ARDUINO KIT
MASTER HANDBOOK

A Complete Hardware & Software Guide
From First Blink to Capstone Build — 2nd Edition, Revised & Expanded

18 guided projects • full wiring tables • ready-to-flash code • troubleshooting guide

How To Use This Handbook

Read this first — it will save you a re-wire or two

This handbook is written specifically for the components in the Ultimate Arduino Starter Kit. Every project below lists the exact parts used, a plain-English wiring table, a copy-paste-ready sketch, and the most common mistake beginners make with that component.

Before you start:

- Install the free Arduino IDE from arduino.cc and select **Tools** → **Board** → **Arduino Uno**.
- Each project is self-contained — pin numbers are reused between chapters, so rewire fully before starting a new project rather than adding to the previous circuit.
- Where a project needs a library beyond the built-in ones, it is listed in a blue ‘Libraries’ box right under the heading.
- Red warning boxes flag steps that can damage a component or the board if skipped — don't skip them.

CHANGES IN THIS EDITION: Corrected the LED current guidance, added missing library call-outs to every project that needs one, fixed the capstone wiring description so it matches the code, and added a full troubleshooting matrix at the back.

Arduino UNO Layout & Mechanics

- **ATmega328P Microcontroller** — an 8-bit processor running at 16 MHz with 32KB of flash storage (roughly 0.5KB of that is reserved for the bootloader, leaving about 31.5KB for your own sketch).
- **Digital I/O (Pins 0–13)** — binary pins (HIGH = 5V, LOW = 0V). Pins marked with a tilde (~3, 5, 6, 9, 10, 11) support 8-bit PWM (Pulse Width Modulation) output, which lets them fake an analog voltage by rapidly switching on and off.
- **Analog Inputs (A0–A5)** — connected to an internal 10-bit Analog-to-Digital Converter (ADC) that returns values from 0 (0V) to 1023 (5V).

Breadboard Internal Routing

Breadboards allow circuit assembly without soldering. Rails run in two distinct patterns:

- **Power rails (+ / –)** — the striped columns running vertically along the edges are connected in long vertical lines, top to bottom.
- **Terminal strips** — the main working area is horizontal rows of five interconnected holes (rows A–E and F–J). A break down the centre divider electrically separates the left and right halves — this is where an IC or module straddles the gap.

Resistor Colour Code Quick Reference

Colour	1st Digit	2nd Digit	Multiplier	Tolerance
Black	0	0	×1	—
Brown	1	1	×10	±1%
Red	2	2	×100	±2%

Colour	1st Digit	2nd Digit	Multiplier	Tolerance
Orange	3	3	×1k	—
Yellow	4	4	×10k	—
Green	5	5	×100k	±0.5%
Blue	6	6	×1M	—
Gold	—	—	×0.1	±5%

OHM'S LAW (V = I × R): Always place a current-limiting resistor (220Ω to 1kΩ) in series with a standard LED. A bare LED across 5V will draw far more than its rated current and burn out almost instantly. Keep continuous current per digital pin to about 20mA — 40mA is the absolute maximum the ATmega328P datasheet allows, not a safe operating target.

ULTIMATE ARDUINO STARTER KIT

Every part used in this handbook, in one box — nothing extra to buy

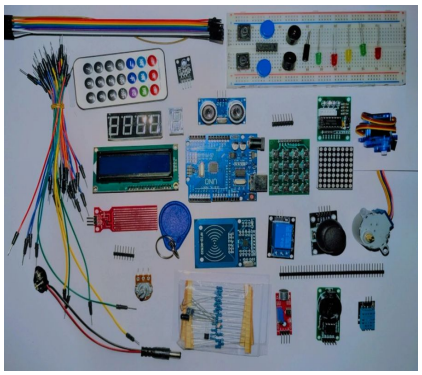
£25

+ shipping · UK & international orders

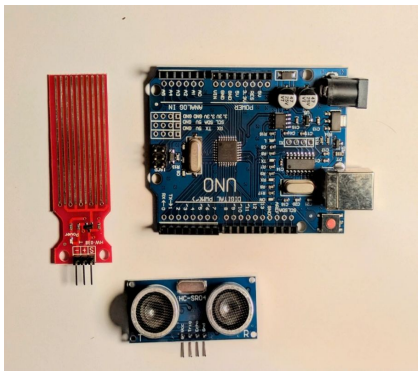
GET YOUR KIT AT
ELECTRONICMAKER.NETLIFY.APP

Secure checkout · ships within 2 business days

Kit Storage Box
Complete component organiser



Full Component Overview
38+ hardware components



UNO Board & Sensors
Arduino UNO board & modules

COMPLETE KIT INVENTORY (WHAT'S INCLUDED)

