MA35D0 Edge Device Web Server Example on Linux

Example Code Introduction for 64/32-bit NuMicro® Family

Document Information

Application	The example code shows remote monitoring and control via a web interface, using Node.js and HTML5 for real-time updates on system metrics and device management across multiple units.
BSP Version	Linux-5.10.x
Hardware	NuMaker-IoT-MA35D0 V1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. Overview

The purpose of this example code is to provide the use of Node.js and HTML5 for constructing a remote-control interface within the system. This example code facilitates a connection through a web browser to deliver live updates of system statuses, such as the device's IP address, UID, CPU load, memory capacity, along with I/O status and controls. Additionally, it provides the capability to monitor key events and conduct remote LED controls. This example features its support for inter-device information sharing amongst multiple MA35D0 evaluation board (EVB) units, enhancing collaborative operations and management.

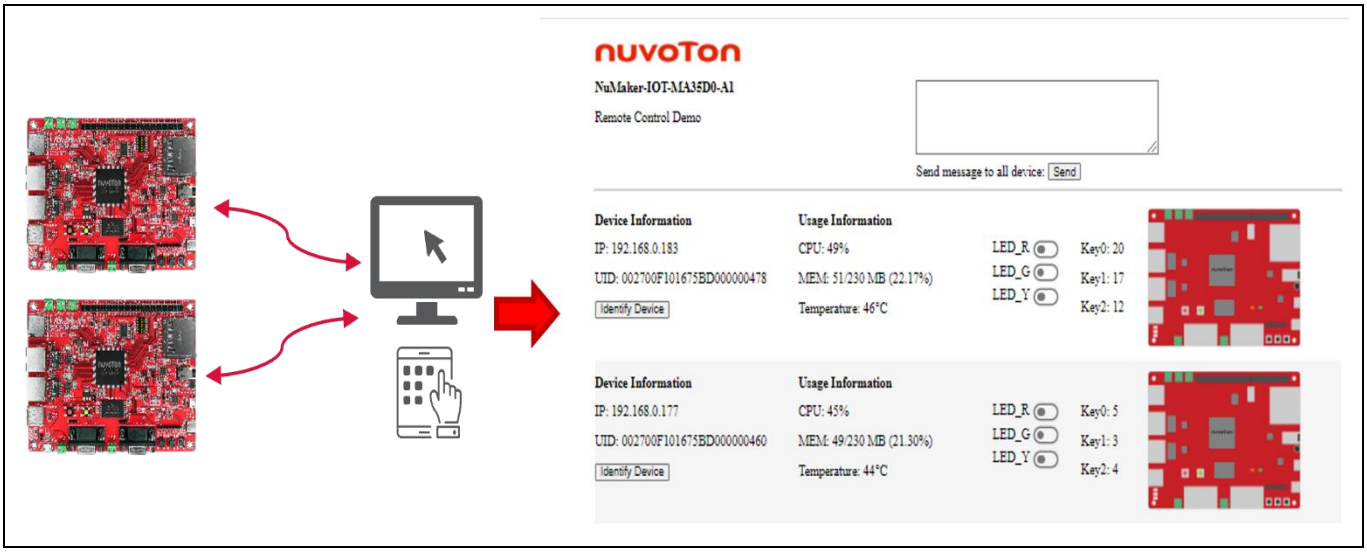

Figure 1-1 IoT Device Remote Monitoring and Control Overview

1.1 Software Architecture

- **init.sh** initializes the demo environment settings and starts the web server (server.js), and calls set_serverIP.sh to allocate IP to the client JavaScript file.
- **set_serverIP.sh** affects the HTML layout and client JavaScript files in the public directory.
- **server.js** server JavaScript file.
- **sys_monitor.sh** script is responsible for obtaining information such as CPU, memory, and temperature.
- **LEDControl.sh** controls the LED's on/off state.
- **identify_device.sh** flashes the LED 6 times to indicate the device has received the corresponding command.
- **get_dev_info.sh** retrieves the device's IP and UID information.

