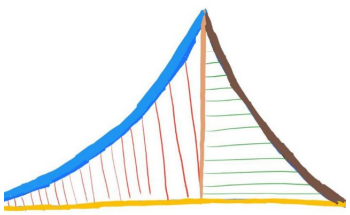
EMBEDDED SYSTEM

G. V. V. Sharma



Contents

Introduction

This book introduces Embedded Systems through using the Vaman framework.

Chapter 1

Vaman-ESP32

This chapter contains various experiments that can be performed using the Vaman-ESP32.

1.1. Flash Vaman-ESP32 Using Arduino

1.1.1. Connections

1.1.1.1. Make sure that Vaman board do not power any devices.

1.1.1.2. Make connections as shown in Table 1.1 and Figure 1.1.

1.1.1.3. The Vaman pin diagram is available in Figure 1.2.

VAMAN LC PINS	ARDUINO PINS
3.3	3.3
GND	GND
TXD0	TXD
RXD0	RXD
0	GND
EN	GND

Table 1.1:

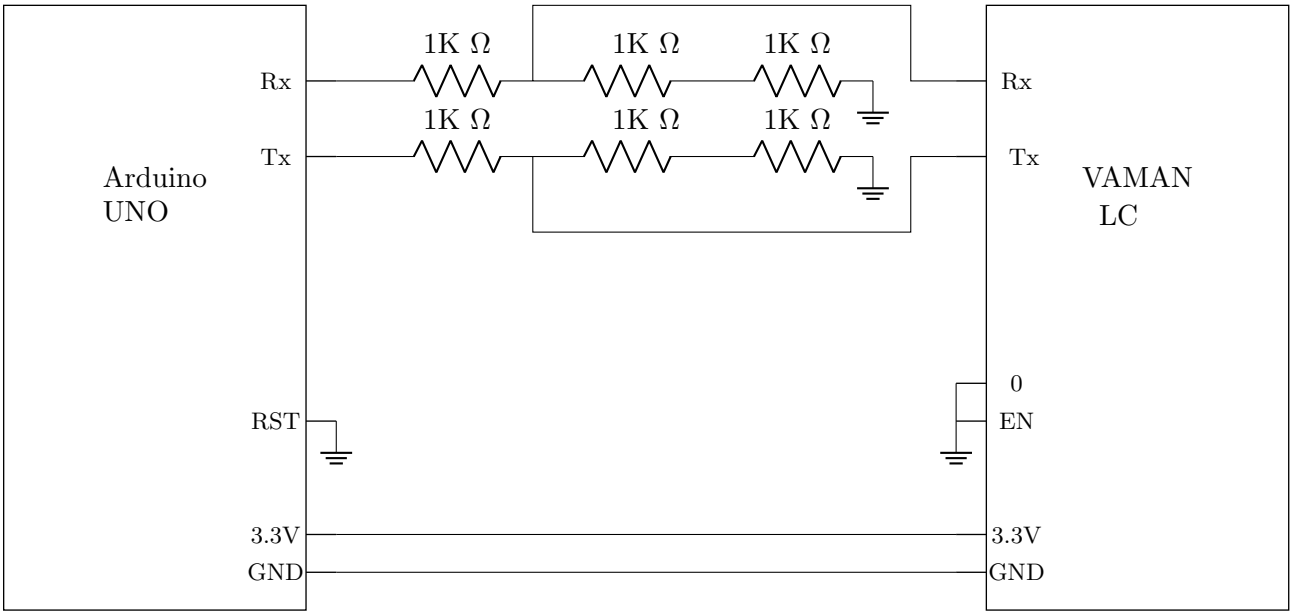


Figure 1.1: Circuit Connections

1.1.2. Building and Flashing

1.1.2.1. For compiling and genrating the bin file, make sure that the platformio.ini file contains these lines.

```
[env:esp32doit−devkit−v1]
platform = espressif32
board = esp32doit−devkit−v1
framework = arduino
platform_packages = toolchain−xtensa−esp32@https://github.com/esphome/
    esphome−docker−base/releases/download/v1.4.0/toolchain−xtensa32.tar.gz
framework−arduinoespressif32@<3.10006.210326
```

1.1.2.2. For uploading bin file to Vaman through ArduinoDroid application:

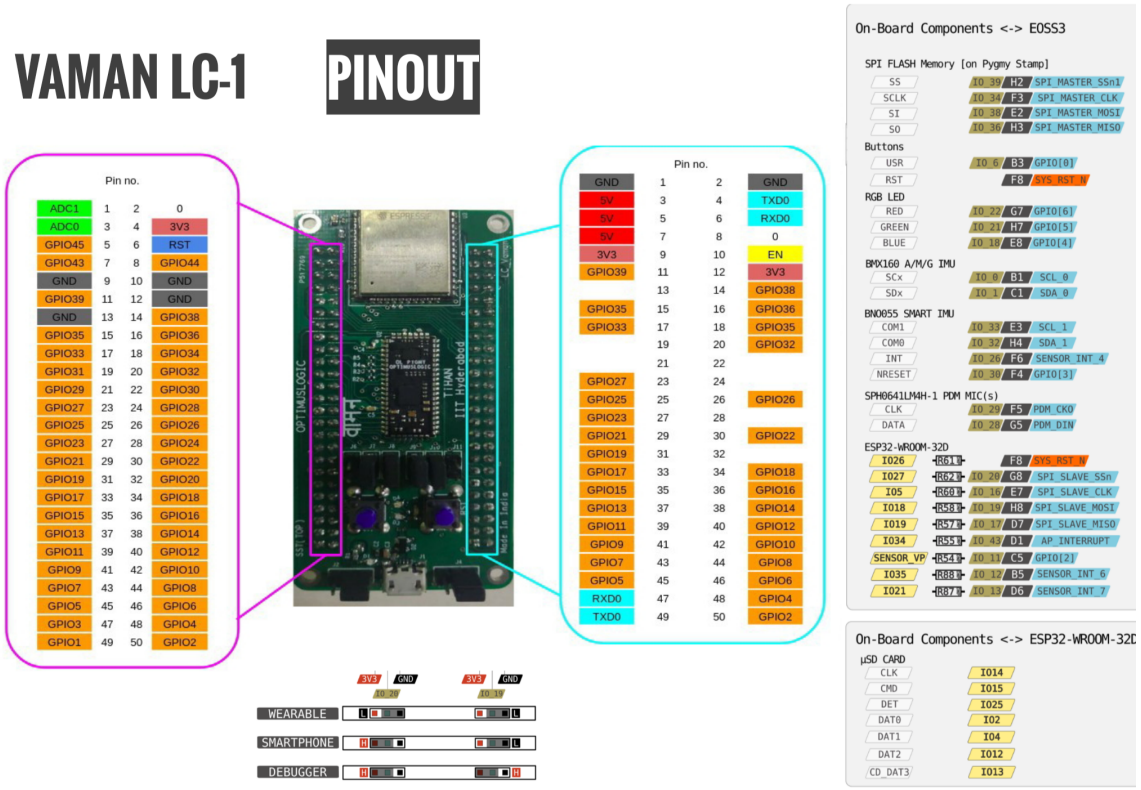


Figure 1.2: Vaman pins

(a) Open the ArduinoDroid application.

(b) Click the three dots in the top right corner.

(c) Navigate to Settings → Board Type.

(d) Select ESP32 → DOIT ESP32 DEVKIT V1.

(e) Change the upload speed to 115200.

(f) Upload the generated .bin file.

1.1.2.3. While the dots are printed on the screen, disconnect the EN wire from GND. Make sure that the Vaman board is not powering any device while flashing. The Vaman-ESP32 should now flash.

1.1.2.4. After flashing, disconnect pin 0 on Vaman-ESP32 from GND. Power on Vaman appropriately.

1.2. Measuring Unknown Resistance

This section describes how to measure an unknown resistance through Vaman-ESP32 and display it on an LCD.

1.2.1. Components

The components required are listed in Table 1.2.

1.2.2. Setting up the Display

1.2.2.1. Plug the LCD in Figure 1.3 to the breadboard.

Component	Value	Quantity
Resistor	220 Ohm	1
	1K	1
ESP32	Devkit V1	1
Jumper Wires		20
Bread board		1
LCD	16 X 2	1
Potentiometer	10K	1

Table 1.2: Components

1.2.2.2. Connect the Vaman-ESP pins to LCD pins as per Table 1.3. Make sure that all 5V sources are connected to the LCD through a 220 Ω resistance.

1.2.2.3. The Vaman pin diagram is available in Figure 1.2.

1.2.2.4. Execute the following code after editing the wifi credentials

```
vaman−esp/lcd/codes/setup
```

You should see the following message

```
Hi
This is CSP Lab
```

1.2.2.5. Modify the above code to display your name.

1.2.3. Measuring the resistance

1.2.3.1. Connect the 5V pin of the Vaman-ESP32 to an extreme pin of the Breadboard shown in Figure 1.4. Let this pin be V_{cc}.

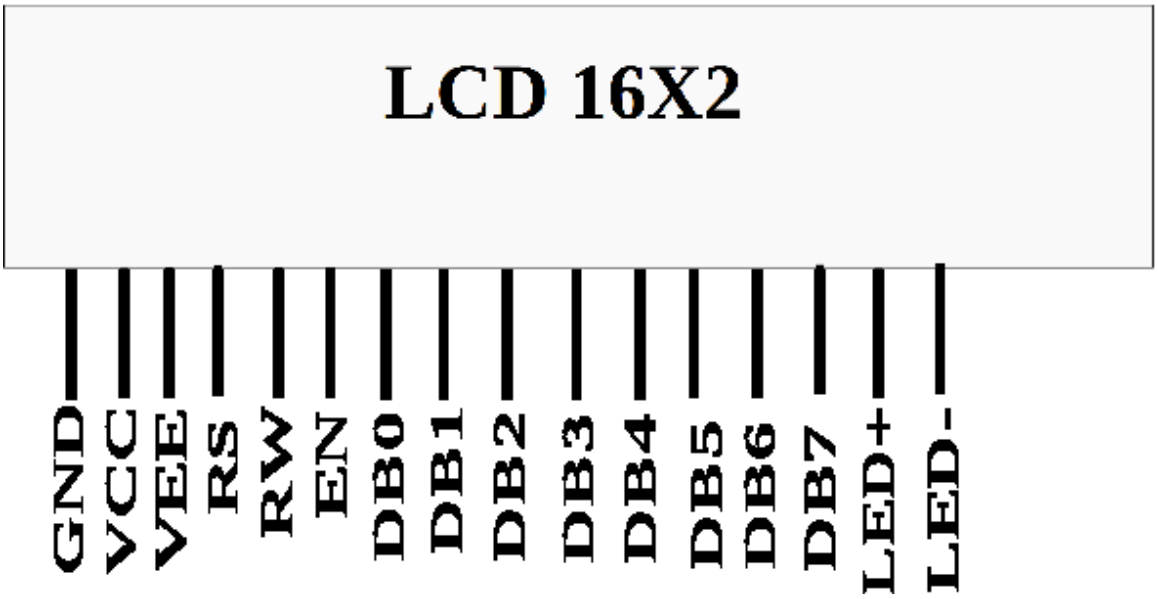


Figure 1.3: LCD pins

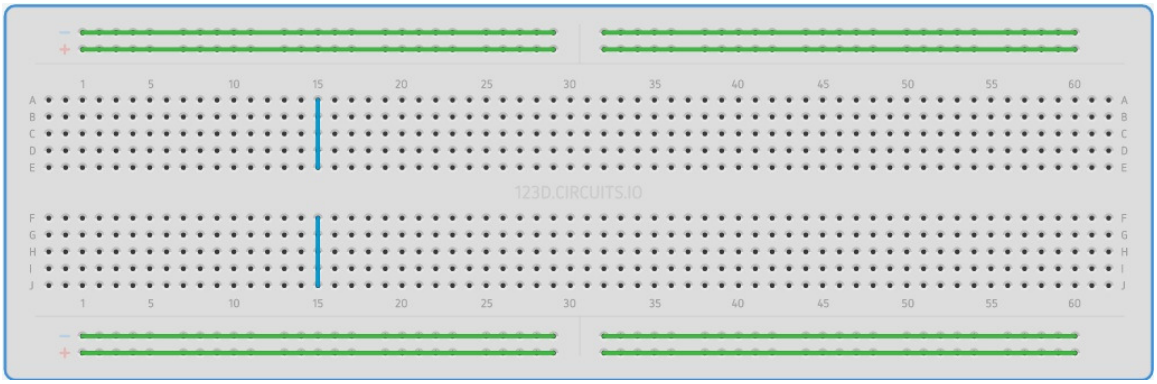


Figure 1.4: Breadboard

1.2.3.2. Connect the GND pin of the Vaman-ESP to the opposite extreme pin of the Breadboard.

1.2.3.3. Let R_1 be the known resistor and R_2 be the unknown resistor. Connect R_1 and R_2 in series such that R_1 is connected to V_{cc} and R_2 is connected to GND. Refer to Figure 1.5.