- ✓ 1 × Arduino Compatible Board
- ✓ 1 × Solderless Breadboard
- ✓ 1 × Female-to-Male Dupont
- ✓ 1 × 74HC595 Shift Register
- ✓ 1 × IR Flame Sensor
- ✓ 1 × Tactile Key Button
- ✓ 1 × 8x8 Dot Matrix Module
- ✓ 1 × 28BYJ-48 Stepper Motor
- ✓ 1 × XY Dual-Axis Joystick
- ✓ 1 × RFID RC522 Reader
- ✓ 1 × Microphone Sound Sensor
- ✓ 1 × 4x4 Keypad Matrix
- ✓ 1 × HC-SR04 Ultrasonic Sensor
- ✓ 1 × USB Cable
- ✓ 5 × LED Light Set
- ✓ 1 × Rotary Potentiometer
- ✓ 1 × Infrared Receiver
- ✓ 1 × Ball/Tilt Switch
- ✓ 1 × IR Remote Control
- ✓ 1 × 1-Digit Display Tube
- ✓ 1 × 9g Micro Servo Motor
- ✓ 1 × Temperature Module
- ✓ 1 × RFID Blue Keychain
- ✓ 1 × 5V Relay Switch Module
- ✓ 1 × RGB 3-Colour Module
- ✓ 1 × Jump Cable Set
- ✓ 1 Pack Resistor Variety
- ✓ 1 × Buzzer Module
- ✓ 1 × DS18B20 Temp Sensor
- ✓ 1 × Photoresistor (LDR)
- ✓ 1 × 4-Digit Display Tube
- ✓ 1 × Stepper Driver Board
- ✓ 1 × 1602 LCD Display
- ✓ 1 × Water Level Test Module
- ✓ 1 × RFID White Card
- ✓ 1 × DS1302 Real-Time Clock
- ✓ 1 × 9V Battery Snap Connector

Every project in this handbook — including the Chapter 5 capstone — uses only parts from this box. Nothing extra to source before you start.

Contents

Jump straight to the project you need

CHAPTER 1

Chapter 1: Foundations

Projects 1.1 & 1.2 — digital output, analog input, and timing

Project 1.1: Standard LED Blink & Traffic Light System

Objective: *Master digital outputs, control flow, and accurate timing using digitalWrite() and delay().*

Component	Arduino Pin	Connection Details
Red LED	Digital Pin 12	Anode (+) via 220Ω resistor to Pin 12, Cathode (–) to GND
Yellow LED	Digital Pin 11	Anode (+) via 220Ω resistor to Pin 11, Cathode (–) to GND
Green LED	Digital Pin 10	Anode (+) via 220Ω resistor to Pin 10, Cathode (–) to GND

```
CIRCUIT SCHEMATIC:
[Pin 12] ---> [220Ω Resistor] ---> (Anode) RED LED (Cathode) ---> GND
[Pin 11] ---> [220Ω Resistor] ---> (Anode) YEL LED (Cathode) ---> GND
[Pin 10] ---> [220Ω Resistor] ---> (Anode) GRN LED (Cathode) ---> GND
```

Project 1.1: Interactive Traffic Light Sequencer

```
const int RED_PIN = 12;
const int YEL_PIN = 11;
const int GRN_PIN = 10;

void setup() {
  pinMode(RED_PIN, OUTPUT);
  pinMode(YEL_PIN, OUTPUT);
  pinMode(GRN_PIN, OUTPUT);
}

void loop() {
  digitalWrite(RED_PIN, HIGH); delay(3000); // Red light ON for 3s
  digitalWrite(RED_PIN, LOW);
  digitalWrite(GRN_PIN, HIGH); delay(3000); // Green light ON for 3s
  digitalWrite(GRN_PIN, LOW);
  digitalWrite(YEL_PIN, HIGH); delay(1000); // Yellow light ON for 1s
  digitalWrite(YEL_PIN, LOW);
}
```

Code explanation: pinMode() configures each target pin as a voltage source rather than a sensor input. digitalWrite(pin, HIGH) then delivers 5V on that pin, while LOW connects it to 0V ground. Because delay() pauses the entire sketch, all three LEDs necessarily change one at a time — later projects that need to do two things at once (like project 3.3) avoid delay() for exactly this reason.

