



Cairo University  
Faculty of Engineering



Credit Hours system

# GP

## Simulation Progress Report



**Date: 12/3/2025**

# Introduction:

This progress report outlines the development and implementation of a simulation for our graduation project, which focuses on an Obstacle Detection System and an Anti-lock Braking System (ABS) integrated through the Controller Area Network (CAN) communication protocol. While the primary objective of our graduation project is the digital design and verification of these systems, this simulation serves as a crucial intermediary step, allowing for a tangible, hardware-based demonstration of the project's purpose and functionality.

The necessity of this simulation arises from the complexity of the core project, which primarily involves FPGA-based digital design. In digital design, the output is typically represented as a waveform, which is not easily readable or understandable for those unfamiliar with digital systems. However, by implementing the project on hardware, the results become visually observable, allowing anyone to perceive the system's functionality without requiring technical knowledge of digital waveforms. By leveraging an embedded C-based approach on TivaC, we ensure that the fundamental principles of obstacle detection and ABS control can be effectively visualized and tested. This simulation aids in validating the system's behavior before final integration, providing valuable insights into potential real-world performance.

# Work Completed:

Significant progress has been made in the development of the Embedded C simulation for the obstacle detection and ABS system using the TivaC TM4C123GH6PM microcontroller. The full C code for the simulation has been completed, covering all essential drivers and system functionalities.

The system is designed to simulate a moving vehicle that stops when an obstacle is detected or when the driver presses the brakes. This is implemented using:

- Four DC motors, representing the vehicle's wheels.
- An ultrasonic sensor, continuously measuring distance to detect obstacles.
- A push button (SW1 on PF4), simulating the driver pressing the brakes.
- A red LED (PF1), providing a visual indication when braking is active.
- UART communication, allowing the user to start the system by sending an "ON" command.

The implementation follows a structured approach, ensuring each hardware component is properly initialized and integrated:

1. **UART Driver:** Enables serial communication for user input ("ON" command) and system messages.
2. **DC Motor Driver:** Controls the movement of all four wheels, allowing the vehicle to move and stop.
3. **Ultrasonic Sensor Driver:** Continuously takes distance readings to detect obstacles.
4. **Buttons and LED Driver:** Handles braking via the push button and provides visual feedback through the red LED.
5. **Main Program:** Integrates all drivers, processes user commands, and continuously monitors distance to make real-time decisions.

## Buttons and LEDs:

The Buttons and LEDs driver is responsible for handling user input through a push button (SW1 on PF4) and providing visual feedback using an LED (PF1 - Red LED) on the TivaC TM4C123GH6PM microcontroller. This driver plays a crucial role in simulating the braking system of the vehicle. The push button acts as the brake pedal, and when pressed, the vehicle stops by halting all four DC motors. Simultaneously, the red LED turns on to indicate that braking has been activated. Once the button is released, the vehicle resumes motion. The driver is implemented using GPIO (General-Purpose Input/Output) configurations and interrupt handling, ensuring efficient and immediate response to user input.

### Push Button Initialization – SW1\_Init()

This function configures PF4 as an input with an edge-triggered interrupt to detect when the button is pressed.

- Disables analog functionality on PF4 to ensure it operates as a general-purpose input.
- Clears PMCx bits to remove any special functions from PF4, making it a standard GPIO pin.
- Configures PF4 as an input by clearing its bit in the DIR (Direction) register.
- Enables an internal pull-up resistor to keep PF4 at logic HIGH when unpressed, preventing it from floating.
- Configures the interrupt to detect a falling edge, meaning it triggers when the button is pressed (HIGH to LOW transition).
- Clears any previous interrupt flags to ensure a clean start.
- Enables the interrupt for PF4 in the NVIC (Nested Vectored Interrupt Controller) to ensure immediate system response.

## **Red LED Initialization – Led\_Red\_Init()**

This function configures PF1 as an output to provide a visual indication when braking is activated.