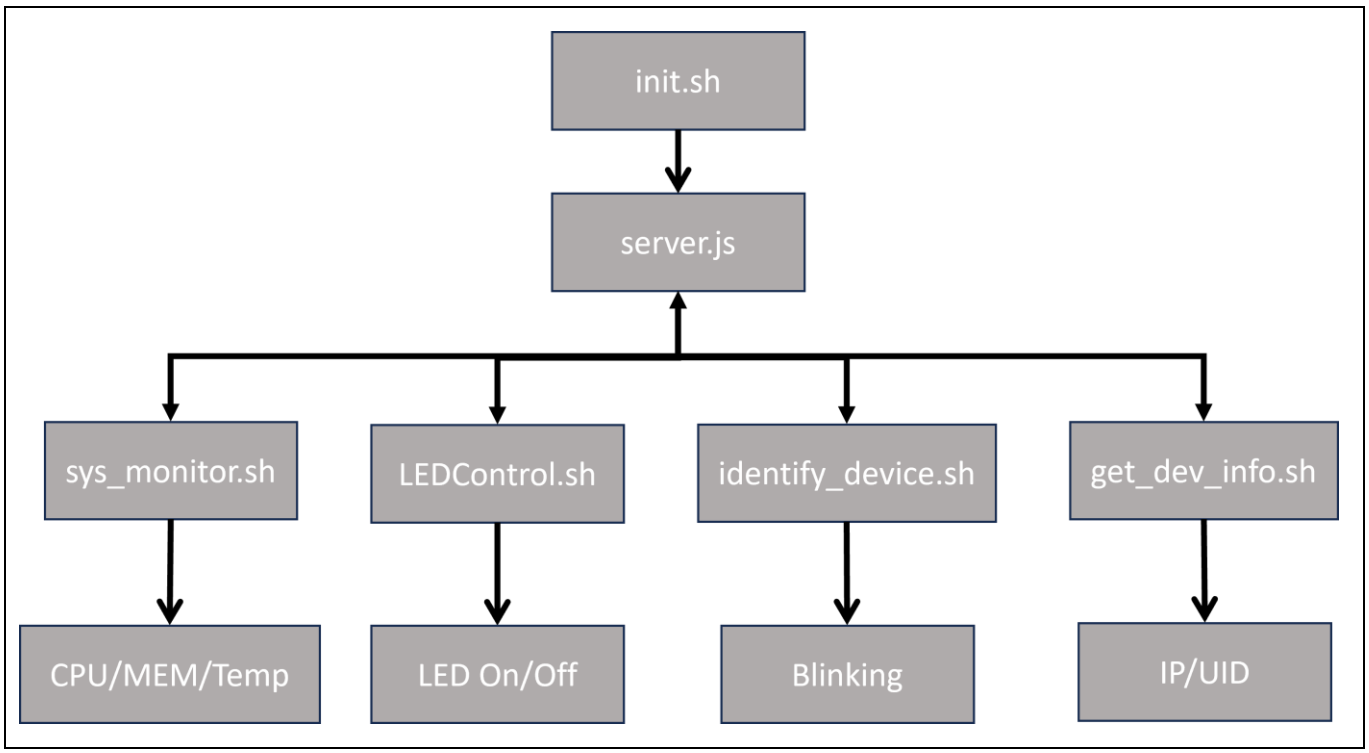

Figure 1-2 Software Framework

1.2 Environment Settings

For MA35D0 environment setup, please refer to the Nuvoton MA35D0 Evaluation Board Quick Start.

- **Buildroot**

To enable the required packages for MA35 support in Buildroot menuconfig, please follow these steps:

1. Open the Buildroot configuration menu by running the command make menuconfig.
2. Search for each package by selecting the corresponding option in the package selection menu and pressing the Spacebar to enable it. It is available to use the '/' key to search for packages by name.
    1) Modify BR2_ROOTFS_OVERLAY="…" to BR2_ROOTFS_OVERLAY=" *board/nuvoton/ma35d0/overlay*".
    2) Enable BR2_PACKAGE_NODEJS to include the Node.js package.
    3) Enable BR2_PACKAGE_NODEJS_NPM to include the Node.js package manager.
    4) Enable BR2_PACKAGE_NODEJS_MODULES_ADDITIONAL= "express ws" to include "express" and "ws" Node.js packages.
3. Once you have enabled all the required packages, exit the menuconfig and save the configuration.

4. Compiling the firmware in Buildroot by executing make. After completion, download the firmware to the MA35D0 development board using NuWriter.