ESP32	**LCD Pins**	**LCD Pin Label**	**LCD Pin Description**
GND	1	GND	
5V	2	Vcc	
GND	3	Vee	Contrast
GPIO 19	4	RS	Register Select
GND	5	R/W	Read/Write
GPIO 23	6	EN	Enable
GPIO 18	11	DB4	Serial Connection
GPIO 17	12	DB5	Serial Connection
GPIO 16	13	DB6	Serial Connection
GPIO 15	14	DB7	Serial Connection
5V	15	LED+	Backlight
GND	16	LED-	Backlight

Table 1.3: Make sure that all 5V sources are connected to the LCD through a 220 Ω resistance.

1.2.3.4. Connect the junction between the two resistors to the GPIO34 pin on the Vaman-ESP32.

1.2.3.5. Connect the Vaman-ESP to the computer so that it is powered.

1.2.3.6. Execute the following code after editing the wifi credentials

vaman/vaman−esp/lcd/codes/resistance

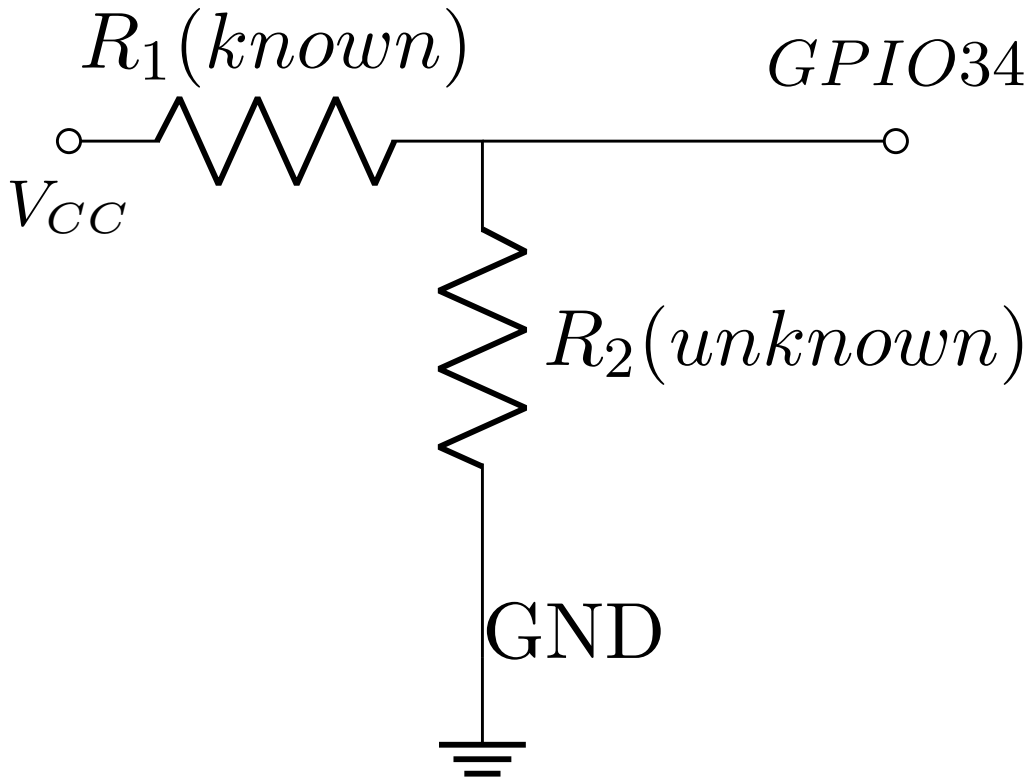


Figure 1.5: Voltage Divider

1.2.4. Displaying the Measured resistance on LCD and website

1.2.4.1. The unknown resistance is measured and diplayed the measured resistance on the LCD display and also on the Vaman-ESP webserver.

1.2.4.2. Connect the Vaman-ESP pins to LCD pins as per Table 1.3.

1.2.4.3. Execute the following code after editing the wifi credentials

vaman−esp/lcd/webserver/codes

1.2.4.4. After flashing the code to vaman-ESP, the board will be connected to the wifi credentials provided.

1.2.4.5. Now connect the same WiFi credentials to the mobile phone for accessing the IP address, which can be accessed by

```
ifconfig
nmap −sn 192.168.x.x/24
```

1.2.4.6. Change the IP address in the second command accordingly with the IP address provided by first command.

1.2.4.7. By the above commands the IP address of vaman-ESP will be diplayed.

1.2.4.8. Now the vaman-ESP will be hosting a webserver

1.2.4.9. Inorder to access the webserver type the IP address of the vaman-ESP in the web browser.

1.2.4.10. In the website loaded by the IP address of vaman-ESP the Unknown resitance is displayed as shown in Figure 1.6

1.2.5. Explanation

1.2.5.1. We create a variable called analogPin and assign it to 0. This is because the voltage value we are going to read is connected to analogPin GPIO34.

1.2.5.2. The 12-bit ADC can differentiate 4096 discrete voltage levels, 5 volt is applied to 2 resistors and the voltage sample is taken in between the resistors. The value which we get from analogPin can be between 0 and 4095. 0 would represent 0 volts falls across

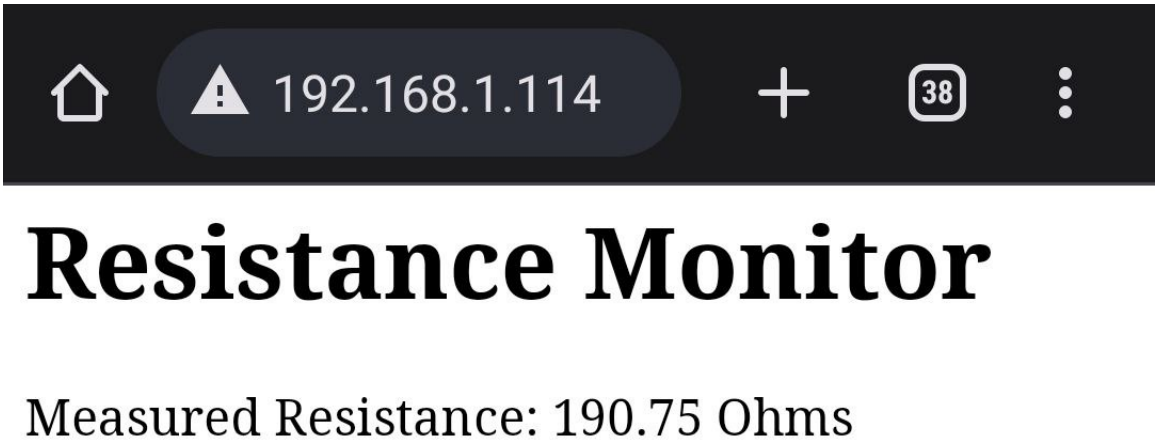


Figure 1.6: Website

the unknown resistor. A value of 4095 would mean that practically all 5 volts falls across the unknown resistor.

1.2.5.3. V_{out} represents the divided voltage that falls across the unknown resistor.

1.2.5.4. The Ohm meter in this manual works on the principle of the voltage divider shown in Figure 1.5.

$$V_{out} = \frac{R_1}{R_1 + R_2} V_{in} \tag{1.1}$$

$$\Rightarrow R_2 = R_1 \left(\frac{V_{in}}{V_{out}} - 1 \right) \tag{1.2}$$

In the above, $V_{in} = 5\text{V}$, $R_1 = 220\Omega$.

1.2.5.5. Repeat the exercise with another unknown resistance.

1.3. I2C Communication Between Vaman-ESP32 and Arduino

This section describes how to setup the Vaman-ESP32 as a Master and Arduino as a slave using I2C protocol.

1.3.1. Components

The components required are listed in Table 1.4.

Component	Value	Quantity
ESP32	Devkit V1	1
Arduino	UNO	1
Connecting Wires		30
LCD	16 X 2	1

Table 1.4: Components.

1.3.2. Setting up the Master and Slave

1.3.2.1. Connect the Vaman-ESP32 pins to Arduino pins as per Table 1.5.

I2C	ESP32	Arduino
SDA	GPIO 21	A4
SDC	GPIO 22	A5
	VCC	VCC
	GND	GND

Table 1.5: Connections Between Arduino and Vaman-ESP32.

1.3.2.2. Connect the Vaman-ESP32 pins to LCD pins as per Table 1.3.

1.3.2.3. The Vaman pin diagram is available in Figure 1.2.

1.3.2.4. Configure Arduino Uno as a Slave using the following PlatformIO project.

vaman−esp32/i2c/codes/I2C_Sender_Arduino

1.3.2.5. Now configure Vaman-ESP32 as a Master using the following PlatformIO project.

vaman−esp32/i2c/codes/I2C_Reciever_ESP32

1.4. I2C Communication between Vaman-ESP32 and Two Arduinos

This section describes how to setup the Vaman-ESP32 as a master and two Arduinos as a slave using I2C protocol. The two unknown resistances are measured by using two Arduinos and sending those two resistance values to Vaman-ESP32 through I2C and displaying the unkwnown resistances on ESP32 webserver.

1.4.1. Components

The components required are listed in Table 1.6.

1.4.2. Setting up one Master and two slaves

1.4.2.1. Connect the vaman-ESP pins to Arduino pins as per Table 1.7.

1.4.2.2. The Vaman pin diagram is available in Figure 1.2

Component	Value	Quantity
Resistor	220 Ohm	1
	2K Ohm	1
	1K Ohm	2
Vaman	LC	1
Arduino	UNO	2
Jumper Wires		20
Bread board		1

Table 1.6: Components

I2C	**Vaman-ESP32**	**Arduino-1**	**Arduino-2**
SDA	GPIO 21	A4	A4
SCL	GPIO 22	A5	A5
		VCC	VCC
		GND	GND

Table 1.7:

1.4.2.3. Configure Arduino Uno as a Slave-1 using the following PlatformIO and upload it.

vaman−esp/i2c−resistance/codes/I2C_Sender_Arduino1

1.4.2.4. Configure Arduino Uno as a Slave-2 using the following PlatformIO and upload it.

vaman−esp32/i2c−resistance/codes/I2C_Sender_Arduino2

1.4.2.5. Now configure vaman-ESP as a Master using the following code and upload it.

vaman−esp32/i2c−resistance/codes/I2C_Reciever_ESP32

1.4.3. Measuring the resistance

1.4.3.1. Connect the 5V pin of the Vaman-ESP32 to an extreme pin of the breadboard shown in Figure 1.4. Let this pin be V_{cc}.