Project 1.2: Potentiometer Dimmer & Button Tone Generator

Objective: *Learn analog input reading using the 10-bit ADC, PWM brightness control, and driving tone frequencies on a buzzer.*

Component / Pin	Arduino Connection	Function / Notes
Potentiometer Pin 1 (outer)	5V	Power rail reference
Potentiometer Pin 2 (centre)	Analog Pin A0	Variable voltage-divider output (0–5V)

Component / Pin	Arduino Connection	Function / Notes
Potentiometer Pin 3 (outer)	GND	Ground reference
Push Button Leg 1	Digital Pin 2	Uses internal INPUT_PULLUP resistor
Push Button Leg 2	GND	Shorts pin to ground when pressed
Buzzer (+) Leg	Digital Pin 8	Square-wave frequency output

```
CIRCUIT SCHEMATIC:
5V ---> [ Potentiometer Outer ]
A0 <--- [ Pot Pin Centre Wiper ]
GND ---> [ Potentiometer Outer ]
Pin 2 ---> [ Push Button ] ---> GND
Pin 8 ---> [ Buzzer (+) ] ---> [ Buzzer (–) ] ---> GND
```

Project 1.2: Analog Reading, PWM Dimming & Buzzer Control

```
const int POT_PIN = A0;
const int BTN_PIN = 2;
const int BUZZER_PIN = 8;
const int LED_PWM = 9;

void setup() {
  pinMode(BTN_PIN, INPUT_PULLUP); // Enables internal ~20k pull-up resistor
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(LED_PWM, OUTPUT);
}

void loop() {
  int rawAnalog = analogRead(POT_PIN); // Returns 0 to 1023
  int brightness = map(rawAnalog, 0, 1023, 0, 255); // Scales to 8-bit PWM
  analogWrite(LED_PWM, brightness);

  if (digitalRead(BTN_PIN) == LOW) { // Button reads LOW when pressed
    tone(BUZZER_PIN, 1000, 100); // Play a 1000Hz tone for 100ms
    delay(100);
  }
}
```

TROUBLESHOOTING: If the push button seems to trigger on its own without being pressed, double-check that INPUT_PULLUP is set in software. Without it, the pin ‘floats’ between HIGH and LOW and picks up electrical noise; INPUT_PULLUP holds it firmly HIGH until the button ties it to ground.

CHAPTER 2

Chapter 2: Sensors & Environmental Inputs

Projects 2.1–2.4 — reading the world around the board

Project 2.1: Photoresistor (LDR) Automatic Nightlight

Objective: *Build an automatic lighting system using a light-dependent resistor in a voltage-divider circuit.*

```
VOLTAGE DIVIDER SCHEMATIC:
5V ---> [ LDR Photoresistor ] ---> Node (A1) ---> [ 10kΩ Resistor ] ---> GND
|
To Pin A1
```

Project 2.1: LDR Automatic Nightlight

```
const int LDR_PIN = A1;
const int NIGHT_LED = 13;
const int LIGHT_THRESHOLD = 400; // Adjust based on room lighting

void setup() {
  pinMode(NIGHT_LED, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  int lightLevel = analogRead(LDR_PIN);
  Serial.print("Current Light Level: ");
  Serial.println(lightLevel);

  if (lightLevel < LIGHT_THRESHOLD) {
    digitalWrite(NIGHT_LED, HIGH); // Turn light ON when dark
  } else {
    digitalWrite(NIGHT_LED, LOW); // Turn light OFF during day
  }
  delay(200);
}
```

Calibration tip: Open the Serial Monitor (Tools → Serial Monitor, 9600 baud) and note the printed value with the room lights on and again with them off. Set LIGHT_THRESHOLD roughly halfway between the two for a reliable switch-over.

Project 2.2: Ball Switch Vibration & Tilt Security Alarm

The ball/tilt switch contains a small metal roller that bridges two internal contacts when held vertically. Shaking or tilting the switch breaks that contact.

