Land Buster - BT-Controlled RC Car -

Cucu Viorel-Cosmin 334CA

GitHub: PROIECT_PM_MASINUTA

MCU: ATmega328P Xplained Mini · PlatformIO

Introduction

Land Buster is a small RC car (1/12 scale Land Buster chassis) that I upgraded into a smart rover. Instead of using a standard radio remote, I control it from my phone via Bluetooth.

The main idea was simple: a normal RC car does whatever you tell it, even if it is about to crash. I wanted to fix that. So I added sensors and automatic logic on top of the manual control.

What it does:

- You drive it with a phone app over Bluetooth. You can pick between 3 speed modes and honk the horn.
- The front ultrasonic sensor watches for obstacles. The buzzer starts beeping faster as you get closer. If you get too close (under 15 cm), the car stops by itself - you cannot override it.
- The headlights turn on automatically when it gets dark, based on a light sensor.
- The LCD screen shows the current speed mode and the distance to the nearest obstacle in real time.

Why it is useful: It shows how a cheap microcontroller can handle multiple real-time tasks at once - wireless communication, sensing, motor control, and display - using only hardware registers and no external libraries.

Course requirements met:

- **5 laboratory concepts:** USART (Lab 1), Timers & Interrupts (Lab 2), PWM (Lab 3), ADC (Lab 4), I2C (Lab 6)
- **5 external peripherals:** HC-05 Bluetooth, HC-SR04 Ultrasonic, 1602 I2C LCD, Photoresistor, Active Buzzer

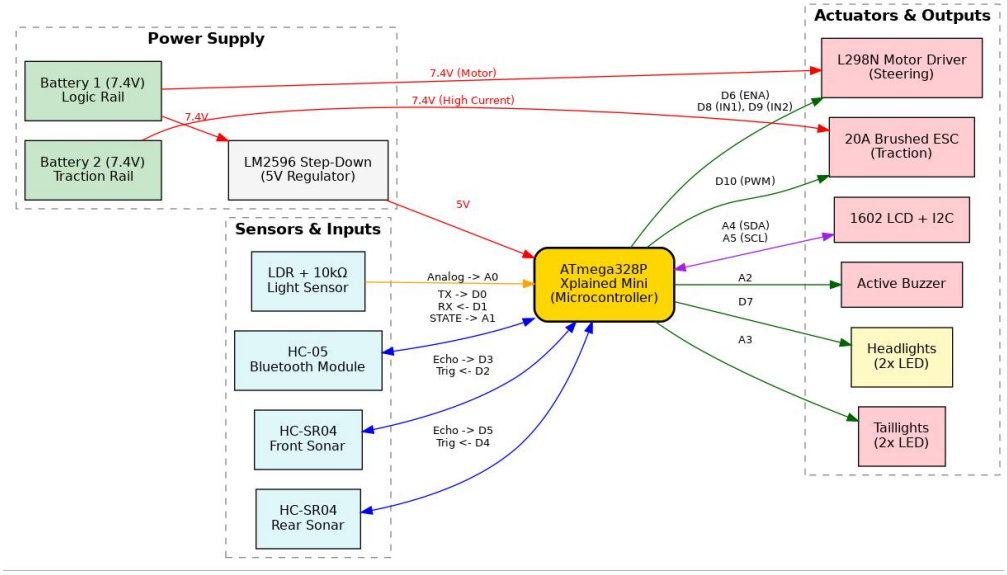
General Description

The ATmega328P is the brain of the system. Everything connects to it.

- **Inputs:** Bluetooth commands from the phone, distance data from the ultrasonic sensor, light level from the photoresistor.
- **Processing:** The MCU reads all inputs, decides what to do (drive, stop, beep, light up), and sends

signals to the outputs.
- **Outputs:** The ESC drives the main motor (speed), the L298N drives the steering motor (direction), the LCD shows status, the buzzer gives audio alerts, and the LEDs provide lighting.