1.4.3.2. Connect the GND pin of the Vaman-ESP32 to the opposite extreme pin of the Breadboard.

1.4.3.3. Let R_1 be the known resistor of 1k ohm and R_2 be the unknown resistor. Connect R_1 and R_2 in series such that R_1 is connected to V_{cc} and R_2 is connected to GND. Refer to Figure 1.5.

1.4.3.4. Connect the junction between the two resistors to the A0 pin on the Arduino board 1, which measures the first unknown resistance.

1.4.3.5. Connect another junction between the two resistors to the A0 pin on the Arduino board 2, which measures the second unknown resistance.

1.4.3.6. Now power the Vaman board

1.4.3.7. Execute the following PlatformIO project after editing the WiFi credentials

vaman−esp32/i2c−resistance/codes/I2C_Reciever_ESP32

1.4.4. Displaying the Measured resistance on website

1.4.4.1. The two unknown resistances are measured and diplayed the measured resistance on the Vaman-ESP32 webserver.

1.4.4.2. After flashing the code to Vaman-ESP32, the board will be connected to the wifi credentials provided.

1.4.4.3. Now connect the same WiFi credentials to the mobile phone for accessing the IP address, which can be accessed by typing the following commands.

```
ifconfig
nmap −sn 192.168.x.x/24
```

1.4.4.4. Change the IP address in the second command accordingly with the IP address provided by first command.

1.4.4.5. By the above commands the IP address of Vaman-ESP32 will be diplayed.

1.4.4.6. Now the Vaman-ESP32 will be hosting a webserver.

1.4.4.7. Inorder to access the webserver type the IP address of the Vaman-ESP32 in the web browser.

1.4.4.8. In the website loaded by the IP address of Vaman-ESP32 the two unknown resitances are displayed as shown in Figure 1.7.

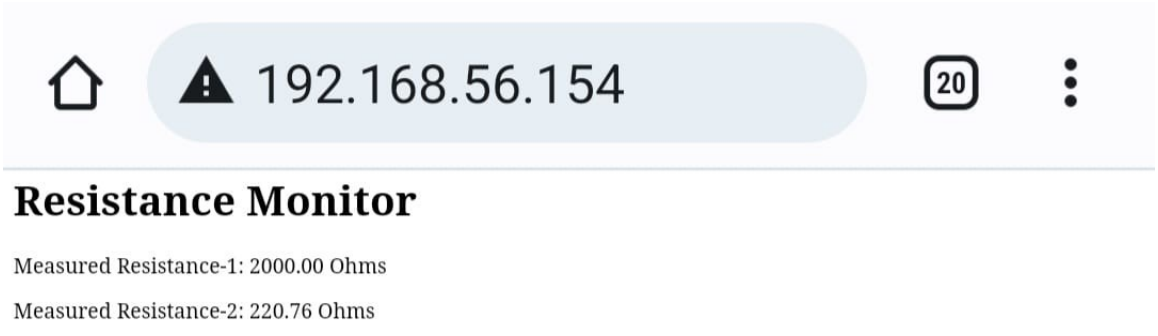


Figure 1.7: Website

1.5. UART Communication between Vaman-ESP32 and Arduino

This section describes how to communicate between Vaman-ESP32 and Arduino UNO through UART Protocol. The Unknown resistance is measured using Arduino and sending the value to Vaman through UART and displaying the unknown resistance on ESP32 Webserver.

1.5.1. Components

The components required are listed in Table 1.8.

Component	Value	Quantity
Resistor	220 Ohm	1
	1K	1
Vaman	LC	1
Arduino	UNO	1
Jumper Wires		10
Bread board		1

Table 1.8: Components

1.5.2. Connections

1.5.2.1. Connect the Vaman and Arduino as shown in Table 1.9.

Arduino UNO	Vaman-ESP
Rx(Pin-0)	17 (Tx)
Tx (Pin-1)	16 (Rx)

Table 1.9: Connections

1.5.2.2. The Vaman pin diagram is available in Figure 1.2.

1.5.2.3. Upload the following code to Arduino UNO

vaman−esp32/uart/codes/UNO

1.5.3. Measuring the resistance

1.5.3.1. Connect the 5V pin of the Vaman-ESP32 to an extreme pin of the breadboard shown in Figure 1.4. Let this pin be V_{cc}.

1.5.3.2. Connect the GND pin of the Vaman-ESP32 to the opposite extreme pin of the breadboard.

1.5.3.3. Let R_1 be the known resistor and R_2 be the unknown resistor. Connect R_1 and R_2 in series such that R_1 is connected to V_{cc} and R_2 is connected to GND. Refer to Figure 1.5.

1.5.3.4. Connect the junction between the two resistors to the A0 pin on the Arduino board.

1.5.3.5. Now power the Vaman board.

1.5.3.6. Execute the following code after editing the WiFi credentials

vaman−esp32/uart/codes/VAMAN

1.5.4. Displaying the Measured resistance on website

1.5.4.1. The unknown resistance is measured and diplayed the measured resistance on the Vaman-ESP32 webserver.

1.5.4.2. After flashing the code to Vaman-ESP32, the board will be connected to the WiFi credentials provided.

1.5.4.3. Now connect the same WiFi credentials to the mobile phone for accessing the IP address, which can be accessed by

```
ifconfig
nmap −sn 192.168.x.x/24
```

1.5.4.4. Change the IP address in the second command accordingly with the IP address provided by first command.

1.5.4.5. By the above commands the IP address of Vaman-ESP32 will be diplayed.

1.5.4.6. Now the Vaman-ESP32 will be hosting a webserver.

1.5.4.7. In order to access the webserver type the IP address of the Vaman-ESP32 in the web browser.

1.5.4.8. In the website loaded by the IP address of Vaman-ESP32 the unknown resistance is displayed as shown in Figure 1.6.

1.6. SPI Communication between Vaman-ESP32 and Arduino

This section describes how to communicate between Vaman-ESP32 and Arduino UNO through SPI Protocol. Here the Arduino sketch for an ESP32 microcontroller that connects to a WiFi network and communicates with a server using the SPI protocol. It sends a

message character by character to the server via the SPI interface and receives a response. The received response is stored in a JSON document and sent back to the server using an HTTP POST request.

1.6.1. Components

The components required are listed in Table 1.10.

Component	Value	Quantity
Vaman	LC	1
Arduino	UNO	1
Jumper Wires		10
Bread board		1

Table 1.10: Components

1.6.2. Connections

1.6.2.1. Connect the Vaman and Arduino as shown Table 1.11.

PIN ID	Vaman-ESP	Ardunio
CLK	14	D13
MISO/CIPO	12	D12
MOSI/CIPO	13	D11
SS/CS	15	D10
GND	GND	GND

Table 1.11: Connections

1.6.2.2. The Vaman pin diagram is available in Figure 1.2.

1.6.2.3. Upload the following code to Arduino UNO.

```
vaman−esp32/spi/codes/arduino
pio run −t upload
```

1.6.2.4. Execute the following code after editing the WiFi credentials

```
vaman−esp32/spi/codes/esp32
pio run −t upload
```

1.6.2.5. To run the server, you require a Python 3 virtual environment with Flask installed. You can run it by entering the following commands at a terminal window

```
vaman−esp32/spi/codes
python3 −m venv venv
source venv/bin/activate
cd ./server
ifconfig
flask run −−host 192.168.xxx.xxx
```

1.6.2.6. Inorder to access the webserver, type the IP address of the Vaman-ESP32 in the web browser.

1.6.2.7. In the website loaded by the IP address of Vaman-ESP32 is to establish a WiFi connection, exchange data with a server using the SPI protocol, handle received data, and communicate with the server using HTTP requests as shown in Figure 1.8.

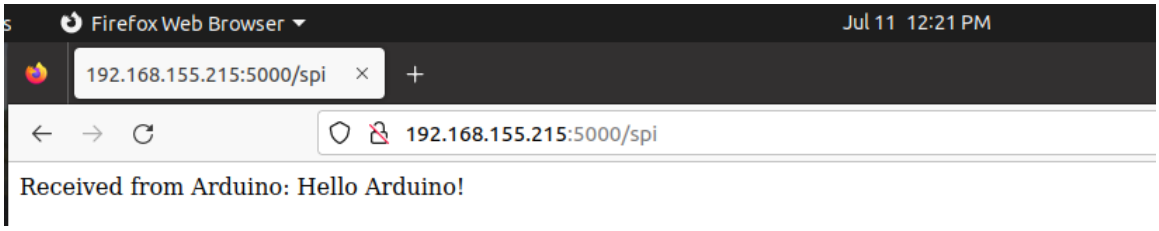


Figure 1.8: Website

1.7. Measuring Unknown Resistance Using SPI

This section describes how to setup the Vaman-ESP32 as a Master and Arduinos as a Slave using SPI protocol. The unknown resistance is measured using Arduino and sending the calculated value to Vaman through SPI and displaying the unknown resistance on ESP-Webserver.

1.7.1. Components

The components required are listed in Table 1.12.

Component	Value	Quantity
Resistor	220 Ohm	1
	1K	1
Vaman	LC	1
Arduino	UNO	1
Jumper Wires		10
Bread board		1

Table 1.12: Components

1.7.2. Connections

1.7.2.1. Connect the Vaman and Arduino as shown Table 1.13.

Arduino UNO	**Vaman-ESP**
Rx(Pin-0)	17 (Tx)
Tx (Pin-1)	16 (Rx)

Table 1.13: Connections

1.7.2.2. The Vaman pin diagram is available in Figure 1.2.

1.7.2.3. Upload the following code to Arduino UNO.

vaman−esp32/spi−resistance/codes/arduino

1.7.3. Measuring the resistance

1.7.3.1. Connect the 5V pin of the Vaman-ESP32 to an extreme pin of the breadboard shown in Figure 1.4. Let this pin be V_{cc}.

1.7.3.2. Connect the GND pin of the Vaman-ESP32 to the opposite extreme pin of the breadboard.

1.7.3.3. Let R_1 be the known resistor and R_2 be the unknown resistor. Connect R_1 and R_2 in series such that R_1 is connected to V_{cc} and R_2 is connected to GND. Refer to Figure 1.5.

1.7.3.4. Connect the junction between the two resistors to the A0 pin on the Vaman-ESP32 GPIO36, which measures the unknown resistance.

1.7.3.5. Now, power the Vaman board.

1.7.3.6. Execute the following code after editing the WiFi credentials.

```
vaman−esp32/spi−resistance/codes/esp32
```

1.7.4. Displaying the Measured resistance on website

1.7.4.1. The unknown resistance is measured and diplayed the measured resistance on the Vaman-ESP32 webserver.

1.7.4.2. After flashing the code to Vaman-ESP32, the board will be connected to the WiFi credentials provided.

1.7.4.3. Now connect the same WiFi credentials to the mobile phone for accessing the IP address, which can be accessed by

```
ifconfig
nmap −sn 192.168.x.x/24
```

1.7.4.4. Change the IP address in the second command accordingly with the IP address provided by first command.

1.7.4.5. By the above commands the IP address of Vaman-ESP32 will be displayed.

1.7.4.6. Now the Vaman-ESP32 will be hosting a webserver.

1.7.4.7. In order to access the webserver type the IP address of the Vaman-ESP32 in the web browser.

1.7.4.8. In the website loaded by the IP address of Vaman-ESP32, the unknown resistance is displayed as shown in Figure 1.9.