Project 2.2: Tilt Sensor Security Trigger

```
const int TILT_PIN = 3;
const int ALARM_BUZZER = 8;

void setup() {
  pinMode(TILT_PIN, INPUT_PULLUP);
  pinMode(ALARM_BUZZER, OUTPUT);
}

void loop() {
  int tiltState = digitalRead(TILT_PIN);
  if (tiltState == HIGH) { // Sensor tilted / contact opened
    tone(ALARM_BUZZER, 2400, 200);
    delay(250);
  }
}
```

Project 2.3: Flame Sensor Safety Detector

***Objective:** Detect infrared wavelength emissions from open flames (roughly 760–1100nm) using an NPN phototransistor detector.*

Flame Sensor Pin	Arduino Connection	Signal Notes
Long leg (cathode)	5V supply voltage	—
Short leg (anode)	Analog Pin A2, plus 10kΩ to GND	Higher reading = stronger IR / flame signal detected

Project 2.3: Infrared Flame Detector

```
const int FLAME_PIN = A2;
const int ALARM_LED = 12;

void setup() {
  pinMode(ALARM_LED, OUTPUT);
}

void loop() {
  int flameVal = analogRead(FLAME_PIN);
  if (flameVal > 600) { // High IR radiation present
    digitalWrite(ALARM_LED, HIGH);
  } else {
    digitalWrite(ALARM_LED, LOW);
  }
  delay(100);
}
```

SAFETY NOTE: This module is a teaching demonstration, not a certified smoke or fire alarm — never rely on it as a substitute for a proper household smoke detector.

Project 2.4: DS18B20 1-Wire Digital Thermometer

The DS18B20 reads ambient temperature with up to 12-bit precision over the 1-Wire protocol. A 4.7kΩ pull-up resistor is required between the Data and VCC lines — without it the sensor cannot pull the data line high and every reading will fail.

```
DS18B20 PINOUT (flat side facing you):
Pin 1: GND | Pin 2: Data (DQ) | Pin 3: VCC (5V)
|
[ 4.7kΩ Resistor ]
|
5V
```

LIBRARIES: Library Manager installs needed: OneWire, DallasTemperature (Sketch → Include Library → Manage Libraries in the Arduino IDE).

Project 2.4: DS18B20 OneWire Protocol Reading

```
#include <OneWire.h>
#include <DallasTemperature.h>

#define ONE_WIRE_BUS 4
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

void setup() {
  Serial.begin(9600);
  sensors.begin();
}

void loop() {
  sensors.requestTemperatures();
  float tempC = sensors.getTempCByIndex(0);
  Serial.print("Temperature: "); Serial.print(tempC); Serial.println(" C");
  delay(1000);
}
```

CHAPTER 3

Chapter 3: Displays & Output Expansion

Projects 3.1–3.5 — driving displays with limited pins

Project 3.1: 74HC595 Shift Register Expansion

Objective: *Convert serial data into eight parallel output pins using just three digital lines from the microcontroller.*

74HC595 Pin	Pin Name	Arduino Connection
Pin 14	SER / DS (Data)	Digital Pin 4
Pin 12	RCLK (Latch)	Digital Pin 5
Pin 11	SRCLK (Clock)	Digital Pin 6
Pin 16 / 8	VCC / GND	5V / GND rail
Pin 10 / 13	SRCLR / OE	Pin 10 to 5V (disables clear) / Pin 13 to GND (enables output)

Project 3.1: Shift Register 8-Bit Counter

```
const int DATA_PIN = 4;
const int LATCH_PIN = 5;
const int CLOCK_PIN = 6;

void setup() {
  pinMode(DATA_PIN, OUTPUT);
  pinMode(LATCH_PIN, OUTPUT);
  pinMode(CLOCK_PIN, OUTPUT);
}

void loop() {
  for (int i = 0; i < 256; i++) {
    digitalWrite(LATCH_PIN, LOW); // Hold output steady while shifting
    shiftOut(DATA_PIN, CLOCK_PIN, MSBFIRST, i); // Clock out 8 bits
    digitalWrite(LATCH_PIN, HIGH); // Update output pins
    delay(100);
  }
}
```

Project 3.2: 1-Digit 7-Segment Display Counter

By mapping segments a–g to outputs Q0–Q6 of the 74HC595 from Project 3.1, digits 0–9 can be drawn efficiently using binary bitmasks — reuse the wiring above and send one byte per digit instead of counting up.