Block Diagram



Hardware Design

Components and Their Role

Component	Model	Role in Project
Microcontroller	ATmega328P Xplained Mini	Central brain. Reads all sensors, runs non-blocking logic, outputs PWM and I2C signals.
Bluetooth Module	HC-05	Bidirectional telemetry with Android app - receives drive commands, sends sensor data.
Front Ultrasonic Sensor	HC-SR04	Measures distance to obstacles ahead. Triggers buzzer alerts and emergency stop at <15 cm.
Rear Ultrasonic Sensor	HC-SR04	Measures distance behind the rover. Powers the reverse parking assist display on the LCD.
Light Sensor	GL5528 Photoresistor + 10kΩ	Voltage divider read by ADC. Activates headlights automatically when ambient light drops.
Speed Controller	20A Brushed ESC	Controls the rear 390 DC traction motor via 50 Hz PWM. Receives signal from MCU Timer 1.
Traction Motor	390 DC Motor (7.4V)	Drives the rover forward and backward from Battery 2's dedicated high-current rail.
Steering Motor	3–6V DC Motor	Turns the front axle left or right under L298N control.

Motor Driver	L298N H-Bridge	Two direction pins (IN1/IN2) to drive the steering DC motor.
LCD Display	1602 LCD + PCF8574 I2C	“Smart dashboard”: shows expressive animated eyes in normal mode, parking assist in reverse.
Active Buzzer	TMB12A05 5V	Proximity alert (beep rate increases near obstacles) and horn command from phone.
Headlights & Taillights	5mm LEDs x4 + 220Ω x4	Front lighting (D7) activates automatically via LDR. Rear taillights (A3) activate on brake/reverse. Wired in parallel.
Voltage Regulator	LM2596 Step-Down	Converts Battery 1's 7.4V down to a stable 5V for all logic (MCU, sensors, LCD).
Battery 1 - Logic Rail	2x Murata US18650VTC5C in series	7.4V, 2600 mAh, 30A. Powers L298N and LM2596 (which feeds 5V logic).
Battery 2 - Traction Rail	2x Samsung 25R 18650 in series	7.4V, 2500 mAh, 8C. Dedicated high-current supply exclusively for the ESC.
Chassis	Land Buster 1/12 scale	Physical base of the rover.

Pin Mapping and Justification

Arduino Pin	AVR Register	Timer / Peripheral	Connected To	Why This Pin
D0 (RX)	PD0	USART0 RX	HC-05 TX	Hardware USART - zero-latency interrupt-driven reception (RXCIE0). No bit-banging needed.
D1 (TX)	PD1	USART0 TX	HC-05 RX via 1kΩ/2kΩ divider	Same USART peripheral. 5V output is divided to 3.3V to protect HC-05 RX logic level.
D2	PD2	GPIO Output	HC-SR04 Front - TRIG	Any GPIO works for the 10 µs trigger pulse. D2 keeps sensor pins grouped together.
D3	PD3	GPIO Input	HC-SR04 Front - ECHO	Paired with D2 for clean wiring. Reads echo pulse duration via pulseIn().
D4	PD4	GPIO Output	HC-SR04 Rear - TRIG	Same reason as D2; rear sensor pins grouped on D4/D5.
D5	PD5	GPIO Input	HC-SR04 Rear - ECHO	Paired with D4. Mirrors front sensor logic for parking assist.
D7	PD7	GPIO Output	Headlight LEDs + 220Ω	Simple digital ON/OFF for the LED array. D7 is free from any timer to avoid conflicts.
D8	PB0	GPIO Output	L298N IN1	Direction bit for steering H-bridge. No special hardware needed - plain digital output.
D9	PB1	GPIO Output	L298N IN2	Direction bit for steering H-bridge (opposite of IN1 to select left/right).
D10	PB2	Timer 1, OC1B (Phase-correct PWM)	ESC Signal	Timer 1 is 16-bit - essential for generating the precise 50 Hz (20 ms period) servo-style PWM the ESC requires. ICR1=39999, OCR1B range 2600–3400 for speed control.
A0	PC0	ADC0	LDR Voltage Divider	First ADC channel; no mux change needed for single-channel reads. Direct analog measurement of light level.
A1	PC1	GPIO Input	HC-05 STATE	Reads hardware connection status (HIGH = connected, LOW = disconnected/searching).