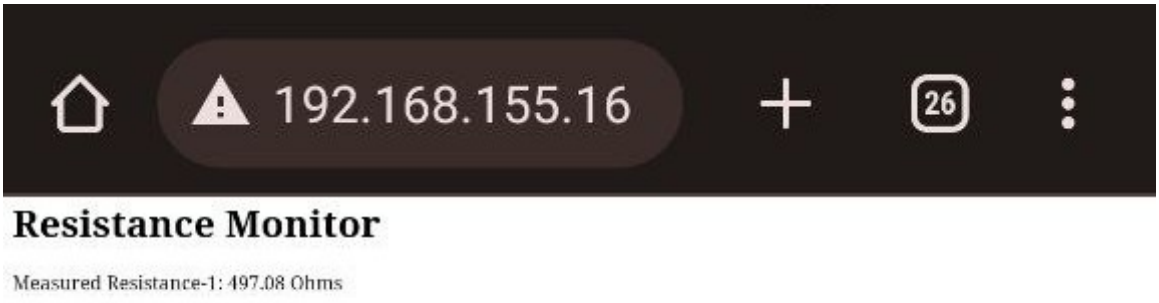


Figure 1.9: Website

1.8. Bluetooth-Controlled Seven Segment Display

This section describes how to control the seven-segment display through the Dabble Android application using Bluetooth, and display an appropriate digit on the seven-segment display according to the controls in the Android app.

1.8.1. Components

The components required are listed in Table 1.14.

1.8.2. Connections

1.8.2.1. Connect the Arduino-UART to VAMAN as per Table 1.1 and Figure 1.1.

1.8.2.2. Now, build the PlatformIO project at

Component		Quantity
Resistor	220 Ohm	1
Seven Segment Display		1
Vaman	LC	1
Arduino	UNO	1
Jumper Wires		10
Bread board		1

Table 1.14: Components

vaman−esp32/bluetooth/codes

1.8.2.3. Install the Dabble Android application and give the necessary permissions.

1.8.2.4. Connect the Bluetooth of Vaman-ESP32 to the mobile where the Bluetooth device name is labelled as “MyEsp32”.

1.8.2.5. Open the Dabble application. Select gamepad option in the app and then select Digital Mode and connect it app to ESP-32 by connecting it ESP-32 bluetooth as shown in Figure 1.10.

1.8.2.6. Now connect the Seven Segment to the Vaman board according to the given Table 1.15

VAMAN ESP pins	Seven Segment pins
IO-32	a
IO-33	b
IO-25	c
IO-26	d
IO-27	e
IO-14	f
IO-12	g

Table 1.15: Connections

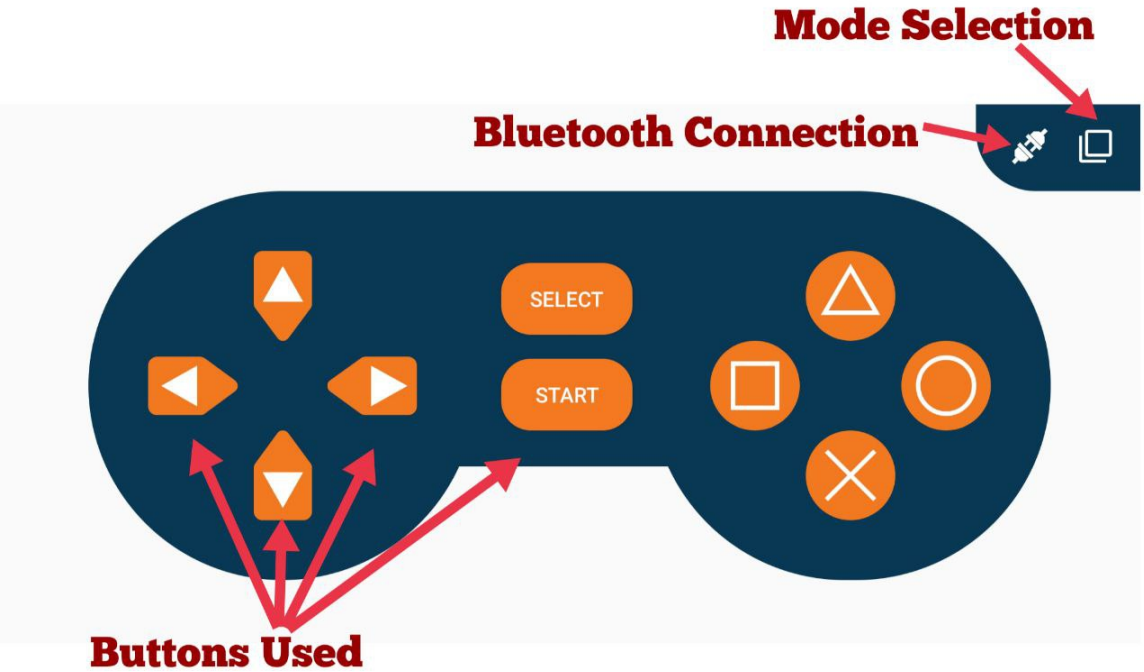


Figure 1.10: Dabble app Interface

1.8.2.7. Now you can observe the changes on seven segment display for Start, Up, Down, Right and Left keys pressed on the Digital Mode on the android application

1.9. WiFi-Controlled Seven Segment Display

This section demonstrates how to control the Seven Segment Display through the Dabble Android application using WiFi and display on the seven segment according to the controls in the Android app.

1.9.1. Connections

1.9.1.1. **Note**: Table 1.14 and Figure 1.1, Table 1.15 are similar to the bluetooth control seven segment display.

1.9.1.2. Now, execute the following code

1.9.1.3. Make sure that change your "ssid","password" in code

```
vaman−esp32/wifi/codes/src
```

1.9.1.4. Build ESP32 firmware.

```
cd vaman−esp32/wifi/codes
pio run
```

1.9.1.5. Flash ESP32 firmware (connect Arduino-UART).

```
pio run −t upload
```

1.9.1.6. Now check that your mobile/tab is connected with ESP32.

1.9.1.7. Install the WiFi Dabble app.

```
vaman−esp32/wifi/Wifi_dabble.apk
```

1.9.1.8. Open the Dabble application. Enter the Vaman-ESP32 IP address as shown in Figure 1.11.

1.9.1.9. Now you can observe the changes on seven-segment display for Start, Up, Down, Right and Left keys pressed on the Android application.

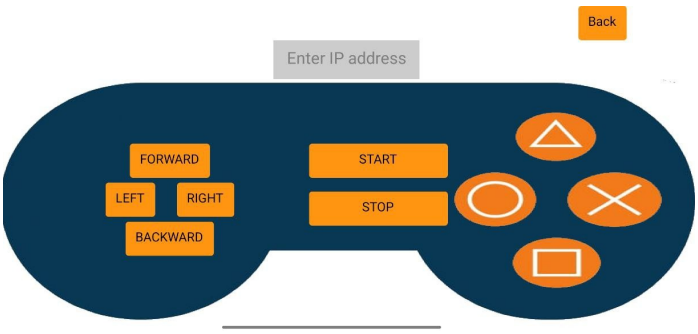


Figure 1.11: Wifi Dabble app

Chapter 2

Maths computing using ESP32

This chapter will demonstrate how to solve math problems using esp.

2.1. Math Computing using ESP32

2.1.1. Show that the points A $\begin{pmatrix} 1 \\ -2 \\ -8 \end{pmatrix}$, B $\begin{pmatrix} 5 \\ 0 \\ -2 \end{pmatrix}$ and C $\begin{pmatrix} 11 \\ 3 \\ 7 \end{pmatrix}$ are collinear, and find the ratio in which B divides AC.

Solution:

The input parameters for this problem are available in Table 2.1

Points **A**, **B** and **C** are on a line if

$$\text{rank}\begin{pmatrix} \mathbf{A} & \mathbf{B} & \mathbf{C} \end{pmatrix} < 3 \tag{2.1}$$

Symbol	Value	Description
A	$\begin{pmatrix} 1 \\ -2 \\ -8 \end{pmatrix}$	First point
B	$\begin{pmatrix} 5 \\ 0 \\ -2 \end{pmatrix}$	Second point
C	$\begin{pmatrix} 11 \\ 3 \\ 7 \end{pmatrix}$	Third point

Table 2.1:

Substituting, we must find the rank of

$$\mathbf{M} = \begin{pmatrix} 1 & 5 & 11 \\ -2 & 0 & 3 \\ -8 & -2 & 7 \end{pmatrix} \tag{2.2}$$

Using row reduction methods to bring $\mathbf{M}$ into row-reduced echelon form,

$$\begin{pmatrix} 1 & 5 & 11 \\ -2 & 0 & 3 \\ -8 & -2 & 7 \end{pmatrix} \xleftrightarrow{R_2 \to R_2 + 2R_1} \begin{pmatrix} 1 & 5 & 11 \\ 0 & 10 & 25 \\ -8 & -2 & 7 \end{pmatrix} \tag{2.3}$$

$$\xleftrightarrow{R_3 \to R_3 + 8R_1} \begin{pmatrix} 1 & 5 & 11 \\ 0 & 10 & 25 \\ 0 & 38 & 95 \end{pmatrix} \tag{2.4}$$

$$\xleftrightarrow{R_3 \to R_3 - \frac{19}{5}R_2} \begin{pmatrix} 1 & 5 & 11 \\ 0 & 10 & 25 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.5}$$

Clearly, the rank of $\mathbf{M}$ is 2, and hence the given points are collinear. Fig. 2.1 verifies that the three points are indeed collinear as claimed.

Let $\mathbf{B}$ divide $\mathbf{AC}$ in k:1 then,

$$\frac{k\mathbf{C} + \mathbf{A}}{k+1} = \mathbf{B} \tag{2.6}$$

$$\implies k\mathbf{C} + \mathbf{A} = \mathbf{B}\,(k+1) \implies k\,(\mathbf{C} - \mathbf{B}) = (\mathbf{B} - \mathbf{A}) \tag{2.7}$$

Multiplying with $(\mathbf{C} - \mathbf{B})^\top$ on both sides,

$$k\,(\mathbf{C} - \mathbf{B})\,(\mathbf{C} - \mathbf{B})^\top = (\mathbf{B} - \mathbf{A})\,\mathbf{C} - \mathbf{B}^\top$$

The value of k is as follows,

$$k = \frac{(\mathbf{B}-\mathbf{A})\,(\mathbf{C}-\mathbf{B})^{\top}}{\|\mathbf{C}-\mathbf{B}\|^{2}} \tag{2.8}$$

where,

$$(\mathbf{B}-\mathbf{A}) = \left(\begin{pmatrix}5\\0\\-2\end{pmatrix} - \begin{pmatrix}1\\-2\\-8\end{pmatrix}\right) = \begin{pmatrix}4\\2\\6\end{pmatrix} \tag{2.9}$$

$$(\mathbf{C}-\mathbf{B}) = \left(\begin{pmatrix}11\\3\\7\end{pmatrix} - \begin{pmatrix}5\\0\\-2\end{pmatrix}\right) = \begin{pmatrix}6\\3\\9\end{pmatrix} \tag{2.10}$$

$$(\mathbf{C}-\mathbf{B})^{\top} = \begin{pmatrix}6 & 3 & 9\end{pmatrix}$$

Substituting the values the value of k is 2/3.

Hence, **B** divides **AC** in the ratio 2 : 3.