Project 3.3: 4-Digit 7-Segment Multiplexed Clock Display

Objective: *Drive a 4-digit display through high-speed persistence-of-vision (POV) multiplexing.*

```
MULTIPLEXING ARCHITECTURE:
[74HC595 Data Bus (a-g)] ================> ALL 4 DIGIT SEGMENTS
Arduino Pins D7, D8, D9, D10 ----------> Digit common cathodes 1, 2, 3, 4
(Rapidly turns individual digits ON/OFF in sequence, every 2–5 milliseconds)
```

Project 3.3: POV Digit Multiplexing (excerpt)

```
byte digitPins[4] = {7, 8, 9, 10};
byte digitCodes[10] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};

void displayDigit(int digitIndex, int value) {
  for (int i = 0; i < 4; i++) digitalWrite(digitPins[i], HIGH); // Turn off all digits
  digitalWrite(LATCH_PIN, LOW);
  shiftOut(DATA_PIN, CLOCK_PIN, MSBFIRST, digitCodes[value]);
  digitalWrite(LATCH_PIN, HIGH);
  digitalWrite(digitPins[digitIndex], LOW); // Turn on the active digit's cathode
  delay(3); // Wait for human-eye persistence
}
```

> **NOTE:** This excerpt reuses DATA_PIN / LATCH_PIN / CLOCK_PIN from Project 3.1 — declare them the same way (pins 4, 5 and 6) before calling displayDigit() in a loop.

Project 3.4: 8x8 LED Matrix Graphic Display

An 8x8 dot matrix is 64 LEDs wired at the intersections of 8 rows and 8 columns. Raster scanning lights one row at a time with a column bit pattern, fast enough that the whole 8x8 image appears to glow at once.

Project 3.4: Displaying a Smiley Face Pattern

```
byte smiley[8] = {
  0b00111100,
  0b01000010,
  0b10100101,
  0b10000001,
  0b10100101,
  0b10011001,
  0b01000010,
  0b00111100
};
// A raster routine scans row by row using dual shift registers or spare I/O pins.
```

Project 3.5: 1602 LCD Parallel Interface Configuration

Objective: *Interface a HD44780-compatible 16x2 character LCD using a 4-bit parallel data bus, no I2C backpack required.*

LCD Pin	Function	Arduino Connection
Pin 1 (VSS) / Pin 2 (VDD)	Ground / 5V power	GND / 5V rail
Pin 3 (V0)	Display contrast	Centre wiper of potentiometer (0–5V)
Pin 4 (RS) / Pin 5 (RW)	Register Select / Read-Write	Pin 12 / Ground (write mode)
Pin 6 (E)	Enable clock pin	Pin 11
Pins 11–14 (D4–D7)	4-bit data bus	Pins 5, 4, 3, 2
Pin 15 (LED+) / Pin 16 (LED–)	Backlight power	5V (via 220Ω resistor) / GND

LIBRARIES: Library Manager installs needed: LiquidCrystal (bundled with the Arduino IDE) (Sketch → Include Library → Manage Libraries in the Arduino IDE).

Project 3.5: Direct 4-Bit Parallel LCD Driver

```
#include <LiquidCrystal.h>

// Initialise the library with the interface pin assignments
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // RS, E, D4, D5, D6, D7

void setup() {
  lcd.begin(16, 2);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Arduino Kit Manual");
  lcd.setCursor(0, 1);
  lcd.print("Status: READY");
}

void loop() {
  lcd.setCursor(12, 1);
  lcd.print(millis() / 1000); // Display live uptime counter
}
```

IMPORTANT: Adjust the contrast potentiometer on LCD Pin 3 if characters appear invisible or as solid white blocks — this is the single most common ‘broken’ LCD report and is almost always just contrast.

CHAPTER 4

Chapter 4: Motion & Remote Control

Projects 4.1–4.3 — servos, steppers, and infrared

Project 4.1: 9g Servo Motor Angle Position Controller

Objective: *Control positional rotation (0° to 180°) using the 50Hz PWM control signal generated by the Servo library.*

Servo Wire Colour	Function	Arduino Connection
Brown / Black	Ground	GND rail
Red	5V VCC	5V rail
Orange / Yellow	PWM control signal	Digital Pin 9

LIBRARIES: Library Manager installs needed: Servo (bundled with the Arduino IDE) (Sketch → Include Library → Manage Libraries in the Arduino IDE).