- **Linux Device Tree**

Disable the **gpio_keys_test** driver, and read the register directly to get the I/O status.

ma35d0-iot-256m.dts

```
gpio_keys_test {
    compatible = "gpio-keys";
    status = "disabled"
    …
}
```

1.3 Creating Overlay folder for Example Code

Create a new folder named "overlay" in the path:

${Buildroot TOPDIR}/board/nuvoton/ma35d0/

Place the example code sub-folders (“**opt**” and “**etc**”) in this directory, and Buildroot will copy the contents of the overlay folder to *output/target/* before compiling.

After completing all the steps in sections 1.2 and 1.3, compile the firmware by navigating to the *${Buildroot}/* directory and running the make command.

After compiling, use NuWriter to program the image to the MA35D0 evaluation board.

2. Code Description

2.1 System Initialization “inittab”

The inittab file has the following functions:

- **System Initialization**: It sets up the initial system processes, which include mounting the proc filesystem, remounting the root filesystem in read-write mode, creating */dev/pts* and */dev/shm* directories, mounting all filesystems specified in the */etc/fstab* file, and enabling swap partitions.
- **Creating Symbolic Links**: It creates symbolic links to the current process's file descriptors in */dev/fd*, and to standard *input/output/error*.
- **Setting Hostname**: The system's hostname is set using the contents of the */etc/hostname* file.
- **Executing Custom Initialization Script**: It executes the */opt/init.sh* script, which might contain custom setup instructions for specific hardware or applications.
- **Executing rc Scripts**: Executes the */etc/init.d/rcS* script, which is the system-provided standard script for initializing run levels.
- **Configuring Serial Console**: It sets up a getty process on the ttyS0 serial port, allowing for logins.

2.2 init.sh

During boot-up, this script configures GPIO pins, network settings, and launches a Node.js application:

- **GPIO Configuration**: The script defines two functions, export_gpio and set_direction, to enable specific GPIOs and set their direction.
- **Network Connection:** The script executes the udhcpc command to obtain an IP address for the eth0 network interface from a DHCP server using the DHCP protocol. If this command fails, the script outputs an error message and terminates execution.
- **Export Node.js Module Path:** NODE_PATH=*/usr/lib/node_modules* set the environment variable for the Node.js module path.
- **Run Server Application:** The Node.js server application is launched in the background with the command node server.js &.

2.3 set_serverIP.sh

The purpose of this Shell script is to update the IP address of the eth0 network interface in the JavaScript file (*/opt/public/index.js*).

- **Retrieving IP Address:** The ifconfig command is used to fetch the IP address of the eth0 network interface and this is stored in the eth0_IP variable.
- **Verifying Successful IP Retrieval:** If the IP address is not fetched successfully (meaning the eth0_IP variable is empty), an error message is outputted and the program is terminated.
- **Checking Existence of /opt/public/index.js:** The existence of the file */opt/public/index.js* is verified. If it doesn't exist, the program execution is stopped.
- **Creating a Temporary File:** A temporary file is created using the mktemp command, and the path is stored in the TMP_FILE variable.
- **Writing New First Line:** The first line, containing the eth0 IP address, is written into the temporary file.
- **Appending Existing Content of index.js:** Using the tail command, the entire content from the second line of *the /opt/public/index.js* file is appended to the temporary file.
- **Replacing Original File with Modified File:** Finally, the mv command is used to rename the newly modified temporary file as */opt/public/index.js*, effectively replacing the original index.js file.

2.4 server.js

The script sets up a server that serves static files, establishes WebSocket connections with clients, and facilitates communication between the server and various shell scripts for monitoring and controlling the system's I/O, system status, LEDs, and device identification.

- **Module Import:** The required modules, Express, WebSocket, and child process are imported. These are used to create web servers and to execute Shell scripts.
- **Building the Express Application:** An Express-based application is set up to handle HTTP requests and to provide static file services.
- **Listening on Port:** The HTTP server is initiated on Port 3000, enabling client access via the internet.
- **Setting Server IP:** The Shell script set_serverIP.sh is run to set the server's IP address. Consequently, the client can connect to the correct server.
- **Retrieving Device Information:** The get_dev_info.sh script is executed to fetch the device's IP address and UID. The information is saved in the devInfo object and is used to identify the connected device.
- **Initiating WebSocket Server:** A WebSocket server is launched on port 121 to establish real-time communication with the client.