2.1.1. C pointer code

2.1.1. The following is the C code for the above question using pointers

vaman/vaman−esp/math_using_esp/codes/cprog/main.c

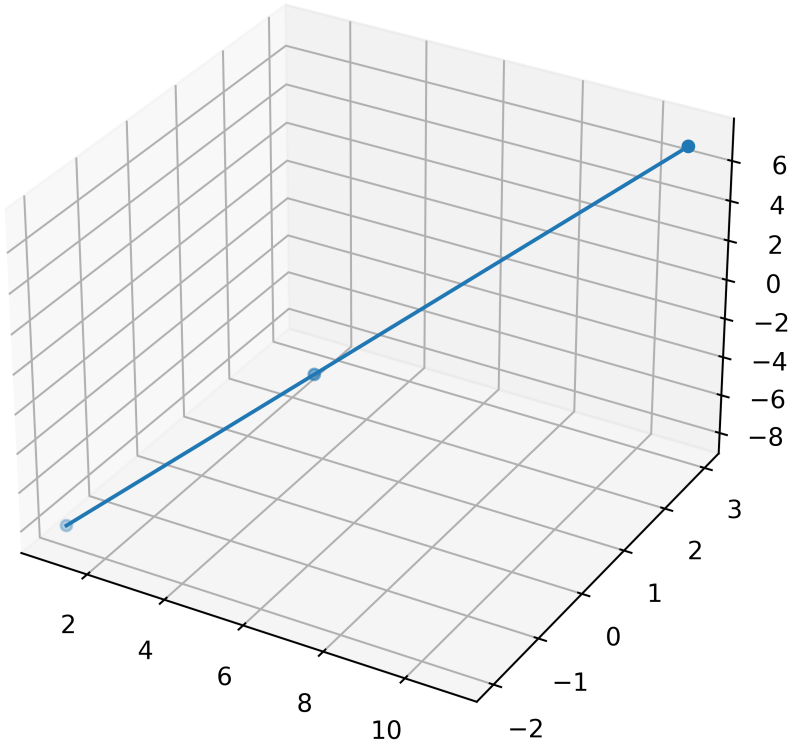


Figure 2.1:

2.2. Code execution for Math Computing using esp32

2.2.1. Now, Execute the following code

math_using_esp/codes/esp_code/src/main.cpp

2.2.2. Build the ESP32 firmware

cd math_using_esp/codes/esp_code

```
pio run
```

2.2.3. Flash ESP32 firmware (using Arduino)

```
pio run −t upload
```

2.2.1. Displaying the output on website

2.2.1. The output of the code i.e; if the points are collinear or not will be displayed in the webserver.

2.2.2. After flashing the code to vaman-ESP, the board will be connected to the wifi credentials provided.

2.2.3. Now connect the same WiFi credentials to the mobile phone for accessing the IP address, which can be accessed by

```
ifconfig
nmap −sn 192.168.x.x/24
```

2.2.4. Change the IP address in the second command accordingly with the IP address provided by first command.

2.2.5. By the above commands the IP address of vaman-ESP will be diplayed.

2.2.6. Now the vaman-ESP will be hosting a webserver

2.2.7. In order to access the webserver type the IP address of the vaman-ESP in the web browser.

2.2.8. In the website loaded by the IP address of vaman-ESP the output is displayed as shown in Fig. 2.2

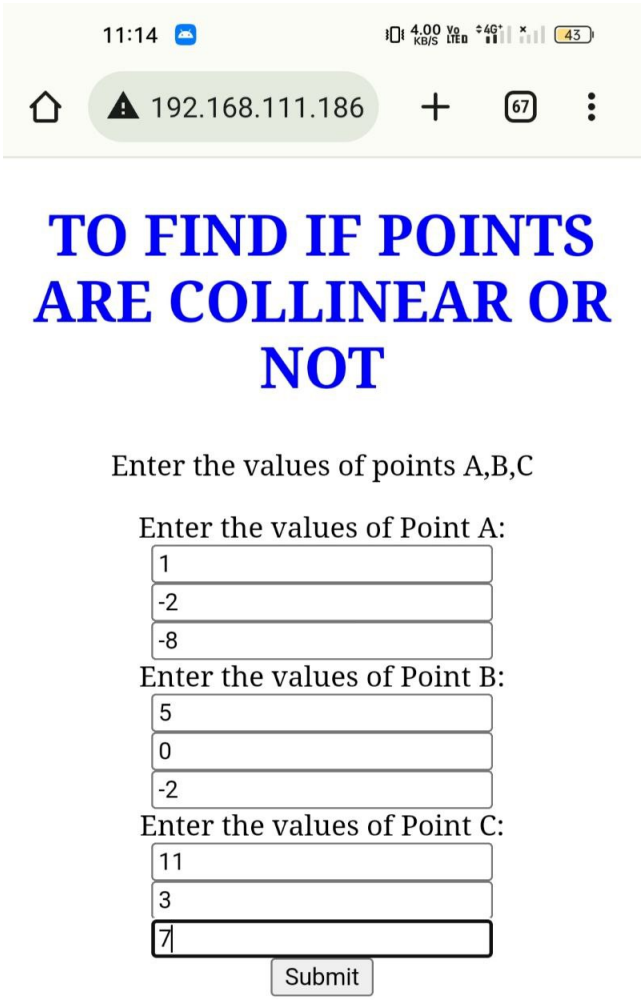


Figure 2.2: Website

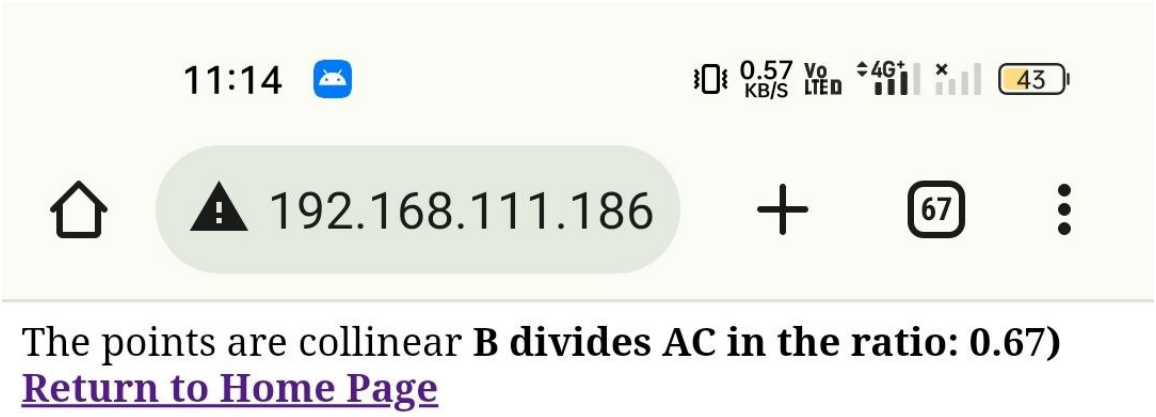


Figure 2.3: Website result

Chapter 3

Inter-Chip Communication

This chapter demonstrates how various platforms on the LC Vaman (ESP32, ARM, FPGA) can communicate with each other through various protocols such as Serial Peripheral Interface (SPI) and the AHB-To-Wishbone bridge on the Pygmy.

3.1. ESP32 and FPGA

This section describes various experiments that interface the ESP32 and eFPGA present on the LC Vaman.

3.1.1. Onboard LED

This subsection illustrates the use of SPI Communication between the Vaman-ESP32 and the FPGA onboard the Vaman-Pygmy by toggling the LEDs onboard the Vaman-Pygmy.

3.1.1.1. Components

The components required are listed in Table 3.1.

Component	Value	Quantity
Vaman	LC	1
USB-UART		1
Jumper Wires	Female-to-Female	20
USB Cable	Type-B	1

Table 3.1: Components Required for Controlling the Onboard LED via SPI.

3.1.1.2. Connections

The connections to be made on the Vaman board are listed in Table 3.2.

Pin	Vaman-ESP32	Vaman-Pygmy
CS/SS	27	IO_20
CIPO/MISO	19	IO_17
COPI/MOSi	18	IO_19
SCLK	5	IO_16

Table 3.2: Connections to Establish SPI between Vaman-ESP32 and Vaman-Pygmy.

3.1.1.3. Building

3.1.1.1. Build the PlatformIO project at

```
inter−chip/esp32−fpga/led/codes/esp32
```

3.1.1.2. Flash the project .bin file using USB-UART connected to the Vaman-ESP32, via PlatformIO or ArduinoDroid.

3.1.1.3. Build the FPGA project .bin file by entering the following commands at a terminal window.

```
# The following variable can be sourced from .shellrc or .venv/bin/activate
export QORC_SDK_PATH=/path/to/pygmy−sdk
```

```
cd inter−chip/esp32−fpga/led/codes/fpga
make
```

3.1.1.4. On building successfully, the .bin file is generated at

```
inter−chip/esp32−fpga/led/codes/fpga/rtl/AL4S3B_FPGA_Top.bin
```

3.1.1.5. Flash the .bin file to the Vaman-Pygmy by resetting it to bootloader mode and entering the following command at a terminal window,

```
python3 /path/to/tinyfpga−programmer−gui.py −−port /dev/ttyACMxx −−mode
    fpga −−appfpga /path/to/AL4S3B_FPGA_Top.bin −−reset
```

where /dev/ttyACMxx is the port at which the Vaman board is available. This can be obtained by inspecting the output of the following command (requires root/sudo privileges).

```
dmesg −w
```

3.1.1.4. Demonstration

3.1.1.6. Find the IP address of the Vaman-ESP32 by inspecting the output of the serial terminal, or by typing at a terminal window

```
ifconfig
nmap −sn xx.yy.zz.0/24
```

where xx.yy.zz represents the first three octets of the IP address of your device on the WiFi network interface, found using `ifconfig`.

3.1.1.7. Then, go to the following website and interact with the radio buttons to toggle the LEDs onboard the Vaman-Pygmy.

```
http://<VAMAN−IP>/led
```

3.1.1.5. Exercises

3.1.1.8. Modify the ESP32 code to have three separate radio buttons for each onboard LED.

3.1.1.9. Modify the ESP32 code to blink the LED periodically when the user toggles a checkbox on the HTML form.

3.1.1.10. SPI transactions are a bottleneck in execution. Minimize the number of SPI transactions and the amount of data transmitted in each of them.

3.1.2. Seven-Segment Display

This subsection illustrates the use of SPI Communication between the Vaman-ESP32 and the FPGA onboard the Vaman-Pygmy by controlling a seven segment display remotely.

3.1.2.1. Components

The components required are listed in Table 3.3.

3.1.2.2. Connections

The connections to be made on the Vaman board are listed in Table 3.2. The connections to be made with the seven segment display are listed in Table 3.4.

Component	Value	Quantity
Vaman	LC	1
USB-UART		1
Jumper Wires	Female-to-Female	10
Jumper Wires	Male-to-Female	10
USB Cable	Type-B	1
Seven-Segment Display	Common Anode	1
Breadboard		1

Table 3.3: Components Required for Controlling the Seven-Segment Display via SPI.

Seven-Segment Display	Vaman-Pygmy
a	IO_1
b	IO_2
c	IO_3
d	IO_4
e	IO_5
f	IO_6
g	IO_7
COM	3v3

Table 3.4: Connections to Interface a Seven-Segment Display with Vaman-Pygmy.

3.1.2.3. Building

3.1.2.1. Build the PlatformIO project at

```
inter−chip/esp32−fpga/sevenseg/codes/esp32
```

3.1.2.2. Flash the project .bin file using USB-UART connected to the Vaman-ESP32, via PlatformIO or ArduinoDroid.

3.1.2.3. Build the FPGA project .bin file by entering the following commands at a terminal window.

```
# The following variable can be sourced from .shellrc or .venv/bin/activate
```

```
export QORC_SDK_PATH=/path/to/pygmy-sdk
cd inter-chip/esp32-fpga/sevenseged/codes/fpga
make
```

3.1.2.4. On building successfully, the .bin file is generated at

```
inter-chip/esp32-fpga/sevenseg/codes/fpga/rtl/AL4S3B_FPGA_Top.bin
```

3.1.2.5. Flash the .bin file to the Vaman-Pygmy by resetting it to bootloader mode and entering the following command at a terminal window,

```
python3 /path/to/tinyfpga-programmer-gui.py --port /dev/ttyACMxx --mode
    fpga --appfpga /path/to/AL4S3B_FPGA_Top.bin --reset
```

where /dev/ttyACMxx is the port at which the Vaman board is available. This can be obtained by inspecting the output of the following command (requires root/sudo privileges).

```
dmesg -w
```

3.1.2.4. Demonstration

3.1.2.6. Find the IP address of the Vaman-ESP32 by inspecting the output of the serial terminal, or by typing at a terminal window

```
ifconfig
nmap -sn xx.yy.zz.0/24
```

where xx.yy.zz represents the first three octets of the IP address of your device on the WiFi network interface, found using `ifconfig`.