Project 4.1: Potentiometer-Controlled Servo Positioner

```
#include <Servo.h>
Servo myServo;
const int POT_PIN = A0;

void setup() {
  myServo.attach(9); // Attaches the servo to digital pin 9
}

void loop() {
  int rawVal = analogRead(POT_PIN);
  int angle = map(rawVal, 0, 1023, 0, 180);
  myServo.write(angle); // Drives the motor to the proportional angle
  delay(15);
}
```

POWER NOTE: A servo can briefly pull more current than the Arduino's 5V pin comfortably supplies, which is why it can reset the board mid-movement. See the troubleshooting appendix for the fix.

Project 4.2: Stepper Motor Driving via ULN2003 Driver Board

The 28BYJ-48 stepper motor provides high torque and precise steps by energising four internal coils in sequence through the Darlington transistor array on the ULN2003 board.

LIBRARIES: Library Manager installs needed: Stepper (bundled with the Arduino IDE) (Sketch → Include Library → Manage Libraries in the Arduino IDE).

Project 4.2: Stepper Motor Continuous Sweep

```
#include <Stepper.h>
const int STEPS_PER_REV = 2048; // Full-revolution resolution
Stepper myStepper(STEPS_PER_REV, 8, 10, 9, 11); // Driver IN1, IN3, IN2, IN4

void setup() {
  myStepper.setSpeed(10); // Set motor speed to 10 RPM
}

void loop() {
  myStepper.step(STEPS_PER_REV); // Clockwise full turn
  delay(500);
  myStepper.step(-STEPS_PER_REV); // Counter-clockwise turn
  delay(500);
}
```

WIRING NOTE: The pin order in the Stepper() constructor above is not a typo — the bundled Stepper library expects IN1, IN3, IN2, IN4 for a 4-wire unipolar motor, not the numeric IN1–IN4 order printed on the ULN2003 board.

Project 4.3: Infrared Receiver & Remote Command Processor

Objective: *Decode 38kHz-modulated infrared light signals to trigger hardware events wirelessly.*

IR Receiver Pin	Module Label	Arduino Connection
Pin 1	OUT / S (Signal)	Digital Pin 11
Pin 2	GND / –	GND
Pin 3	VCC / +	5V

LIBRARIES: Library Manager installs needed: IRremote (Sketch → Include Library → Manage Libraries in the Arduino IDE).

Project 4.3: IR Remote Decoder System

```
#include <IRremote.hpp>
const int IR_RECEIVE_PIN = 11;
const int RELAY_LED = 13;

void setup() {
  Serial.begin(9600);
  IrReceiver.begin(IR_RECEIVE_PIN, ENABLE_LED_FEEDBACK);
  pinMode(RELAY_LED, OUTPUT);
}

void loop() {
  if (IrReceiver.decode()) {
    unsigned long rawCode = IrReceiver.decodedIRData.decodedRawData;
    Serial.print("Hex Key Received: 0x");
    Serial.println(rawCode, HEX);

    // Toggle output on a specific IR remote key press
    if (IrReceiver.decodedIRData.command == 0x45) { // Example: POWER button
      digitalWrite(RELAY_LED, !digitalRead(RELAY_LED));
    }
    IrReceiver.resume(); // Ready to capture the next command
  }
}
```

TROUBLESHOOTING: Direct sunlight and incandescent bulbs both emit strong IR light that can saturate the receiver module. Keep IR projects away from windows and halogen/incandescent lamps.

CHAPTER 5

Chapter 5: Capstone System Integration

Smart Station — combining five projects into one build

Capstone Project: Integrated Smart Terminal & Environment Monitor

This build combines the 1602 LCD, DS18B20 temperature sensor, flame safety system, servo-actuated deadbolt, and IR remote into one automation hub. Wire it fresh on a clear breadboard rather than reusing an earlier project's layout.

```
SYSTEM ARCHITECTURE MATRIX:
DS18B20 Temp Sensor (Pin D4) ------------> Real-Time LCD Display Engine
1602 LCD (Pins D2, D3, D5, D6, D11, D12) -> Status &amp;amp; temperature readout
IR Receiver Module (Pin D7) -------------> Wireless Control Centre (Mode Switch)
9g Servo Actuator (Pin D9) --------------> Automated Security Lock
Flame Sensor (Pin A2) -------------------> Override Safety Alarm &amp;amp; Buzzer (Pin D8)
```

PIN NOTE: The LCD alone uses six pins (2, 3, 5, 6, 11, 12) in this build — different from its wiring in Project 3.5 — because pins 4, 9 and 7 are now needed for the temperature sensor, servo and IR receiver. Always match pins to the code for the specific project you're building, not the chapter it first appeared in.

