(TONE pin freq dur)`</code>	Play tone
PULSE	<code>`(PULSE pin count delay)`</code>	Generate pulses
BLINK	<code>`(BLINK pin count delay)`</code>	Blink LED
MAP	<code>`(MAP val inMin inMax outMin outMax)`</code>	Remap range
CONSTRAIN	<code>`(CONSTRAIN val min max)`</code>	Clamp value
RANDOM	<code>`(RANDOM min max)`</code>	Random integer
LM35	<code>`(LM35 pin)`</code>	Read LM35 temperature
ULTRASONIC	<code>`(ULTRASONIC trig echo)`</code>	Read ultrasonic distance
DHT-READ	<code>`(DHT-READ pin model)`</code>	Read DHT sensor

V-SERVO	`(V-SERVO pin angle)`	Set servo angle
I2C-WRITE	`(I2C-WRITE addr reg val)`	I2C write
I2C-READ	`(I2C-READ addr reg)`	I2C read
LCD-INIT	`(LCD-INIT rs en d4 d5 d6 d7)`	Initialize LCD
LCD-CLEAR	`(LCD-CLEAR)`	Clear LCD
LCD-PRINT	`(LCD-PRINT row col text)`	Print to LCD
PID-INIT	`(PID-INIT ch inPin outPin kp ki kd)`	Initialize PID
CLOSED-LOOP	`(CLOSED-LOOP ch setpoint)`	Enable PID
OPEN-LOOP	`(OPEN-LOOP ch)`	Disable PID
NEO-SET	`(NEO-SET px r g b)`	Set NeoPixel color
NEO-SHOW	`(NEO-SHOW)`	Update NeoPixels
MP3-PLAY	`(MP3-PLAY pin track)`	Play MP3
MP3-VOL	`(MP3-PLAY pin vol)`	Set MP3 volume
SD-WRITE	`(SD-WRITE pin data)`	Write to SD
RTC-READ	`(RTC-READ)`	Read RTC time
RTC-WRITE	`(RTC-WRITE s m h)`	Set RTC time
ON-EVENT	`(ON-EVENT pin edge macro)`	Arm digital interrupt
ON-CHANGE	`(ON-CHANGE pin thresh macro)`	Arm analog change
ATTENTION	`(ATTENTION pin)`	High-freq polling
BATCH-BEGIN	`(BATCH-BEGIN)`	Start batch mode
BATCH-END	`(BATCH-END)`	End batch mode
AUTOPILOT-BEGIN	`(AUTOPILOT-BEGIN)`	Start recording
AUTOPILOT-END	`(AUTOPILOT-END)`	Commit recording

B. Arduino Uno Quick Reference Card

```

+-----+
|           Arduino Uno Quick Reference           |
+-----+
| Digital Pins: D0-D13 (20mA max each)             |
| PWM Pins: D3, D5, D6, D9, D10, D11 (0-255)       |
| Analog Pins: A0-A5 (0-1023, 10-bit ADC)          |
| I2C: A4 (SDA), A5 (SCL)                          |
| SPI: D10 (SS), D11 (MOSI), D12 (MISO), D13 (SCK)|
| Interrupts: D2 (INT0), D3 (INT1)                 |
+-----+
| Power: 5V (500mA), 3.3V (50mA), VIN (7-12V)      |
| Flash: 32KB, SRAM: 2KB, EEPROM: 1KB, Clock: 16MHz|
+-----+
| HIL-OS Pin Mapping:                             |
| WRITE D 13 1 -> digitalWrite(13, HIGH)          |
| WRITE A 9 128 -> analogWrite(9, 128)             |
| READ A 0      -> analogRead(A0)                  |
| READ D 2      -> digitalRead(2)                  |
| TONE 11 440 200 -> tone(11, 440, 200)            |
+-----+

```

C. Sensor Wiring Quick Reference

```
+-----+
|           Sensor Wiring Quick Reference           |
+-----+
| LM35:      Vs->5V  GND->GND  Out->A0                |
| DHT22:     VCC->5V  Data->D4+10k pullup  GND->GND    |
| HC-SR04:   VCC->5V  Trig->D7  Echo->D8  GND->GND    |
| LDR:       5V--[LDR]--+--A0  [10k]--GND           |
| Button:    5V--[Btn]--D2  [10k]--GND (pull-down)   |
|            or D2--[Btn]--GND (internal pull-up)    |
| Servo:     Red->5V  Brown->GND  Orange->D9         |
| Buzzer:    +->D11  -->GND (passive only)          |
| LED:       D13--[330 ohm]--Anode-->Cathode-->GND  |
| Relay:     IN->D7  VCC->5V  GND->GND               |
+-----+
```

D. Error Messages Reference

Message	Cause	Solution
`Unknown symbol: X`	Variable used before SET	Add `(SET X value)` before use
`INCLUDE cycle detected`	Circular include	Remove circular dependency
`Parse error at line N`	Syntax error	Check parentheses and arguments
`Monitor not running`	No monitor process	Start with `hil-cli run` first
`Port busy`	Port in use by another app	Close other serial terminals
`CRC mismatch`	Corrupted serial data	Check cable connections, reduce baud rate
`TinyGo not found`	TinyGo not installed	Install from tinygo.org
`avrdude not found`	avrdude not installed	Install Arduino IDE
`CRUMB not found`	CRUMB not installed	Set HIL_CRUMB_DIR or install CRUMB