A2	PC2	GPIO Output	Active Buzzer	Moved here from SPI pins to keep PB3/PB4/PB5 free for the Xplained Mini's programming interface (EDBG).
A3	PC3	GPIO Output	Rear LED Taillights	Dedicated pin for red rear taillights. Activates independently during AEB, reverse, or stop.
A4	PC4	TWI SDA (Hardware I2C)	LCD PCF8574 SDA	Hardware I2C peripheral - only PC4/PC5 support hardware TWI on ATmega328P.
A5	PC5	TWI SCL (Hardware I2C)	LCD PCF8574 SCL	Paired with A4. Hardware TWI runs at 100 kHz without CPU intervention.

Key Design Choices & Electrical Schematic

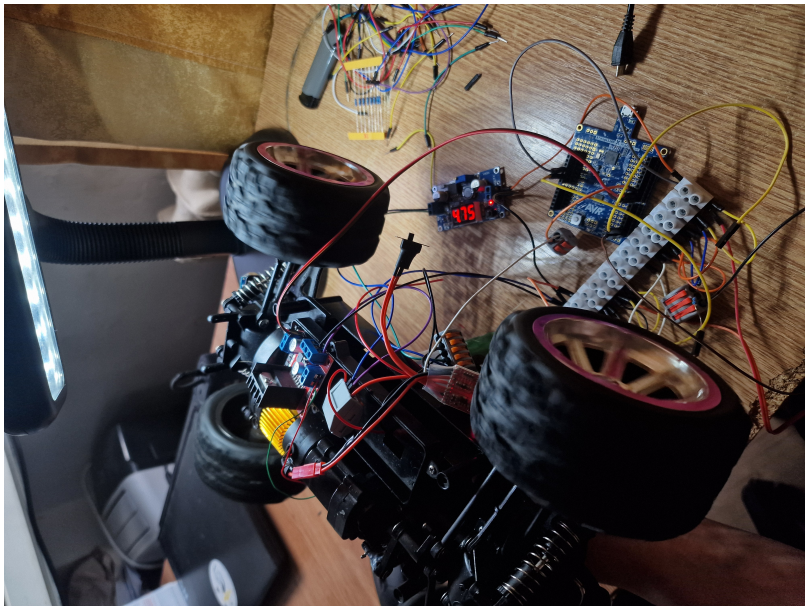
kicad_cucu2.pdf

The electrical schematic above illustrates the complete wiring of the Land Buster rover. The hardware architecture is built upon the following core engineering rules:

- **Dual Isolated Power System:** The system utilizes two separate battery packs. Battery 1 (Logic Rail) powers the L298N and the LM2596 regulator, which supplies a clean 5V to the MCU, sensors, and LCD. Battery 2 (Traction Rail) is dedicated exclusively to the high-current ESC. This isolation prevents massive voltage drops caused by the main motor from resetting the microcontroller.
- **Common Ground:** Despite having two isolated power sources, all ground (GND) connections across the entire system are tied together. This sets a unified 0V reference point, ensuring that PWM and digital logic signals are accurately interpreted by every component.
- **LED Parallel Wiring:** To ensure maximum brightness and protect the MCU pins, all 4 LEDs (2 front, 2 rear) are wired in parallel. Each LED connects to its respective MCU pin (D7 or A3) through its own dedicated 220Ω current-limiting resistor before returning to the common ground.
- **HC-05 Level Shifting:** The Bluetooth module operates at 3.3V logic. To prevent hardware damage, a simple voltage divider (1kΩ in series from MCU D1, then 2kΩ to GND) safely steps down the 5V USART TX signal to ≈3.3V before it reaches the module.
- **Timer Selection Strategy:** Timer 0 (8-bit) is used for the low-resolution steering PWM (Fast PWM, ~980 Hz). Timer 1 (16-bit) is strictly reserved for the ESC because it requires a precise 50 Hz period, which is impossible to achieve accurately with only 8 bits.

