Command should Start with “ and end with “#” and desire imput in between.

Sample @[…input…]#

The input can be:

* @B# for Turn ON the Reading Inputs Process.
* @F# for Turn OFF the Reading Inputs Process.
* This device has 58 Actuators. The input should be an array of 580 (58 set of 10)(with starting, ending char and null Transmission will be 582 chars) numeric digits in which the first 5 digits indicates **low duration** and second 5 digits indicates **high duration.**

All Off :

@0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000#

@0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000009000040000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000#

For Second Microcontroller : (It is an internal communication of microcontrollers)

A000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000E

Considering that we can set al pins of ESP32-S3 as PWM BUT not assigning more than 8 PWM at the same time, force me to design a custom PWM (Pulse Width Modulation) mechanism to use all the GPIO pin of ESP32 as PWM for controlling my actuators.

### Maximum and Minimum Frequency

The maximum and minimum frequencies you can achieve with your PWM setup depend on the minimum and maximum duration values you can set for the high and low times. Given your script's configuration:

- \*\*Minimum Duration:\*\* 100 microseconds

- \*\*Maximum Duration:\*\* 999,999 microseconds

#### Maximum Frequency

The highest frequency is achieved when both high and low durations are at their minimum. Using 100 µs for each:

- \*\*Total Period:\*\* \(100 \text{ µs (Low)} + 100 \text{ µs (High)} = 200 \text{ µs}\)

- \*\*Frequency:\*\* \(f = \frac{1}{200 \times 10^{-6}} = 5000 \text{ Hz}\)

#### Minimum Frequency

The lowest frequency occurs when both durations are at their maximum:

- \*\*Total Period:\*\* \(999,999 \text{ µs (Low)} + 999,999 \text{ µs (High)} = 1,999,998 \text{ µs}\)

- \*\*Frequency:\*\* \(f = \frac{1}{1.999998 \text{ s}} \approx 0.0005 \text{ Hz}\) or one cycle every 2000 seconds.

### Flexibility in Setting Different High and Low Durations

The ability to set different durations for high and low states allows you to manipulate the duty cycle of the PWM signal independently of the frequency. The \*\*duty cycle\*\* describes the proportion of the time a digital signal is 'ON' versus 'OFF' and is a key factor in controlling the energy level delivered by the signal.

#### Practical Advantages:

1. \*\*Control Over Signal Power\*\*: By adjusting the duty cycle (through different high and low durations), We can control the amount of power delivered to a load without changing the frequency. This is crucial for applications like motor speed control or Vibrators, where we want to maintain consistent performance characteristics while varying output levels.

2. \*\*Generating Different Signal Profiles\*\*: Different high and low time ratios can create distinct signal patterns, which can be useful in applications requiring specific signal shapes — for instance, generating unique waveforms for sound generation or complex motor control scenarios.

3. \*\*Creating Different Textures\*\*: In applications like haptic feedback or sound production, the texture or feel of the output can be adjusted by varying the duty cycle. This can create sensations ranging from smooth to rough, which can enhance user interaction in devices like smartphones, wearable technology, or virtual reality systems.

#### Claiming Different Textures:

Yes, you can claim that the script allows the generation of different "textures" in terms of output feel or interaction quality, especially relevant in systems involving tactile feedback. The variable duty cycle made possible by independent high and low duration settings directly affects the nature of the PWM signal, thereby influencing the physical sensation or response experienced by the user.

By providing these detailed controls, your script enhances the versatility and applicability of the PWM signals generated, catering to a wider range of practical applications and allowing for more nuanced control systems.

Your script processes PWM (Pulse Width Modulation) settings for multiple pins using 12-character strings for each pin. Each 12-character block consists of 6 characters for the low duration and 6 characters for the high duration, both in microseconds. Here's a brief explanation and some examples in a table format showing how you can achieve specific frequencies with these settings:

Description for achieving difference frequency:

Low Duration: The first 6 characters define how long (in microseconds) the pin stays in the LOW state.

High Duration: The next 6 characters specify the duration (in microseconds) for the HIGH state.

Total Cycle Time: Sum of Low and High durations determine the frequency.

Given that frequency 𝑓 is defined as 𝑓=1/total period , where the total period is the sum of the low and high durations, you can adjust these values to achieve desired frequencies.