- Disables analog functionality on PF1 to ensure it works as a standard GPIO pin.
- Clears PMCx bits to remove any special functions from PF1.
- Configures PF1 as an output by setting its bit in the DIR (Direction) register.
- Disables alternate functions to ensure PF1 is treated purely as a GPIO pin.
- Enables digital functionality by setting PF1 in the DEN (Digital Enable) register.
- Initially turns off the LED by clearing PF1 in the DATA register

## **Interrupt Service Routine – GPIOPortF\_Handler()**

This function is the Interrupt Service Routine (ISR) that executes when SW1 (PF4) is pressed.

- Checks if PF4 triggered the interrupt by examining the RIS (Raw Interrupt Status) register.
- Stops all four DC motors to simulate the braking system in action.
- Sends a message via UART ("Brakes are pressed") to provide feedback.
- Waits for the button to be released before restarting the vehicle.
- Restarts all four DC motors, allowing the vehicle to resume motion.
- Clears the interrupt flag by writing 1 to the ICR (Interrupt Clear) register, ensuring the system is ready for the next button press.

Finally, the interrupt flag is cleared by writing 1 to the ICR (Interrupt Clear) register, ensuring that the system is ready to detect the next button press without false triggering.

## DC Motors:

The DC Motors driver is responsible for controlling the movement of the four wheels of the simulated vehicle. Each motor represents a wheel, and the system uses H-Bridge motor drivers to control the direction and speed of each motor. The TivaC TM4C123GH6PM microcontroller sends signals to the motor driver ICs, determining whether each motor moves forward, reverse, or stops.

In this system:

- The motors start when the user sends the "ON" command via UART.
- The ultrasonic sensor continuously measures the distance to detect obstacles.
- If an obstacle is detected, the motors stop, simulating an emergency braking scenario.
- The push button (SW1) on the TivaC board also acts as a brake; when pressed, the motors immediately stop.

The motor control is implemented using GPIO pins configured as outputs, with dedicated enable pins (ENA/ENB) to control power flow. The following sections explain the initialization and control functions for each motor in detail.

### **Motor A Initialization – DC\_MOTORA\_INIT()**

This function initializes Motor A, which is controlled through Port B on the microcontroller.

- Enables the clock for Port B to allow GPIO functionality.
- Waits for the clock to stabilize before configuring the pins.
- Disables analog functionality to use the pins as digital outputs.
- Clears PMCx bits to set the pins as standard GPIO outputs.
- Configures PB2, PB3, and PB6 as output pins to control the motor.

- Disables alternate functions so that the pins work purely as GPIO.
- Enables digital I/O on PB2, PB3, and PB6 for proper motor control.
- Turns off the motor initially by clearing PB2 and PB3.
- Enables the H-Bridge driver by setting PB6 high, allowing the motor to be controlled.

### **Motor A Start – DC\_MOTORA\_START()**

This function activates Motor A in the forward direction.

- Sets PB2 HIGH and PB3 LOW, making the motor rotate forward.
- The H-Bridge ensures that current flows in one direction, causing movement..

### **Motor A Stop – DC\_MOTORA\_STOP()**

This function stops Motor A by cutting off power.

- Sets PB2 and PB3 HIGH, forcing both motor terminals to the same potential and stopping rotation.
- Waits for 100 ms to prevent sudden back EMF from damaging components.
- Clears PB2 and PB3, ensuring that the motor remains OFF.

### **Motor B Initialization – DC\_MOTORB\_INIT()**

This function initializes Motor B, similar to Motor A, but on different GPIO pins (PB4, PB5, PB7).

- PB4 and PB5 control direction (IN3, IN4).
- PB7 enables/disables the motor (ENB).
- Same configuration steps as Motor A, ensuring it's properly set up.

## **Motor B Start – DC\_MOTORB\_START()**

- Sets PB4 HIGH and PB5 LOW, making the motor rotate forward.