3.1.2.7. Then, go to the following website and interact with the HTML form to see the output on the seven-segment display.

```
http://<VAMAN−IP>/sevenseg
```

3.1.2.5. Exercises

3.1.2.8. Modify the ESP32 code to have a radio button for each segment. Optionally, add another radio button for the dot.

3.1.2.9. Try to minimize the number of SPI transactions and the amount of data transmitted in each of them.

3.2. FPGA and M4

This section describes various experiments that can be performed by interfacing the M4 and eFPGA susystems on the Vaman-Pygmy.

3.2.1. Onboard LED

This subsection illustrates the interface between the ARM Cortex-M4 subsystem and the eFPGA subsystem onboard the Vaman-Pygmy through a blink program, where the GPIO pins corresponding to the onboard LEDs are exposed by the FPGA platform.

3.2.1.1. Components

The components required are listed in Table 3.5.

Component	Value	Quantity
Vaman	LC	1
USB Cable	Type-B	1

Table 3.5: Components Required for Controlling the Onboard LED.

3.2.1.2. Building

3.2.1.1. Build the FPGA project .bin file by entering the following commands at a terminal window.

```
# The following variable can be sourced from .shellrc or .venv/bin/activate
export QORC_SDK_PATH=/path/to/pygmy-sdk
cd inter-chip/m4-fpga/blink/codes/fpga
make
```

3.2.1.2. On building successfully, the .bin file is generated at

```
inter-chip/m4-fpga/blink/codes/fpga/rtl/AL4S3B_FPGA_Top.bin
```

3.2.1.3. Edit the variable PROJ_ROOT in the following file to point to the location of the pygmy-sdk folder on your system.

```
inter-chip/m4-fpga/blink/codes/m4/GCC_Project/config.mk
```

3.2.1.4. Build the M4 project .bin file by entering the following commands at a terminal window.

```
cd inter-chip/m4-fpga/blink/codes/m4/GCC_Project
make -j4
```

3.2.1.5. On building successfully, the .bin file is generated at

```
inter−chip/m4−fpga/blink/codes/m4/GCC_Project/output/bin/m4.bin
```

3.2.1.6. Flash both FPGA and M4 .bin files to the Vaman-Pygmy by resetting it to bootloader mode and entering the following command at a terminal window.

```
python3 /path/to/tinyfpga−programmer−gui.py −−port /dev/ttyACMxx −−mode
    m4−fpga −−m4app /path/to/m4.bin −−appfpga /path/to/
    AL4S3B_FPGA_Top.bin −−reset
```

where /dev/ttyACMxx is the port at which the Vaman board is available. This can be obtained by inspecting the output of the following command (requires root/sudo privileges).

```
dmesg −w
```

3.2.1.3. Demonstration

All three LEDs onboard the Vaman will toggle every one second when the Vaman boots normally.

3.2.1.4. Exercises

3.2.1.7. Find out the addresses corresponding to the GPIO module by inspecting the code in pygmy-sdk, as well as in this folder.

3.2.1.8. Create code to blink each LED in turn.

3.2.2. Seven-Segment Display

This subsection demonstrates the interface between the M4 subsystem and eFPGA subsystem on the Vaman-Pygmy by controlling a seven-segment display, where the GPIO interface is exposed to the M4 by the FPGA platform.

3.2.2.1. Components

The components required are listed in Table 3.6.

Component	**Value**	**Quantity**
Vaman	LC	1
USB-UART		1
Jumper Wires	Female-to-Female	10
Jumper Wires	Male-to-Female	10
USB Cable	Type-B	1
Seven-Segment Display	Common Anode	1
Breadboard		1

Table 3.6: Components Required for Controlling the Onboard LED.

3.2.2.2. Connections

The connections between the seven-segment display and Vaman-Pygmy board are listed in Table 3.4.

3.2.2.3. Building

3.2.2.1. Build the FPGA project .bin file by entering the following commands at a terminal window.

```
# The following variable can be sourced from .shellrc or .venv/bin/activate
```

```
export QORC_SDK_PATH=/path/to/pygmy-sdk
cd inter-chip/m4-fpga/sevenseg/codes/fpga
make
```

3.2.2.2. On building successfully, the .bin file is generated at

```
inter-chip/m4-fpga/sevenseg/codes/fpga/rtl/AL4S3B_FPGA_Top.bin
```

3.2.2.3. Edit the variable PROJ_ROOT to point to the location of the pygmy-sdk folder on your system in the following file.

```
inter-chip/m4-fpga/sevenseg/codes/m4/GCC_Project/config.mk
```

3.2.2.4. Build the M4 project .bin file by entering the following commands at a terminal window.

```
cd inter-chip/m4-fpga/sevenseg/codes/m4/GCC_Project
make -j4
```

3.2.2.5. On building successfully, the .bin file is generated at

```
inter-chip/m4-fpga/sevenseg/codes/m4/GCC_Project/output/bin/m4.bin
```

3.2.2.6. Flash both FPGA and M4 .bin files to the Vaman-Pygmy by resetting it to bootloader mode and entering the following command at a terminal window.

```
python3 /path/to/tinyfpga-programmer-gui.py --port /dev/ttyACMxx --mode
      m4-fpga --m4app /path/to/m4.bin --appfpga /path/to/
     AL4S3B_FPGA_Top.bin --reset
```

where /dev/ttyACMxx is the port at which the Vaman board is available. This can be obtained by inspecting the output of the following command (requires root/sudo privileges).

```
dmesg −w
```

3.2.2.4. Demonstration

All three LEDs onboard the Vaman will toggle every one second when the Vaman boots normally.

3.2.2.5. Exercises

3.2.2.7. Create a decade counter with this interface.

3.2.2.8. Extend the code to accommodate hexadecimal digits.

3.2.3. Binary-Coded Decimal Decoder

This subsection demonstrates the interface between the M4 subsystem and eFPGA subsystem on the Vaman-Pygmy by simulating a Binary-Coded Decimal (BCD) Decoder, otherwise known as a 7447 IC. The GPIO interface is exposed to the M4 by the FPGA platform.

3.2.3.1. Components

The components required are listed in Table 3.7.

Component	Value	Quantity
Vaman	LC	1
USB-UART		1
Jumper Wires	Male-to-Female	20
USB Cable	Type-B	1
Seven-Segment Display	Common Anode	1
Breadboard		1

Table 3.7: Components Required for Simulating the 7447 IC.

3.2.3.2. Connections

The connections to be made between the seven-segment display and Vaman board are listed in Table 3.4. The equivalent 7447 IC input pins are listed in Table 3.8.

7447 IC	Vaman-Pygmy
A (MSB)	IO_10
B	IO_11
C	IO_12
D (LSB)	IO_13

Table 3.8: Equivalent Input Pins for the 7447 IC on the Vaman-Pygmy.

3.2.3.3. Building

3.2.3.1. Build the FPGA project .bin file by entering the following commands at a terminal window.

```
# The following variable can be sourced from .shellrc or .venv/bin/activate
export QORC_SDK_PATH=/path/to/pygmy-sdk
cd inter-chip/m4-fpga/bcd/codes/fpga
make
```

3.2.3.2. On building successfully, the .bin file is generated at

```
inter−chip/m4−fpga/bcd/codes/fpga/rtl/AL4S3B_FPGA_Top.bin
```

3.2.3.3. Edit the variable PROJ_ROOT to point to the location of the pygmy-sdk folder on your system in the file

```
inter−chip/m4−fpga/bcd/codes/m4/GCC_Project/config.mk
```

3.2.3.4. Build the M4 project .bin file by entering the following commands at a terminal window.

```
cd inter−chip/m4−fpga/bcd/codes/m4/GCC_Project
make −j4
```

3.2.3.5. On building successfully, the .bin file is generated at

```
inter−chip/m4−fpga/bcd/codes/m4/GCC_Project/output/bin/m4.bin
```

3.2.3.6. Flash both FPGA and M4 .bin files to the Vaman-Pygmy by resetting it to bootloader mode and entering the following command at a terminal window

```
python3 /path/to/tinyfpga−programmer−gui.py −−port /dev/ttyACMxx −−mode
      m4−fpga −−m4app /path/to/m4.bin −−appfpga /path/to/
     AL4S3B_FPGA_Top.bin −−reset
```

where /dev/ttyACMxx is the port at which the Vaman board is available. This can be obtained by inspecting the output of the following command (requires root/sudo privileges).

```
dmesg −w
```

3.2.3.4. Demonstration

Give an input to the BCD by connecting the male ends of the jumpers at pins 10 to 13 on the Vaman-Pygmy to high voltage or ground. The equivalent digit should be displayed on the seven-segment display.

3.2.3.5. Exercises

3.2.3.7. Create code for an incrementing encoder instead of a display decoder.

3.2.3.8. Extend the code to perform mathematical operations modulo 10 and display the result.

3.2.3.9. Extend the code to accommodate hexadecimal digits.

3.3. ESP32, FPGA and M4

This section describes various experiments that can be performed by interfacing all three platforms, ESP32, FPGA and M4 onboard the LC Vaman.

3.3.1. Onboard LED

This subsection illustrates the use of SPI Communication between the Vaman-ESP32 and the ARM Cortex-M4 subsystem onboard the Vaman-Pygmy by toggling the LEDs via the ARM interface, where the GPIO pins are exposed by the FPGA platform.

3.3.1.1. Components

The required components are listed in Table 3.9.

Component	Value	Quantity
Vaman	LC	1
USB-UART		1
Jumper Wires	Female-to-Female	20
USB Cable	Type-B	1

Table 3.9: Components Required for Controlling the Onboard LED via SPI.

3.3.1.2. Connections

The connections to be made on the Vaman are listed in Table 3.2.

3.3.1.3. Building

3.3.1.1. Build the PlatformIO project at

```
inter−chip/esp32−m4−fpga/led/codes/esp32
```

3.3.1.2. Flash the project .bin file using USB-UART connected to the Vaman-ESP32, via PlatformIO or ArduinoDroid.

3.3.1.3. Build the FPGA project .bin file by entering the following commands at a terminal window.

```
# The following variable can be sourced from .shellrc or .venv/bin/activate
export QORC_SDK_PATH=/path/to/pygmy−sdk
cd inter−chip/esp32−fpga/led/codes/fpga
make
```

3.3.1.4. On building successfully, the .bin file is generated at

```
inter−chip/esp32−fpga/led/codes/fpga/rtl/AL4S3B_FPGA_Top.bin
```

3.3.1.5. Edit the variable PROJ_ROOT to point to the location of the pygmy-sdk folder on your system in the file

```
inter−chip/esp32−m4−fpga/led/codes/m4/GCC_Project/config.mk
```

3.3.1.6. Build the M4 project .bin file by entering the following commands at a terminal window.

```
cd inter−chip/esp32−m4−fpga/led/codes/m4/GCC_Project
make −j4
```

3.3.1.7. Flash both FPGA and M4 .bin files to the Vaman-Pygmy by resetting it to bootloader mode and entering the following command at a terminal window,

```
python3 /path/to/tinyfpga−programmer−gui.py −−port /dev/ttyACMxx −−mode
      m4−fpga −−m4app /path/to/m4.bin −−appfpga /path/to/
     AL4S3B_FPGA_Top.bin −−reset
```

where /dev/ttyACMxx is the port at which the Vaman board is available. This can be obtained by inspecting the output of the following command (requires root/sudo privileges).

```
dmesg −w
```

3.3.1.4. Demonstration

3.3.1.8. Find the IP address of the Vaman-ESP32 by inspecting the output of the serial terminal, or by typing at a terminal window

```
ifconfig
```

```
nmap -sn xx.yy.zz.0/24
```

where xx.yy.zz represents the first three octets of the IP address of your device on the WiFi network interface, found using `ifconfig`.