- **Handling Client Connection and Disconnection**: Corresponding handling functions are available when the client connects or disconnects. These are used for tracking connected clients and update clients' statuses when necessary.
- **Regularly Sending I/O and System Status:** A timer is used to regularly check and send device I/O and system status to all.
- **Handling WebSocket Messages:** Defines functions to process WebSocket messages received from clients, executing corresponding device control or feedback operations based on the received commands.

2.5 sys_monitor.sh

The script monitors the status of GPIO pins, LEDs, and various system resources such as CPU, RAM, and temperature:

- **read_gpio_value:** Reads the value of a specified GPIO pin.
- **io_status:** Outputs the status of keys connected to GPIOs and the brightness levels of three connected LEDs.
- **sys_status:** Outputs the system's CPU utilization, RAM usage, and temperature values.
- **cpu_usage:** Retrieves the CPU usage using the top command.
- **ram_usage:** Retrieves the RAM usage using the free command.

2.6 LEDControl.sh

The script controls the brightness of three LEDs (LED0, LED1, and LED2).

- The script checks if the user has entered a valid LED name, i.e., either "LED0", "LED1", or "LED2". If not, the script will output an error message "Invalid LED".
- If the entered LED name is valid, the script will then validate the second input parameter, which is the target state of the LED. If the target state is "On", the script will apply this setting to the corresponding */sys/class*.

2.7 identify_device.sh

Blinking effect with three LEDs (LED0, LED1, and LED2).

2.8 get_dev_info.sh

The script obtains the IP address of the device's eth0 network interface and a unique identifier (UID):

- **Retrieving eth0 IP address**: Using the ifconfig command to obtain the IP address of eth0 and store it in the eth0_IP variable.
- **Reading memory addresses to form UID**: Using the devmem command to read specific memory addresses and then concatenate these addresses to form the UID.

2.9 index.html

Web page layout design. This webpage mainly includes the following sections:

- **Device Information:** This section will display the device's IP address and Unique Identifier (UID). Readers can use this information to identify connected devices.
- **Usage Information:** This part will show the device's CPU utilization, memory status, and temperature. Users can use this information to understand the operating condition of the device.
- **LED Lights and Key Status:** This section allows users to remotely control and view the status of the LED lights on the device, including red (LED0), green (LED1), and yellow (LED2) lights. "On" represents turned on, and "Off" represents turned off. The webpage will also display the current status of Key0, Key1, and Key2.
- **Send Messages to All Devices:** Users can enter messages in the text area and then click the 'Send' button to send the message to all devices.

2.10 index.js

The script discovers and interacts with multiple devices on the local network through WebSocket connections. It allows users to control LED lights, identify devices, and send messages to the device's console. The user interface dynamically updates based on the status information received from the device, with the main functions as follows:

- **Establishing WebSocket connections:** The code scans a specific range of IP addresses and attempts to establish a WebSocket connection for each IP address. If a connection is successfully established, the code creates a corresponding device object and stores it in the deviceProfile array.
- **Handling WebSocket messages:** An event handler is registered for each established WebSocket connection to handle open, error, close, and received message events. If a WebSocket receives a message from the server, the code will parse it and update the status and user interface of the corresponding device.
- **Dynamically generating device status interface**: Through the duplicateElement function, a new device status block is copied and inserted onto the webpage for each discovered device. This device status block will display information such as the device's IP, UID, button press counts, and temperature.
- **Device identification and LED control:** The identifyDevice and controlLED functions are used to send device identification requests and control the opening or closing of LED lights, respectively.
- **Sending messages to all devices:** The sendMessageToAllDevices function sends a specific message to all connected devices.

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - Linux-5.10.x

3.2 Hardware Requirements

- Circuit components
  - NuMaker-IoT-MA35D0 V1.0

4. Directory Information

The directory structure is shown below.