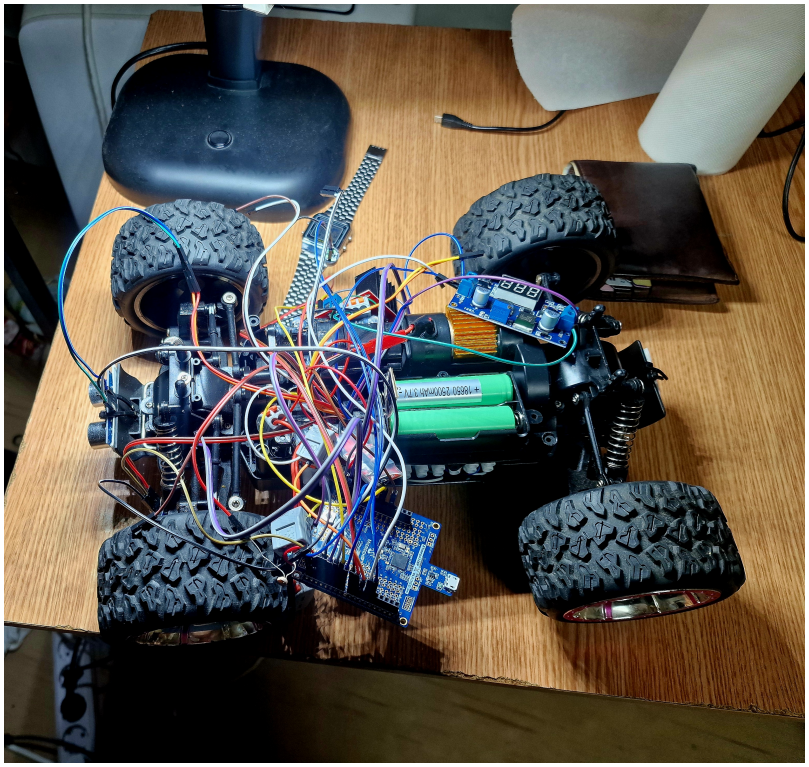
Proof of Functionality & Assembly

Photo 1: Power Drop Test



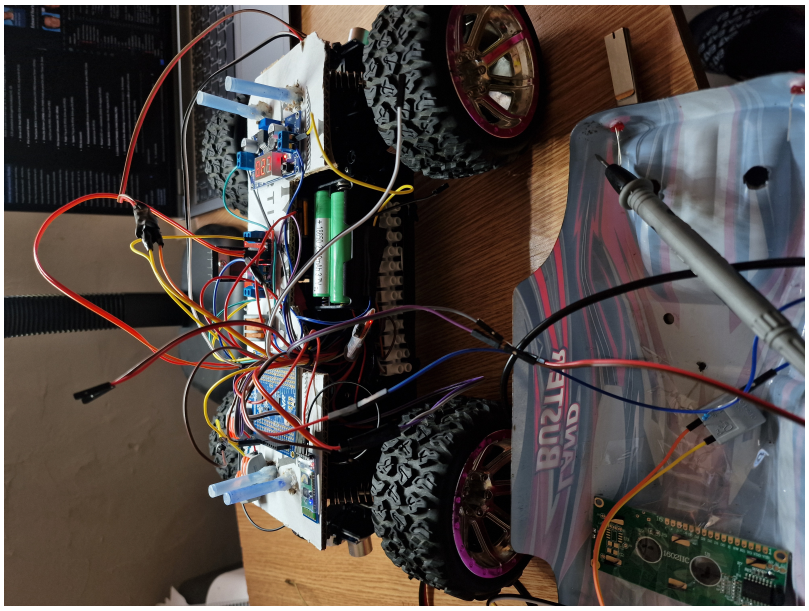
This early prototype test was conducted with only the steering system and ESC connected. While running the main traction motor at a low RPM, the output of the LM2596 regulator dropped dangerously to 4.7V. This critical observation proved that a single battery configuration was insufficient to handle the high current spikes, directly justifying the upgrade to the current dual-battery (4-cell) isolated power system.