Examples:

| **Low Duration (µs)** | **High Duration (µs)** | **Total Period (µs)** | **Frequency (Hz)** |
| --- | --- | --- | --- |
| 100000 | 100000 | 200000 | 5 |
| 50000 | 50000 | 100000 | 10 |
| 20000 | 20000 | 40000 | 25 |
| 10000 | 10000 | 20000 | 50 |
| 5000 | 5000 | 10000 | 100 |

**Duty Cycle and Power**

The duty cycle is calculated as:

Duty Cycle=( High Duration/ Total Period)×100%

Where:

* High Duration is the time the signal is HIGH.
* Total Period is the sum of High and Low durations.
* A higher duty cycle generally means more power is delivered (because the signal is HIGH a greater proportion of the time).

Table Explanation

* For this example, let's keep the frequency at 100 Hz, which means the total period must be 10,000 microseconds (since Frequency=1/Period ​):

| **Low Duration (µs)** | **High Duration (µs)** | **Total Period (µs)** | **Frequency (Hz)** | **Duty Cycle (%)** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 9000 | 1000 | 10000 | 100 | 10% | Lower power, signal is HIGH 10% of time |
| 8000 | 2000 | 10000 | 100 | 20% | Increased power, HIGH 20% of time |
| 5000 | 5000 | 10000 | 100 | 50% | Moderate power, HIGH 50% of the time |
| 2000 | 8000 | 10000 | 100 | 80% | Higher power, HIGH 80% of the time |
| 1000 | 9000 | 10000 | 100 | 90% | Maximum power (almost), HIGH 90% of time |

How ESP32 Pins are connected:

| ESP32-S3 No.1 | Connected to | ESP32-S3 No.2 |
| --- | --- | --- |
| GPIO9-QUADHD | 🡨🡪 | GPIO9-QUADHD |
| GPIO10-CS0 | 🡨🡪 | GPIO10-CS0 |
| GPIO11-MOSI-MISO | 🡨🡪 | GPIO13-MISO-MOSI |
| GPIO12-SCLK | 🡨🡪 | GPIO12-SCLK |
| GPIO13-MISO-MOSI | 🡨🡪 | GPIO11- MOSI-MISO |
| GPIO14-QUADWP | 🡨🡪 | GPIO14-QUADWP |

Communication protocol:

**Overview**

The protocol is designed to send and receive data in 4-bit segments (nibbles), making it efficient for simple data transmission tasks where high bandwidth is not necessary but low overhead and simplicity are valued.

**Components**

Master Device: Initiates communication, sends data in 4-bit segments.

Slave Device: Receives data in 4-bit segments.

**GPIO Pin Configuration**

CLOCK\_PIN (GPIO 12): Used to synchronize data transmission.

DATA\_Transmission\_STATE\_PIN (GPIO 10): Indicates when data transmission is active.

DATA\_PINS (GPIO 9, 13, 14, 11): Transmit the data bits.

**Communication Process**

**Master Side**

1. **Initialization:** Set the DATA\_PINS as outputs and the CLOCK\_PIN as an output. The DATA\_Transmission\_STATE\_PIN is also set to output to signal the beginning and end of data transmission.
2. **Data Transmission:**
   * The Master places a nibble (4-bit data) on the DATA\_PINS.
   * It sets the CLOCK\_PIN high to indicate that the data is ready to be read.
   * After a brief delay (to ensure the Slave has time to read the data), it sets the CLOCK\_PIN low.
   * This process repeats for each nibble of data to be sent.
   * To start a transmission, the Master sets the DATA\_Transmission\_STATE\_PIN high, and once done, sets it low.

**Slave Side**

Initialization: Set the DATA\_PINS as inputs and both the CLOCK\_PIN and DATA\_Transmission\_STATE\_PIN as inputs to receive signals from the Master.

**Data Reception:**

The Slave continuously monitors the CLOCK\_PIN.

When the CLOCK\_PIN goes high, it reads the 4 bits from the DATA\_PINS.

It constructs the data byte by byte from the nibbles received.

Continues to read data until the DATA\_Transmission\_STATE\_PIN is set low by the Master, indicating the end of the transmission.

**Protocol Characteristics**

**Simple and Low Overhead:** Utilizes only a few GPIO pins and simple digital read/write operations.

**Synchronization:** Achieved through the CLOCK\_PIN, ensuring that data integrity is maintained even without high-speed communication hardware.

**Flexibility:** Can be expanded or modified to include more data lines or different signaling methods depending on the application requirements.

**Manual Control:** The Master manually controls the pace of data transmission, which can be advantageous in scenarios where timing is flexible and dictated by application-specific needs.

**Practical Use Cases**

This protocol is particularly useful for applications where two microcontrollers need to communicate small amounts of data reliably without the complexity of full-scale serial communication protocols like UART, SPI, or I2C. Examples include simple sensor data sharing, basic command and control signals, or interfacing with custom hardware.