## **Motor B Stop – DC\_MOTORB\_STOP()**

- Stops the motor using the same back EMF protection method as Motor A.

## **Motor C Initialization – DC\_MOTORC\_INIT()**

Motor C is controlled using Port E (PE0, PE1, PE2).

- Enables the clock for Port E and waits for stabilization.
- Disables analog functionality and clears PMCx bits.
- Configures PE0, PE1, and PE2 as outputs.
- Enables digital I/O and turns off the motor initially.
- PE2 is used to enable the H-Bridge for Motor C.

## **Motor C Start – DC\_MOTORC\_START()**

- Sets PE0 HIGH and PE1 LOW, activating the motor forward.

## **Motor C Stop – DC\_MOTORC\_STOP()**

- Stops the motor with back EMF protection, similar to the other motors.

## **Motor D Initialization – DC\_MOTORD\_INIT()**

Motor D is controlled using Port D (PD2, PD3, PD6).

- Same initialization steps as the other motors, but using PD2, PD3, and PD6.
- PD6 is used to enable the H-Bridge for Motor D.



## **Motor D Start – DC\_MOTORD\_START()**

- Sets PD2 HIGH and PD3 LOW, making the motor rotate forward.

## **Motor D Stop – DC\_MOTORD\_STOP()**

- Stops the motor with the same protection mechanism as the others.

## **UART:**

The UART (Universal Asynchronous Receiver-Transmitter) driver is responsible for enabling serial communication between the TivaC TM4C123GH6PM microcontroller and external devices, such as a PC terminal or another microcontroller. This driver plays a crucial role in the vehicle simulation, allowing users to send commands and receive status messages.

In this system:

- The user sends the "ON" command via UART to start the vehicle simulation.
- The system sends status messages, such as "Brakes are pressed", when the push button is triggered.
- The driver supports both sending and receiving single bytes, strings, and fixed-size data, ensuring smooth communication.

UART communication is implemented using PA0 (RX - Receive) and PA1 (TX - Transmit). The following sections explain the initialization and control functions in detail.

## **GPIO Configuration for UART – GPIO\_SetupUARTToPins()**

Before enabling UART, PA0 and PA1 must be configured as UART pins. This function ensures that the correct GPIO settings are applied.

- Enables the clock for Port A, allowing PA0 and PA1 to function.
- Waits for the clock to stabilize before proceeding.

- Disables analog functionality to use PA0 and PA1 as digital I/O.
- Configures PA0 as an input (RX) and PA1 as an output (TX).
- Enables alternate functions for PA0 and PA1, allowing them to operate as UART pins.
- Sets the PMCx bits to assign UART0 functionality to these pins.
- Enables digital I/O on PA0 and PA1, completing the setup.

## **UART Initialization – UART0\_Init()**

This function configures UART0 to enable serial communication.

- Calls GPIO\_SetupUART0Pins() to set up PA0 and PA1 for UART operation.
- Enables the clock for UART0 and waits for it to stabilize.
- Disables UART0 temporarily to allow safe modifications.
- Sets the baud rate to 9600 bps for standard communication.
- Configures the UART format to 8-bit data, no parity, 1 stop bit, ensuring compatibility with most serial devices.
- Enables UART transmission and reception, making UART0 fully functional.

## **Send a Single Byte – UART0\_SendByte(unsigned char data)**

This function transmits a single byte over UART.

- Waits until the transmit FIFO (buffer) is empty, ensuring the previous byte has been sent.
- Writes the new byte to the UART data register, sending it over the TX line.

## **Receive a Single Byte – UART0\_ReceiveByte()**

This function receives a single byte from UART.

- Waits until data is available in the receive FIFO, preventing invalid reads.
- Reads and returns the received byte.

## **Send a String – UART0\_SendString(unsigned char \*pData)**

This function transmits a sequence of characters (a string).