EC_MA35D0_Edge_Device_WEB_Server_Example_On_Linux_V1.00	
SampleCode	
etc	
inittab	Script to handle boot-up process
opt	
get_dev_info.sh	Procedure to obtain IP and UID
identify_device.sh	Logic to handle LED blinking
init.sh	Custom script for handling boot procedures
LEDControl.sh	Script for managing LEDs
server.js	Server-side application
set_serverIP.sh	Update IP in the index.js file
sys_monitor.sh	Procedure to obtain CPU/MEM usage and I/O status
Images	Images for the web interface
public	Design of the web page layout
index.html	The client-side application
index.js	

Figure 4-1 Directory Structure

5. Example Code Execution

Initial Setup and LED Blinking Behavior:

Connect the MA35D0 ethernet port (RMII0) to the router, and connect the PC side to the Console along with TYPE C for power supply.Upon booting, the default system will run the example successfully and the LED will flash synchronously 6 times.

Accessing MA35D0 Device Information through the Console and Browser:

If the demo runs successfully, the IP address and related information will be displayed. Use a PC or mobile phone in the same network domain, and enter ${MA35D0 IP}:3000 into the browser.

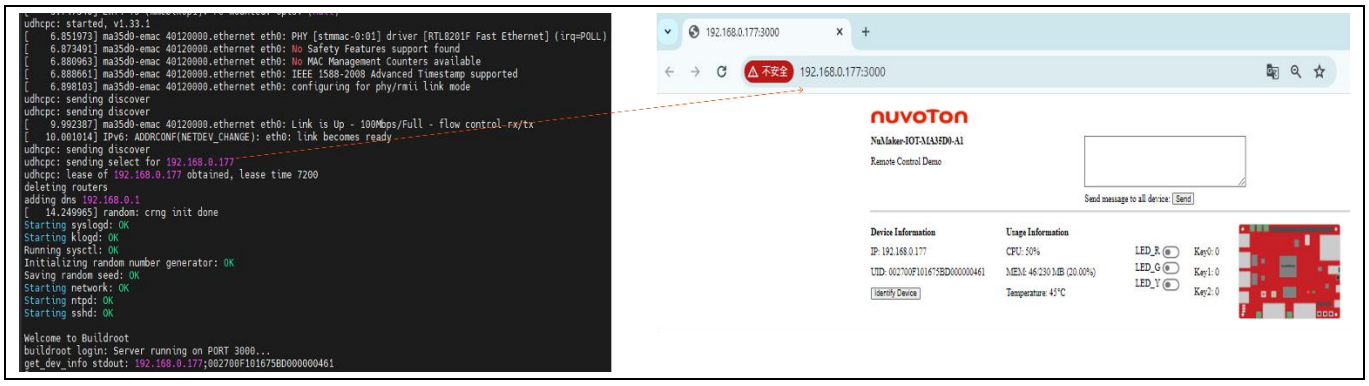

Figure 5-1 Accessing MA35D0 Device Information

Broadcast:

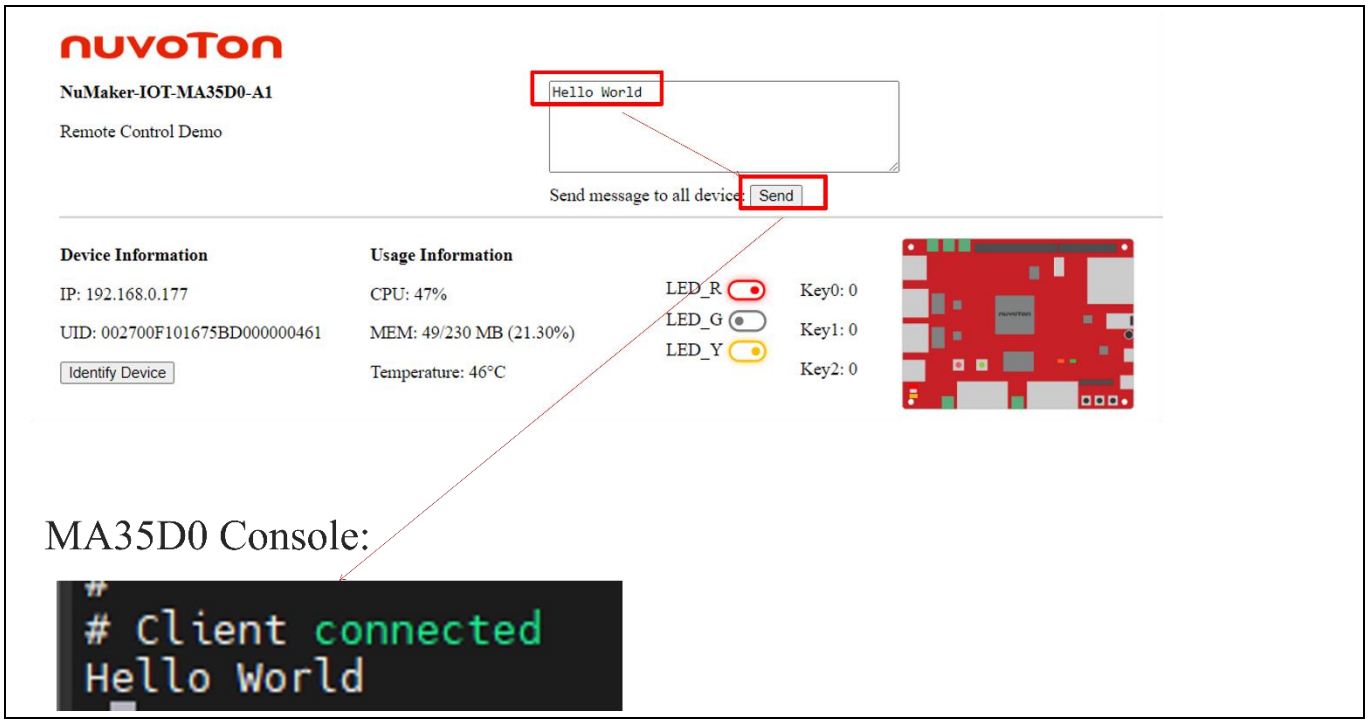

Figure 5-2 Broadcast

LED Controls and Key Status:

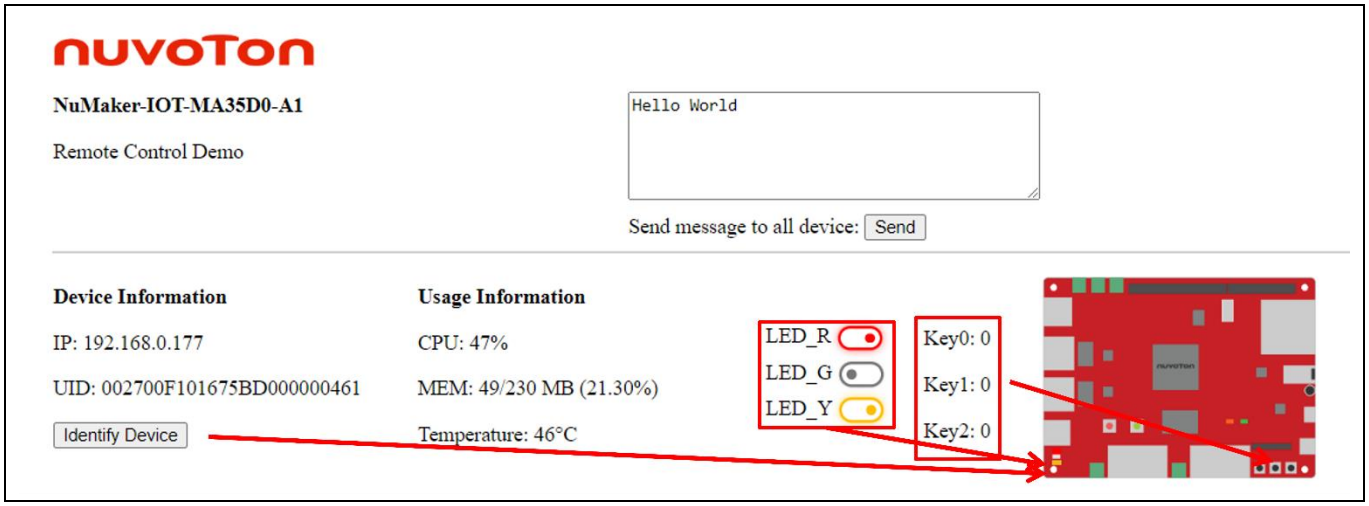

Figure 5-3 Controls and Key Status

Multiple MA35D0 EVBs Share Information:

Power up the other EVB and connect to the console; modify the initialization script at */opt/init.sh* to change the MAC address.

init.sh code snippet

```
#Set new MAC address
ifconfig eth0 down
ifconfig eth0 hw ether 22:33:44:55:66:77
```

Reboot the EVB and check the new IP address.