Photo 2: Full System Test



All electronic components-including the ultrasonic sensors, I2C LCD, HC-05 Bluetooth module, and motor drivers-are integrated and tested together. This phase successfully verified the software integration, ensuring that the non-blocking logic, telemetry, and autonomous emergency braking (AEB) functioned simultaneously without interference, proving the system's core functionality.

Photo 3: Clean Build



The final hardware assembly mounted on the Land Buster chassis. This image highlights the strict cable management and the custom mechanical modifications made to the chassis to accommodate all the new components. To ensure maximum stability, the electronics and wiring were permanently secured using high-strength adhesives (epoxy resin and cyanoacrylate).

Photo 4: Final Assembly

The completed autonomous rover with the original outer car shell successfully mounted.

Software Design

Development Environment & Libraries Motivation

- **IDE:** PlatformIO (VS Code)
- **Language:** C++ (Arduino framework)
- **AI Assistance:** I used Claude Opus AI to assist with writing the code, structuring the non-blocking logic, and debugging hardware timer issues.
- **Libraries Motivation:**
  - <Arduino.h>: The core framework library. I used it for essential built-in functions like `millis()` (crucial for my non-blocking state machines), `pulseIn()` (to safely read ultrasonic sensors with a strict timeout), and basic GPIO control (`pinMode`, `digitalWrite`, `analogRead`).
  - <Wire.h>: The standard I2C library. Used to abstract the low-level TWI (Two-Wire Interface) hardware registers of the ATmega328P, providing robust communication with I2C devices.
  - <LiquidCrystal_I2C.h>: An external library that relies on <Wire.h>. It handles the complex data formatting required by the PCF8574 I2C expander attached to the 1602 LCD. I chose to use this library because writing a custom I2C LCD driver from scratch would be redundant and reinventing the wheel, allowing me to easily display custom animated characters. Everything else (like PWM and Timers) was implemented using direct register manipulation.

Justification of Laboratory Functionalities

The project actively uses 5 core concepts from the PM laboratory:

- **USART. Debugging (Laboratorul 1):** Essential for wireless communication. The HC-05 module receives driving commands from the Android phone via the hardware USART peripheral at 9600 baud.
- **Întreruperi, Timere (Laboratorul 2):** Critical for the non-blocking architecture. The project heavily relies on `millis()` (which operates via Timer 0 overflow interrupts) to manage sensor timing, signal debouncing, and connection failsafe timeouts without halting the CPU.
- **Timere, Pulse Width Modulation - PWM (Laboratorul 3):** Used to control the speed of the main traction motor via the ESC. Timer 1 (16-bit) is manually configured via hardware registers (`TCCR1A`, `TCCR1B`, `ICR1`) in Fast PWM mode to generate a highly precise 50Hz signal. The `OCR1B` register is varied between 3500 and 4000 to achieve 4 different speed gears.
- **Analog Digital Convertor - ADC (Laboratorul 4):** Used to read the analog voltage from the photoresistor (LDR). The ADC converts the light intensity into a digital value, which is compared against a threshold to automatically toggle the headlights.
- **I2C (Laboratorul 6):** Used to communicate with the PCF8574 expander on the back of the 1602 LCD, saving valuable GPIO pins.

Project Skeleton, Interaction, and Validation

The software is structured around a single, highly optimized, non-blocking `loop()`:

- **Interaction:** The loop continuously checks for incoming Bluetooth commands. It then alternates reading the front and rear ultrasonic sensors. Based on the distances, it evaluates the Autonomous Emergency Braking (AEB) logic. Finally, it sets the motor outputs and updates the LCD animations without using any delays.
- **Validation:** I validated the system by building a custom Android application that receives live telemetry from the car (sensor distances, light levels, active gear). I tested edge cases by driving at full speed towards walls to ensure the AEB triggers correctly and stops the car before impact.