- **Loops through the string and sends each character one by one using UART0\_SendByte().**
- **Stops when the null terminator (\0) is reached**, ensuring only the intended data is sent.

## **Receive a String – UART0\_ReceiveString(unsigned char \*pData)**

This function receives a sequence of characters until a # symbol is detected.

- Calls UART0\_ReceiveByte() repeatedly, storing each received character in pData.
- Stops when # is received, indicating the end of the command.
- Replaces # with a null terminator (\0) to correctly format the string.

## **Send a Specific Number of Bytes – UART0\_SendData(unsigned char \*pData, unsigned long uSize)**

This function sends a fixed amount of data over UART.

- Loops through uSize bytes and transmits each using UART0\_SendByte().
- Ensures only the required amount of data is sent, making it useful for structured communication protocols.

## **Receive a Specific Number of Bytes – UART0\_ReceiveData(unsigned char \*pData, unsigned long uSize)**

This function receives a fixed amount of data over UART.

- Calls UART0\_ReceiveByte() exactly uSize times, storing the received bytes in pData.
- Useful for receiving structured messages where the length is known in advance.

## Ultrasonic:

The Ultrasonic Sensor driver is responsible for continuously measuring the distance between the vehicle and any obstacle in front of it. This sensor is a critical component of the system as it determines when the vehicle should stop to prevent a collision. The driver controls the ultrasonic sensor's trigger and echo mechanism, allowing it to calculate distances based on sound wave reflections.

In this system:

- The ultrasonic sensor is connected to Port E, using:
  - PE4 (Trigger pin) → Sends out ultrasonic pulses.
  - PE5 (Echo pin) → Receives the reflected pulse and measures the time taken.
- The system continuously measures the distance using `Ultrasonic_GetDistance()`.
- If an obstacle is detected within the defined threshold (< 5 cm), the system:
  - Stops all DC motors to prevent a collision.
  - Turns on the red LED (PF1) as a warning indicator.
  - Sends a message via UART ("Near Collision") to inform the user.

The ultrasonic sensor works using the time-of-flight principle:

- The sensor sends an ultrasonic pulse (via the trigger pin).
- The pulse travels forward until it hits an object and bounces back.
- The echo pin captures the returned signal and measures the time taken.
- Using the formula:

$$\text{Distance} = \frac{\text{Time} \times 340}{2 \times 1000}$$

## **Ultrasonic Sensor Initialization – Ultrasonic\_Init()**

This function initializes Port E for ultrasonic sensor operation.

- Enables the clock for Port E, ensuring it is ready for use.
- Waits for the clock to stabilize before proceeding.
- Disables analog functionality on PE4 and PE5, making them digital pins.
- Clears PMCx bits to assign standard GPIO functionality to PE4 and PE5.
- Disables alternate functions to ensure normal GPIO behavior.
- Enables digital I/O for both pins.
- Configures PE4 as an output (Trigger pin) and PE5 as an input (Echo pin).

## **Triggering the Ultrasonic Sensor – Ultrasonic\_Trigger()**

This function sends an ultrasonic pulse to start the measurement.

- Clears PE4 (Trigger pin) to LOW to ensure a clean pulse.
- Waits for 10 microseconds to stabilize.
- Sets PE4 HIGH for 10 microseconds, sending a short burst of ultrasonic waves.
- Clears PE4 LOW again, completing the pulse cycle.

## **Measuring the Distance – Ultrasonic\_GetDistance()**

This function calculates the distance based on the ultrasonic signal's round-trip time.

- Triggers the sensor using Ultrasonic\_Trigger().
- Waits until the Echo pin (PE5) goes HIGH, indicating the signal has been sent.
- Starts measuring the time while PE5 remains HIGH.
- Stops measuring when PE5 goes LOW, meaning the signal has returned.

- Uses the time value to calculate distance in centimeters using: Distance=

$$\text{Distance} = \frac{\text{Time} \times 340}{2 \times 1000}$$

- Returns the measured distance for further processing.