3.3.1.9. Then, go to the following website and interact with the radio buttons to toggle the LEDs onboard the Vaman-Pygmy.

```
http://<VAMAN-IP>/led
```

3.3.1.5. Exercises

3.3.1.10. Modify the ESP32 code to have one radio button for each onboard LED.

3.3.1.11. Modify the ESP32 code to blink the LED periodically when the user toggles a checkbox on the HTML form.

3.3.1.12. Minimize the number of SPI transactions and the amount of data transmitted in each of them.

3.3.1.13. (Optional) Use CSS to style the bland HTML form.

3.3.1.14. (Optional) Use JavaScript to make the form reactive i.e., changes should be seen on toggling buttons or switches.

3.3.2. Seven-Segment Display

This subsection illustrates the use of SPI Communication between the Vaman-ESP32 and the ARM Cortex-M4 subsystem onboard the Vaman-Pygmy by controlling a seven-segment display via the ARM interface, where the GPIO pins are exposed by the FPGA platform.

3.3.2.1. Components

The components required are listed in Table 3.3.

3.3.2.2. Connections

The connections required on the Vaman board are listed in Table 3.2. The connections to interface the seven-segment display are listed in Table 3.4.

3.3.2.3. Building

3.3.2.1. Build the PlatformIO project at

```
inter−chip/esp32−m4−fpga/sevenseg/codes/esp32
```

3.3.2.2. Flash the project .bin file using USB-UART connected to the Vaman-ESP32, via PlatformIO or ArduinoDroid.

3.3.2.3. Build the FPGA project .bin file by entering the following commands at a terminal window.

```
# The following variable can be sourced from .shellrc or .venv/bin/activate
export QORC_SDK_PATH=/path/to/pygmy−sdk
cd inter−chip/esp32−fpga/sevenseg/codes/fpga
make
```

3.3.2.4. On building successfully, the .bin file is generated at

```
inter−chip/esp32−fpga/sevenseg/codes/fpga/rtl/AL4S3B_FPGA_Top.bin
```

3.3.2.5. Edit the variable PROJ_ROOT to point to the location of the pygmy-sdk folder on your system in the file

```
inter−chip/esp32−m4−fpga/sevenseg/codes/m4/GCC_Project/config.mk
```

3.3.2.6. Build the M4 project .bin file by entering the following commands at a terminal window.

```
cd inter−chip/esp32−m4−fpga/sevenseg/codes/m4/GCC_Project
make −j4
```

3.3.2.7. Flash both FPGA and M4 .bin files to the Vaman-Pygmy by resetting it to bootloader mode and entering the following command at a terminal window,

```
python3 /path/to/tinyfpga−programmer−gui.py −−port /dev/ttyACMxx −−mode
    m4−fpga −−m4app /path/to/m4.bin −−appfpga /path/to/
    AL4S3B_FPGA_Top.bin −−reset
```

where /dev/ttyACMxx is the port at which the Vaman board is available. This can be obtained by inspecting the output of the following command (requires root/sudo privileges).

```
dmesg −w
```

3.3.2.4. Demonstration

3.3.2.8. Find the IP address of the Vaman-ESP32 by inspecting the output of the serial terminal, or by typing at a terminal window

```
ifconfig
nmap −sn xx.yy.zz.0/24
```

where xx.yy.zz represents the first three octets of the IP address of your device on the WiFi network interface, found using `ifconfig`.

3.3.2.9. Then, go to the following website and interact with the HTML form to display the required digit on the seven-segment display.

```
http://<VAMAN_IP>/sevenseg
```

3.3.2.5. Exercises

3.3.2.10. Modify the ESP32 code to have one radion button for each segment. Optionally, add another radio button for the dot.

3.3.2.11. Minimize the number of SPI transactions and the amount of data transmitted in each of them.

3.3.2.12. (Optional) Use CSS to style the bland HTML form.

3.3.2.13. (Optional) Use JavaScript to make the form reactive i.e., changes should be seen on toggling buttons or switches, and not when the user clicks to submit the form.

3.3.3. Bluetooth-Controlled Seven-Segment Display

This subsection demonstrates how to control seven-segment display using EOS-S3 and ESP32 through SPI protocol. Here, ESP32 acts as master and EOS-S3 acts as slave. The values that are entered in Dabble Terminal are received by ESP32 through Bluetooth and the values are transferred to EOS-S3 through SPI. This is facilitated only when all 4 jumpers on the board are closed.

3.3.3.1. Components

Component		Quantity
Resistor	1k Ohm	6
Seven Segment Display		1
Vaman	LC	1
Arduino	UNO	1
Jumper Wires		10
Bread board		1

Table 3.11: Components

3.3.3.1. Now connect the Seven Segment to the Vaman board according to the given Table Table 3.13.

Pygmy	Seven Segment pins
IO-4	a
IO-5	b
IO-6	c
IO-7	d
IO-8	e
IO-10	f
IO-11	g
VCC	VCC
GND	GND

Table 3.13: Connections

3.3.3.2. Building and Flashing

3.3.3.2. Build the ESP32 firmware

```
cd inter−chip/esp32−m4−fpga/bluetooth−sevenseg/codes/spi_esp32
```

```
pio run
```

3.3.3.3. Flash ESP32 firmware (connect Arduino-UART)

```
pio run −t upload
```

3.3.3.4. Modify line 140 of config.mk to setup path to pygmy-sdk and then build m4 firmware using

```
cd spi_m4/GCC_Project
make
```

3.3.3.5. If using termux, send output/bin/spi_m4.bin to PC using

```
scp output/spi_m4.bin username@IPaddress:
```

3.3.3.6. Connect usb cable to vaman board and Flash eos s3 soc, using

```
sudo python3 <Type path to tiny fpga programmer application> −−port /dev/
    ttyACM0 −−appfpga top.bin −−m4app spi_m4.bin −−mode m4−fpga −−reset
```

3.3.3.7. Install the **Dabble app** on the Mobile from the **Playstore**. Connect it to the **ESP32** on the Vaman Board using **Bluetooth**. Change the controls to **Terminal** to control seven-segment display.

Chapter 4

UGV

This chapter describes the setup of an Unmanned Ground Vehicle (UGV) and a few experiments that can be performed on it with the Vaman board.

4.1. Components Table

The components of the UGV kit are listed in Table 4.1.

Components	**Quantity**	**References**
Vaman Board ESP32	1	Figure 1.2
Arduino UART	1	Figure 1.1
UGV Chasis	1	Figure 4.3
L293 Motor Driver	1	Figure 4.2
DC Motors	2	Figure 4.1
Batteries	4	Figure 4.4
Jumper wires	15	-
Bread Board	1	Figure 1.4

Table 4.1: components table of toycar

Figure 4.1: DC motors

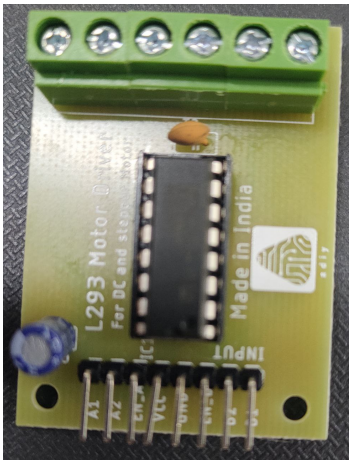


Figure 4.2: L293 motor driver

4.2. Assembling the UGV kit

Assemble the Chassis using the provided nuts/screws, Wheels, and parts.

Figure 4.3: UGV frame/chassis

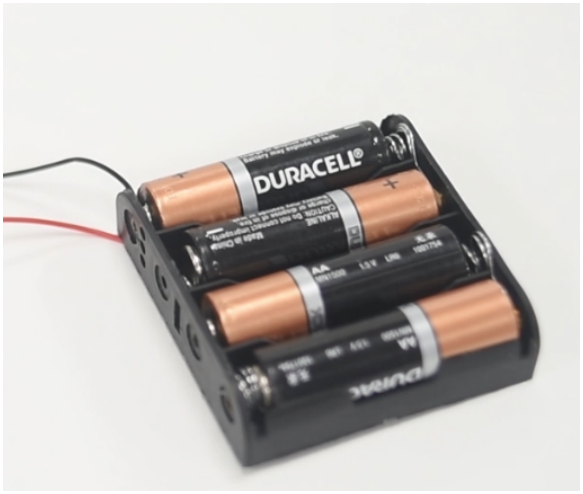


Figure 4.4: Batteries for powering various equipments

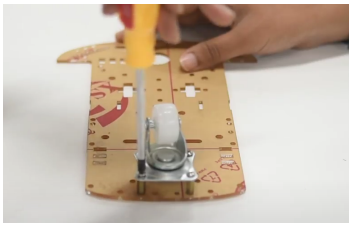


Figure 4.5: screws connecting

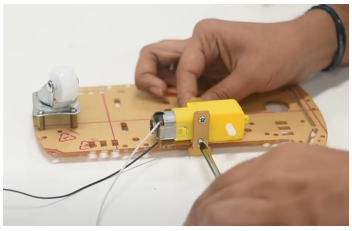


Figure 4.6: Dc motors connecting

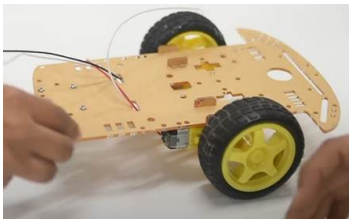


Figure 4.7: wheels connections

4.3. Circuit Connections

Motor Driver Unit	**DC Motor**
MA1	Right Motor Input 1
MA2	Right Motor Input 2
MB1	Left Motor Input 1
MB2	Left Motor Input 2
5v	VCC
GND	GND

Table 4.2: DC motor connection with L293 Motor Driver

4.3.1. Connect the Arduino-UART pins to the Vaman ESP32 pins according to Table 1.1.

4.3.2. For Bluetooth toycar connect the circuits connection as per Table 4.2 and Table 4.3.

4.3.3. For **Integrated Bluetooth toycar** connect the circuits connection as per Table 4.2 , Table 4.4 and Table 4.5.

Vaman Board ESP 32	Motor Driver Unit
Pin 16	Input A1
Pin 17	Input A2
Pin 18	Input B1
Pin 19	Input B2
5v	VCC
GND	GND

Table 4.3: vaman Connections

Vaman Board	Motor Driver Unit
Pin 21	Right Motor Input 1
Pin 18	Right Motor Input 2
Pin 23	Left Motor Input 1
Pin 22	Left Motor Input 2
5v	VCC
GND	GND

Table 4.4: connection with vaman board

INPUT	VAMAN BOARD	OUTPUT	MOTOR
A1	PYGMY 21	Vcc	5V
A2	PYGMY 18	GND	GND
EN	-	MA1	MOTOR A1
VCC	5V	MA2	MOTOR A2
B2	PYGMY 23	MB1	MOTOR B1
B1	PYGMY 22	MB2	MOTOR B2
5V	VCC	-	-
GND	GND	-	-

Table 4.5: vaman connection with L293 Motor Driver

4.4. Code Execution For Bluetooth Toycar

4.4.1. Now,Execute the following code

vaman/toycar/codes/bluetooth_toycar/src

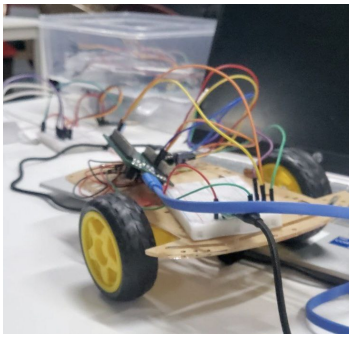


Figure 4.8: After all connections

4.4.2. Build the ESP32 firmware

```
cd vaman/toycar/codes/bluetooth_toycar
pio run
```

4.4.3. Flash ESP32 firmware (connect Arduino-UART)

```
pio run −t upload
```

4.4.4. Install the **Dabble app** on the Mobile from the **Playstore**. Connect it to the **ESP32** on the Vaman Board using **Bluetooth**. Change the controls to **Joystick mode** as shown in Figure 1.10to navigate the UGV.