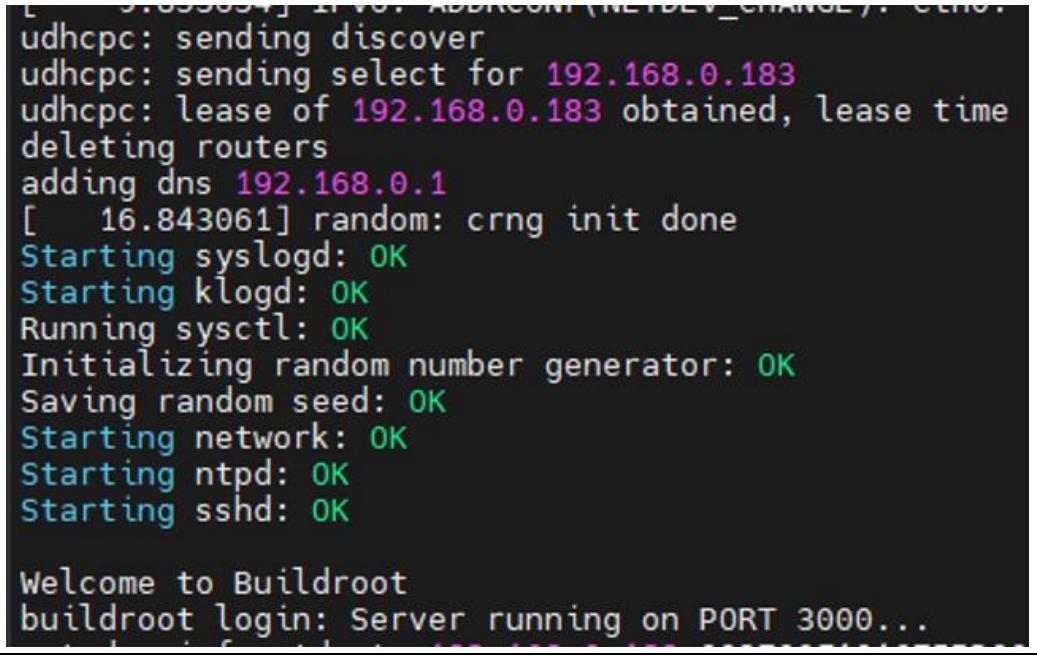

Figure 5-4 Check New IP Address

Enter ${MA35D0 IP}:3000 into the browser to enter each EVB web server.

Through the browser, you can observe the status of each MA35D0 on the server of the two EVBs.

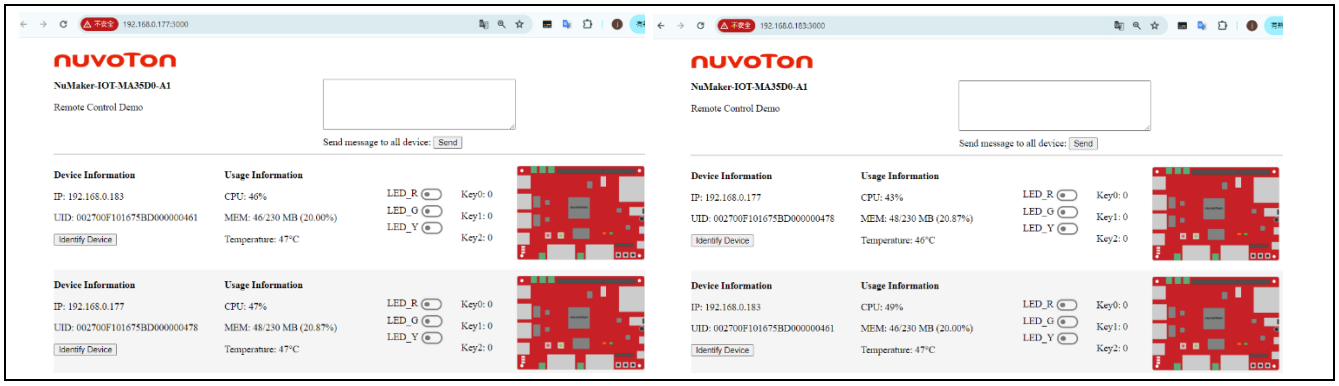

Figure 5-5 EVBs Status

6. Revision History

Date	Revision	Description
2024.05.13	1.00	Initial version.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.