## **Monitoring Distance and Preventing Collisions – Monitor\_DIST()**

This function continuously monitors the distance and stops the vehicle if an obstacle is too close.

- Calls Ultrasonic\_GetDistance() to measure the current distance.
- Compares the distance to DISTANCE\_THRESHOLD (5 cm).
- If an obstacle is detected:
  - Sends a warning message ("Near Collision") over UART.
  - Turns on the red LED (PF1) to alert the user.
  - Stops all four DC motors, preventing a crash.

## **The main:**

The main function serves as the central controller, coordinating the initialization and execution of the system. It begins by initializing all essential hardware components, including the DC motors, ultrasonic sensor, push button, LED, and UART communication module. The program then waits for user input via UART, prompting the user to enter "ON" to activate the system. Once the correct command is received, the system enters an infinite loop, continuously monitoring the distance to obstacles using the ultrasonic sensor. If an obstacle is detected within the predefined threshold distance (5 cm), the system immediately stops all DC motors, turns on the red warning LED, and sends an alert message via UART, simulating an automatic braking system. This simple yet effective structure ensures real-time obstacle detection and collision prevention while maintaining interactive control through user input

# Challenges & Solutions:

Throughout the development of this project, several technical and implementation challenges were encountered. Each challenge required careful analysis and problem-solving to ensure an efficient and reliable system. Below are some of the key challenges faced during the implementation and the solutions that were applied to resolve them.

- **Implementing an immediate braking system using push buttons**

The push button was intended to act as a braking mechanism for the system, immediately stopping all DC motors when pressed. However, initial attempts to implement this functionality using polling methods or simple conditional checks proved inefficient, as they introduced delays and did not guarantee an immediate response.

**Solution:** To ensure a real-time response, an interrupt-based approach was implemented. The push button was configured to trigger an interrupt request, allowing the system to detect and process the button press event immediately. When the button is pressed, an interrupt service routine is executed, stopping all DC motors instantly without delay. This significantly improved the system's responsiveness, ensuring immediate braking when needed.

- **Ensuring continuous readings from the ultrasonic sensor**

The ultrasonic sensor is responsible for detecting obstacles, but in the initial implementation, it was only triggered once, which meant the system could not continuously monitor the environment for obstacles.

**Solution:** To enable continuous distance monitoring, a while loop was added in the main function, ensuring that the Monitor\_DIST function runs indefinitely once the



system is started. This allows the ultrasonic sensor to continuously take distance measurements and trigger the braking mechanism whenever an obstacle is detected.

- **Efficiently stopping the system when needed**

Since the system continuously runs after receiving the "ON" command, a mechanism was needed to allow for a controlled stop when necessary.

**Solution:** Instead of adding unnecessary complexity with additional user inputs, the same push button used for braking was also utilized to stop the entire system when pressed. This was achieved by modifying the interrupt routine to not only stop the motors but also exit the monitoring loop, effectively stopping the system execution.

- **Ensuring reliable UART communication for system start-up**

The system relies on UART input ("ON") to start, but there was a risk that incomplete or incorrect inputs could cause the system to behave unpredictably.



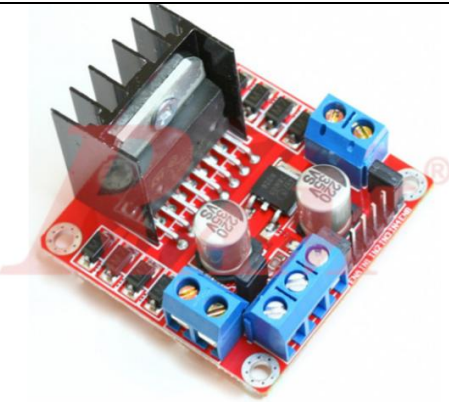
**Solution:** The UART input function was modified to ensure that only the exact "ON" command, followed by a null terminator, is accepted. This prevents incorrect or partial inputs from starting the system and improves input reliability.