LIBRARIES: Library Manager installs needed: OneWire, DallasTemperature, Servo, IRremote, LiquidCrystal (bundled) (Sketch → Include Library → Manage Libraries in the Arduino IDE).

Capstone Integrated System Code (Part 1 — Setup)

```
#include <LiquidCrystal.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <Servo.h>
#include <IRremote.hpp>

LiquidCrystal lcd(12, 11, 5, 3, 2, 6); // RS, E, D4, D5, D6, D7
OneWire oneWire(4);
DallasTemperature sensors(&oneWire);
Servo doorLock;

const int IR_PIN = 7;
const int BUZZER = 8;
const int SERVO_PIN = 9;
const int FLAME_PIN = A2;
bool systemLocked = true;

void setup() {
  lcd.begin(16, 2);
  sensors.begin();
  doorLock.attach(SERVO_PIN);
  IrReceiver.begin(IR_PIN);
  pinMode(BUZZER, OUTPUT);
  doorLock.write(0); // Locked position
}
```

Capstone Integrated System Code (Part 2 — Main Loop)

```
void loop() {
  // 1. Safety check: flame sensor override
  if (analogRead(FLAME_PIN) > 600) {
    lcd.clear();
    lcd.print("!! ALARM: FIRE !!");
    tone(BUZZER, 3000);
    doorLock.write(90); // Emergency door release
    delay(500);
    return;
  } else {
    noTone(BUZZER);
  }

  // 2. Read sensors
  sensors.requestTemperatures();
  float currentTemp = sensors.getTempCByIndex(0);

  // 3. Process IR inputs
  if (IrReceiver.decode()) {
    if (IrReceiver.decodedIRData.command == 0x1C) { // Lock/Unlock button
      systemLocked = !systemLocked;
      doorLock.write(systemLocked ? 0 : 90);
    }
    IrReceiver.resume();
  }

  // 4. Refresh dashboard LCD
  lcd.setCursor(0, 0);
  lcd.print("Temp: "); lcd.print(currentTemp, 1); lcd.print(" C");
  lcd.setCursor(0, 1);
  lcd.print("Lock: "); lcd.print(systemLocked ? "LOCKED " : "UNLOCKED ");
  delay(200);
}
```

BUILD TIP: The flame check runs first and uses **return** to skip the rest of the loop — this means the LCD and lock status freeze during an alarm, which is intentional so the fire warning isn't overwritten. If you want the lock and IR control to keep working during an alarm, remove the return statement and let the loop fall through.

Appendix

Complete Kit Troubleshooting Master Guide

Symptom	Probable Cause	Resolution Step
Code fails upload with ‘sync error’	Incorrect port or board selected	Check Tools → Board is set to ‘Arduino Uno’ and Tools → Port matches the listed COM/USB port.
LCD output shows solid white or black blocks	Contrast voltage incorrectly biased	Turn the contrast potentiometer on LCD Pin 3 slowly until characters appear.
Servo jitters or resets the board	Excessive current draw during movement	Power the servo from a separate 5V supply, or add a 100uF capacitor across its 5V/GND leads.
DS18B20 returns –127.0°C	Missing pull-up resistor or bad connection	Install a 4.7kΩ pull-up resistor between 5V and the DQ data line; re-check the breadboard connection.
IR receiver never triggers, or triggers randomly	Sunlight/incandescent light interference, or wrong pin in code	Move the project away from direct light sources; confirm IR_RECEIVE_PIN matches the wired signal pin.
LED stays dim or barely lights	Resistor value too high, or LED wired reversed	Use a 220Ω–1kΩ resistor; check that the longer leg (anode) faces the resistor/power side.
Nothing happens at all / board not detected	Faulty USB cable or driver not installed	Try a different USB cable (some are charge-only) and confirm the CH340/FTDI driver is installed for your OS.

STILL STUCK? Re-read the wiring table for that specific project before assuming the component is faulty — the great majority of ‘broken’ modules turn out to be a swapped pin or a missing ground connection.