4.5. Code Execution for Integrated Bluetooth Toycar

4.5.1. Now,Execute the following code

```
vaman/toycar/codes/bluetooth_toycar
```

4.5.2. Build the ESP32 firmware

```
cd esp32_pwmctrl
pio run
```

4.5.3. Flash ESP32 firmware (connect Arduino-UART)

```
pio run −t nobuild −t upload
```

4.5.4. If using termux, send .pio/build/esp32doit-devkit-v1/firmware.bin to PC using

```
scp .pio/build/esp32doit−devkit−v1/firmware.bin Username@IPAddress:
```

4.5.5. Modify line 140 of config.mk to setup path to pygmy-sdk and then Build m4 firmware using

```
cd m4_pwmctrl/GCC_Project
make
```

4.5.6. If using termux, send output/m4_pwmctrl.bin to PC using

```
scp output/m4_pwmctrl.bin username@IPaddress:
```

4.5.7. Build fpga source (.bin file)

```
cd fpga_pwmctrl/rtl
ql_symbiflow −compile −d ql−eos−s3 −P pu64 −v ∗.v −t AL4S3B_FPGA_Top −p
    quickfeather.pcf −dump jlink binary
```

4.5.8. If using termux, send AL4S3B_FPGA_Top.bin to PC using

```
scp AL4S3B_FPGA_Top.bin username@IPaddress:
```

4.5.9. Connect usb cable to vaman board and Flash eos s3 soc, using

```
sudo python3 <Type path to tiny fpga programmer application> −−port /dev/
    ttyACM0 −−appfpga AL4S3B_FPGA_Top.bin −−m4app m4_pwmctrl.bin −−
    mode m4−fpga −−reset
```

4.5.10. Install the **Dabble app** on the Mobile from the **Playstore**. Connect it to the **ESP32** on the Vaman Board using **Bluetooth**. Change the controls to **Joystick mode** as shown in Figure 1.10 to navigate the UGV.

4.5.1. Working

On the hardware level there are three key points: SPI,Wishbone Interfacing and Address Mapping. Vaman Board, we have an EOS S3 and ESP32. The Communication between these two happens via Serial Peripheral Interface(SPI).

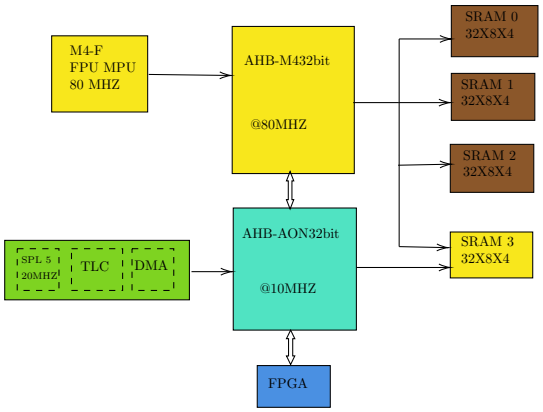


Figure 4.9: EOS S3 Architecture

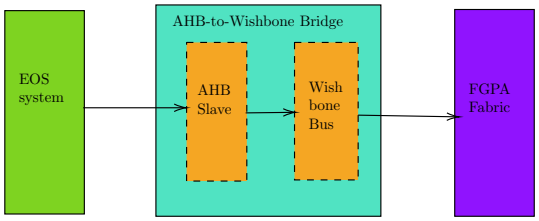


Figure 4.10: Wishbone Slave Interface

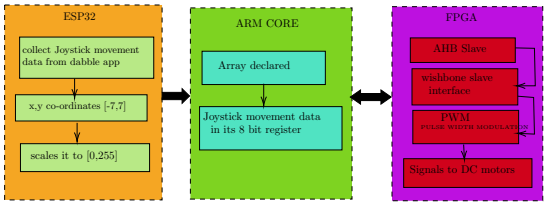


Figure 4.11: Hardware Block level Architecture

Chapter 5

Pulse Width Modulation

This chapter demonstrates various applications of Pulse Width Modulation (PWM) using the Vaman board.

5.1. Onboard LED

This section illustrates the use of SPI Communication between the Vaman-ESP32 and the Vaman-Pygmy in adjusting the brightness of an LED onboard the Vaman-Pygmy.

5.1.1. Components

The components required are listed in Table 5.1.

Component	Value	Quantity
Vaman	LC	1
Jumper Wires	Female-to-Female	20
USB-UART		1
USB Cable	Type-B	1

Table 5.1: Components Required for Controlling the Onboard LED via SPI.

5.1.2. Connections

The connections on the Vaman board are listed in Table 5.2.

Pin	Vaman-ESP32	Vaman-Pygmy
CS/SS	27	20
CIPO/MISO	19	17
COPI/MOSI	18	19
SCLK	5	16

Table 5.2: Connections to establish SPI between Vaman-ESP32 and Vaman-Pygmy.

5.1.3. Building

5.1.3.1. Build the PlatformIO project at

```
pwm/led/codes/esp32
```

5.1.3.2. Flash the project .bin file using USB-UART connected to the Vaman-ESP32, via PlatformIO or ArduinoDroid.

5.1.3.3. Build the FPGA project .bin file by entering the following commands at a terminal window.

```
# The following variable can be sourced from .shellrc or .venv/bin/activate
export QORC_SDK_PATH=/path/to/pygmy-sdk
cd pwm/led/codes/fpga
make
```

5.1.3.4. On building successfully, the file

```
pwm/led/codes/fpga/rtl/AL4S3B_FPGA_Top.bin
```

is generated.

5.1.3.5. Edit the variable PROJ_ROOT in

```
pwm/led/codes/m4/GCC_Project/config.mk
```

to point to the location of the pygmy-sdk folder on your system.

5.1.3.6. Build the M4 project .bin file by entering the following commands at a terminal window.

```
cd pwm/led/codes/m4/GCC_Project
make -j4
```

5.1.3.7. Flash both FPGA and M4 .bin files to the Vaman-Pygmy by resetting it to bootloader mode and entering the following command at a terminal window.

```
python3 /path/to/tinyfpga-programmer-gui.py --port /dev/ttyACMxx --mode
    m4-fpga --m4app /path/to/m4.bin --appfpga /path/to/
    AL4S3B_FPGA_Top.bin --reset
```

where /dev/ttyACMxx is the port at which the Vaman board is available. This can be obtained by inspecting the output of the following command (requires root/sudo privileges).

```
dmesg -w
```

5.1.4. Demonstration

5.1.4.8. Find the IP address of the Vaman-ESP32 by inspecting the output of the serial terminal, or by typing at a terminal window

```
ifconfig
nmap −sn xx.yy.zz.0/24
```

where xx.yy.zz represents the first three octets of the IP address of your device on the WiFi network interface, found using `ifconfig`.

5.1.4.9. Then, go to the site

```
http://<VAMAN−IP>/pwm
```

and enter the PWM value, which is an integer between 0 and 255.

5.1.4.10. On entering the value, the brightness of the green LED will change according to the PWM value.

5.1.5. Exercises

5.1.5.11. Add code to gradually increase or decrease brightness of the onboard LED.

5.1.5.12. Avoid using the M4 platform entirely by having the ESP32 write to the appropriate memory address of the PWM using SPI directly.

5.2. UGV

This section illustrates the use of SPI Communication between the Vaman-ESP32 and the Vaman-Pygmy in adjusting the speed of the UGV.

5.2.1. Components

The components required are listed in Table 5.3.

Component	Value	Quantity
Vaman	LC	1
USB-UART		1
Jumper Wires	Male-to-Female	20
USB Cable	Type-B	1
Breadboard		1
Battery	12 V	1
UGV Kit		1

Table 5.3: Components Required for Controlling the UGV PWM via SPI.

5.2.2. Connections

L293 IC	Vaman-Pygmy	12 V Battery
MA1	IO_18	
MA2		
MB1	IO_22	
MB2		
ENA	3v3	
ENB		
VCC		12 V
GND		GND

Table 5.4: Connections to establish SPI between Vaman-ESP32 and Vaman-Pygmy.

5.2.3. Building

5.2.3.1. Build the PlatformIO project at

```
pwm/ugv/codes/esp32
```

5.2.3.2. Flash the project .bin file using USB-UART connected to the Vaman-ESP32, via PlatformIO or ArduinoDroid.

5.2.3.3. Build the FPGA project .bin file by entering the following commands at a terminal window.

```
# The following variable can be sourced from .shellrc or .venv/bin/activate
export QORC_SDK_PATH=/path/to/pygmy-sdk
cd pwm/ugv/codes/fpga
make
```

5.2.3.4. On building successfully, the file

```
pwm/ugv/codes/fpga/rtl/AL4S3B_FPGA_Top.bin
```

is generated.

5.2.3.5. Edit the variable PROJ_ROOT in

```
pwm/ugv/codes/m4/GCC_Project/config.mk
```

to point to the location of the pygmy-sdk folder on your system.

5.2.3.6. Build the M4 project .bin file by entering the following commands at a terminal window.

```
cd pwm/ugv/codes/m4/GCC_Project
make -j4
```

5.2.3.7. Flash both FPGA and M4 .bin files to the Vaman-Pygmy by resetting it to bootloader mode and entering the following command at a terminal window.

```
python3 /path/to/tinyfpga−programmer−gui.py −−port /dev/ttyACMxx −−mode
    m4−fpga −−m4app /path/to/m4.bin −−appfpga /path/to/
    AL4S3B_FPGA_Top.bin −−reset
```

where /dev/ttyACMxx is the port at which the Vaman board is available. This can be obtained by inspecting the output of the following command (requires root/sudo privileges).

```
dmesg −w
```

5.2.4. Demonstration

5.2.4.8. Find the IP address of the Vaman-ESP32 by inspecting the output of the serial terminal, or by typing at a terminal window

```
ifconfig
nmap −sn xx.yy.zz.0/24
```

where xx.yy.zz represents the first three octets of the IP address of your device on the WiFi network interface, found using `ifconfig`.

5.2.4.9. Then, go to the site

```
http://<VAMAN−IP>/pwm
```

and enter the PWM value, which is an integer between 0 and 255.

5.2.4.10. On entering the value, the speed of the UGV will change according to the PWM value.

5.2.5. Exercises

5.2.5.11. Introduce a slow accelerating motion in the UGV by stepping up the PWM. Take user inputs for starting PWM and step size. Additionally, you can also let the user control whether the PWM will step-up (accelerate) or step-down (decelerate).

5.2.5.12. Modify the Verilog code to have four separate PWMs available, and connect the motor pins to each PWM. Also, rewrite the M4 code for facilitating the use of these four PWMs.

5.2.5.13. Avoid using the M4 platform entirely, by writing directly to the correct memory locations from the Vaman-ESP32 using SPI. This will improve the latency in the autonomous vehicle system.