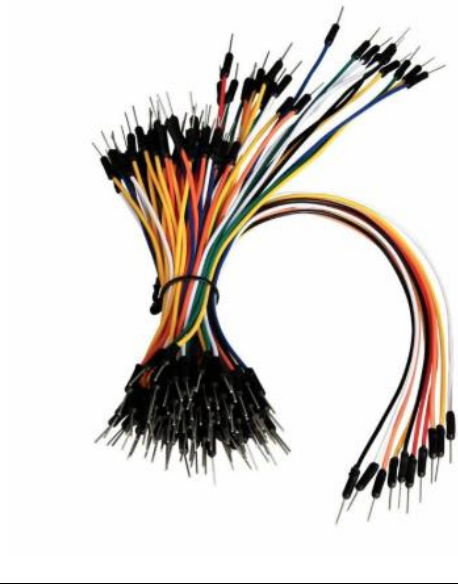
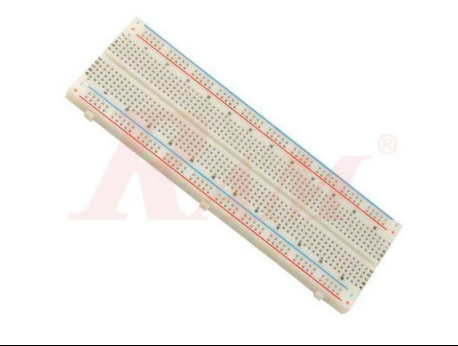

- **Selecting the appropriate ports for motor control**

The microcontroller has limited GPIO pins, and choosing the wrong ones could lead to conflicts with other peripherals.

**Solution:** A careful pin mapping strategy was applied, ensuring that each DC motor was assigned to a GPIO port that does not interfere with the UART or ultrasonic sensor operations. Additionally, ports requiring an unlock sequence were avoided to simplify the configuration process.

## Cost Analysis:

Product	Quantity	Price	Picture
TivaC	1	3,500	
DC Motors (Robot Car Chassis Kit)	1	360	
Hbridge	2	160	

Ultrasonic	2	80	
Jumper Wires	75	52	
Breadboard	2	55	
Power Adapter	1	75	
<b>Total</b>		<b>4,282 LE</b>	

## **Current Status:**

At this stage, all embedded C drivers for the system have been fully developed, including UART communication, DC motor control, ultrasonic sensing, and push button integration with interrupts. The code is structured, modular, and efficiently handles real-time responses for braking and obstacle detection. The simulation is fully functional, successfully demonstrating the vehicle's ability to start upon receiving the "ON" command, move forward, continuously monitor for obstacles, and stop immediately when an obstacle is detected or when the brake button is pressed. The system effectively integrates all components in software, ensuring seamless interaction between sensors, actuators, and control logic. However, while the software has been carefully designed and reviewed, it has not yet been tested on the actual hardware, meaning real-world verification is still required.

## **Next Steps:**

The next critical phase is hardware integration and testing. This will begin by individually verifying each driver on the TivaC microcontroller to ensure that all peripherals function as expected. The UART module will be tested first to confirm reliable data transmission and reception. The DC motor drivers will be tested separately to validate correct direction control and braking response. The ultrasonic sensor will then be integrated and tested to ensure accurate and continuous distance measurements. Once each component is verified, the complete system will be assembled and tested in a real-world environment to assess overall performance. Any necessary adjustments or optimizations will be made to improve response time, accuracy, and reliability.

## Conclusion:

The project has successfully transitioned from concept to implementation, with a fully developed and structured software system that simulates an obstacle detection and braking mechanism. The use of embedded C on the TivaC microcontroller provides a strong foundation for real-time embedded control. The modular nature of the code allows for easy debugging and future enhancements. With all software components completed, the focus now shifts to hardware validation, where practical testing will determine the effectiveness of the system in real-world conditions. This marks a crucial step toward finalizing the project and ensuring a fully functional embedded safety system.