Sensor Calibration

- **Photoresistor (LDR):** It is connected in a voltage divider with a 10kΩ resistor. I calibrated the software by reading the ADC values in different lighting conditions. I set the threshold to `< 400` so the headlights turn on automatically only when the room gets dark.
- **Ultrasonic Sensors (HC-SR04):** Distance is calculated using `duration / 58`. To calibrate them for real-time use, I reduced the timeout from the default 1,000,000µs to just `15000µs`. This limits the max read distance to ~2.5 meters, which is perfectly calibrated for indoor use and prevents the code from freezing when no echo is received.

Optimizations and Bug Fixes

How, why, and where I optimized the code:

- **Alternating Sensor Reads:** Initially, reading both ultrasonic sensors blocked the CPU for up to 40ms per loop. I optimized this by reading only *one* sensor per frame (front, then back). This cut the blocking time in half, making the steering perfectly responsive.
- **Removed Delays:** I eliminated all `delay()` calls from the main loop, replacing them with `millis()` state machines to keep the CPU free.
- **The Stuttering Bug (Critical Fix):** During development, the car had a major bug where the motor would stutter constantly. I originally used Timer 0 for the steering PWM and changed its prescaler from `/64` to `/1` to get more torque. **Why it broke:** Timer 0 is used internally by the framework for `millis()`. Changing the prescaler made `millis()` run 64x faster than normal. This caused my 800ms safety failsafe to trigger every 12.5 real milliseconds, violently turning the motor on and off. **How I fixed it:** I removed the Timer 0 manipulation entirely, used simple `digitalWrite()` for steering (giving full power), and the stuttering disappeared instantly.

Novelty

The main element of novelty is the **Autonomous Emergency Braking (AEB)** layered on top of manual RC control. Unlike a normal RC car that crashes if you make a mistake, this car intelligently intercepts your Bluetooth commands. If you command it to drive forward into a wall, the software debounces the sensor readings and overrides your command, applying a neutral brake to stop the car automatically. It implements a “directional block”: it stops you from hitting the wall in front, but still allows you to reverse safely.

Custom Android Application

To control the rover, I developed a custom Android application from scratch. Instead of relying on generic Bluetooth terminal apps, this dedicated app provides a custom user interface tailored for

driving. It features:

- A responsive joystick and button layout for steering and acceleration.
- 4 selectable speed gears (including a Turbo mode).
- Real-time telemetry: the app receives data back from the car (like sensor distances) and provides haptic feedback (vibration) when the Autonomous Emergency Braking (AEB) system is triggered.

Conclusions

This project shows that one small 8-bit microcontroller can handle wireless control, sensing, motor driving, and display - all at once -using only hardware registers.

Key lessons:

- **Interrupts** keep Bluetooth reception reliable without blocking the main loop.
- **Hardware timers** give accurate distance measurement.
- **Separate power rails** protect the MCU from motor noise and voltage drops.

Download

All project files (C sources, Makefile / PlatformIO config, and schematics) are available in the GitHub repository: PROIECT_PM_MASINUTA GitHub

Bibliography and Resources

Hardware Datasheets

- ATmega328P Datasheet (Microchip Technology)
- HC-SR04 Ultrasonic Sensor
- LM2596 Step-Down Regulator (Texas Instruments)
- PCF8574 I2C Expander (Texas Instruments)
- HC-05 Bluetooth Module (Vendor documentation)
- GL5528 Photoresistor (Manufacturer datasheet)
- L298N H-Bridge (STMicroelectronics datasheet)

Tools

- **PlatformIO + VS Code**: Writing and uploading firmware
- **Serial Bluetooth Terminal (Android)**: Sending commands from phone

Export to PDF