# References:

- [https://www.google.com.eg/imgres?q=embedded%20systems&imgurl=https%3A%2F%2Fwww.velvetechnology.com%2Fwp-content%2Fuploads%2F2023%2F03%2Fwhat-are-embedded-systems.png&imgrefurl=https%3A%2F%2Fwww.velvetechnology.com%2Fblog%2Ftypes-of-embedded-systems%2F&docid=b8G5GpcNHIFcVM&tbnid=3\\_hAn1WnSRKC0M&vet=12ahUKEwjT16W\\_goOMAxU21AIHHZFINI0QM3oECHQQAA..i&w=900&h=350&hcb=2&ved=2ahUKEwjT16W\\_goOMAxU21AIHHZFINI0QM3oECHQQAA](https://www.google.com.eg/imgres?q=embedded%20systems&imgurl=https%3A%2F%2Fwww.velvetechnology.com%2Fwp-content%2Fuploads%2F2023%2F03%2Fwhat-are-embedded-systems.png&imgrefurl=https%3A%2F%2Fwww.velvetechnology.com%2Fblog%2Ftypes-of-embedded-systems%2F&docid=b8G5GpcNHIFcVM&tbnid=3_hAn1WnSRKC0M&vet=12ahUKEwjT16W_goOMAxU21AIHHZFINI0QM3oECHQQAA..i&w=900&h=350&hcb=2&ved=2ahUKEwjT16W_goOMAxU21AIHHZFINI0QM3oECHQQAA)
- <https://free-electronic.com/product/robot-car-chassis-kit-1-layer-4wd/>
- [https://www.google.com.eg/imgres?q=ultrasonic%20sensor&imgurl=https%3A%2F%2Fstore.fut-electronics.com%2Fcdn%2Fshop%2Fproducts%2Fhc-sr04-ultrasonic-module-ultrasonic-sensor\\_grande.jpg%3Fv%3D1564132976&imgrefurl=https%3A%2F%2Fstore.fut-electronics.com%2Fproducts%2Fultrasonic-sensor-module&docid=aJRyTiPLSn9ybM&tbnid=ftiWMQIqiCBj4M&vet=12ahUKEwj01rD75IOMAxV5YEEAHWOMEa8QM3oECBsQAA..i&w=600&h=600&hcb=2&ved=2ahUKEwj01rD75IOMAxV5YEEAHWOMEa8QM3oECBsQAA](https://www.google.com.eg/imgres?q=ultrasonic%20sensor&imgurl=https%3A%2F%2Fstore.fut-electronics.com%2Fcdn%2Fshop%2Fproducts%2Fhc-sr04-ultrasonic-module-ultrasonic-sensor_grande.jpg%3Fv%3D1564132976&imgrefurl=https%3A%2F%2Fstore.fut-electronics.com%2Fproducts%2Fultrasonic-sensor-module&docid=aJRyTiPLSn9ybM&tbnid=ftiWMQIqiCBj4M&vet=12ahUKEwj01rD75IOMAxV5YEEAHWOMEa8QM3oECBsQAA..i&w=600&h=600&hcb=2&ved=2ahUKEwj01rD75IOMAxV5YEEAHWOMEa8QM3oECBsQAA)
- <https://www.ram-e-shop.com/shop/kit-1298-red-1298-module-red-board-dual-h-bridge-motor-driver-using-1298n-7084>
- <https://www.ram-e-shop.com/shop/bb01-bread-board-bb-01-breadboard-830-tie-point-6143>
- <https://www.ram-e-shop.com/shop/adapter-fixed-12v-1a-wall-power-adapter-12vdc-1a-high-quality-adapter-6413>
- <https://free-electronic.com/product/65pcs-flexible-breadboard-jumper-wires/>