Design of Battery Charger with IoT Capabilities for Electric Bikes

Kevin Naoko[1], Dhanurangga Al Fadh, Danu Ihza Pamungkas, Muhammad Amin Sulthoni, Arwindra Rizqiawan

School of Electrical Engineering and Informatics, Bandung Institute of Technology

Bandung, 40135, West Java, Indonesia

[1]kevin.naoko11@gmail.com

***Abstract*— This paper describes the design process of a microcontroller-based portable battery charger that has an Internet of Things (IoT) capability to monitor the charging process. A standard DC-DC buck converter is used as power converter to give a DC charging current to a battery, which is controlled by ESP32. At various points of operation, the microcontroller can also send various data to a remote server for monitoring purposes via the Global System for Mobile Communications (GSM) network. It can also send its current location to the server. A rudimentary web dashboard is also made to monitor the charger's state of operation at any given time. The designed system is able to utilize a single microprocessor to simultaneously control the charging process, and also to communicate with the remote server. The charger itself has a maximum output of 650 watts (65.5V, 10A). Experimental works have been done to verify the proposed design**

Keywords*— *battery charger, buck converter, ESP32, IoT monitoring

I. INTRODUCTION

Indonesia's environmental and energy policies have steered a transformation in the domestic automotive industry, speeding up the transition from oil to renewable energy. A few stakeholders are trying to help ease this transition. Some of them provide an e-bike rental service, complete with its charging infrastructure. Some of the e-bikes have a removable battery. These kinds e-bikes have a faster "fueling" time compared to its non-removable counterparts, as a near-empty battery can be quickly swapped with a fully-charged pack in one of the many battery swapping stations that are placed in public places. However, not everyone is able to access the swapping station with ease, and some would rather to charge the e-bikes in the comfort of their own homes. For that reason, a portable battery charger with a suitable output is needed.

Currently, there are many off-the-shelf battery chargers with an output that are suitable for the battery packs. However, there are some communications protocol that need to be done in order to enable the battery's power flow. The default setting is usually off in most cases, to prevent unwanted short circuits that can happen when handling the battery pack outside of the e-bike. Consequently, such generic chargers can't be used to charge proprietary battery packs. The portable chargers are also rented to the patrons as part of the charging infrastructure. Therefore, there is also a need to monitor the portable charger remotely at any given time.

The rated voltage of an e-bike battery pack are not the same for every type of e-bike. There are a lot of factors to be considered when choosing a certain voltage for a battery pack, such as the power needed by the electric motor, cable gauge, and component availability [1]. Some use a 48V battery [2][3], or even a 100V battery [1]. In this paper, we will use a 11.2Ah battery pack with a rated voltage of 60.2V. To keep costs low, it is preferred that one portable charger can be used to charge batteries with different nominal voltages. With a digital-based charger, changing the voltage or current output can be done with relative ease, by changing some of the parameters in the designed firmware. Many have used an Arduino as the digital control for their designed buck converters [4][5][6]. However, one Arduino is not powerful enough to run the required tasks, that is the charging control, the IoT control, and a few other controls. Therefore, ESP32 is chosen as a more powerful alternative than the Arduino microprocessor.

There are a few options of power converters that can be used. Two of the most popular power converters are boost converters and buck converters. For a stationary battery charger that receives power from the mains, a buck converter is preferred. We will use a standard buck converter topology to reduce cost, size, and increase efficiency [7]. To control the buck converter's output, a minorly-modified P controller is implemented in the ESP32, with the feedback parameters being the current and voltage output. The advised charging current for a battery pack is about 0.5C to 1.0C [8]. For our battery pack, that translates to 10A of maximum charging current, or around 600W of power. There is also an option to reduce the charging current to 5A, in case the portable charger is used in an area with limited power.

The IoT functionality enables the charger to send and receive data from a server via the GSM network. In general, the functionality is intended to monitor the system as it's being used to charge a battery, and also to monitor the system's current location. To prevent misuse of the charger, it can also be disabled remotely. The system and the server communicates via the Message Queuing Telemetry Transport (MQTT) protocol with a Quality of Service (QoS) level of 2, ensuring data integrity that is sent and received by both parties, i.e. the system and server [9].

In this paper, we will show the design and implementation process in making a portable battery charger with a rated output that is suitable to charge our battery pack, including the buck converter circuit, buck converter control and the IoT control. The system's voltage input is 220V AC. The charger's output voltage is be able to adjust according to the battery's State of Charge (SoC). In the case of charging our battery pack, it can vary between 50 to 67.5V. The charger's maximum current output is around 10A. When the battery is nearly full, the current will decrease until the battery reaches 100%. To prevent any fire hazard, a simple overheat cutoff protection needs to be implemented in the designed firmware.

There are also a few functionality that are also implemented but not explained in this paper, such as the communications process that can enable the battery pack's power flow, as well as the human interface processing.

II. System Overview

A. System Architecture

As seen in Figure 1, one ESP32 microcontroller is responsible for handling the charger control and the IoT control. The designed system has two battery slots to simultaneously charge two batteries at the same time, if need be. We will further discuss about the charger subsystem, which includes the buck converter and its controller, as well as the IoT subsystem.

B. Charger Subsystem

The charger subsystem is composed of a rectifier, a standard buck converter, and a gate driver circuit based on the TLP250 chip. To control the buck converter, a PWM signal with a frequency of 23kHz is given from the ESP32. The duty cycle given is based on the voltage and current readings provided by the voltage and current sense. The components used in the designed buck converter were selected based on the maximum current and voltage output that is needed to charge the battery.

With a switching frequency of 23kHz, a 661µH inductor is needed for the buck converter in order to operate in Continuous Current Mode (CCM). In order to avoid the inductor to enter saturation mode, we need to calculate the magnetomotive force (mmf) absorbed by the inductor at maximum rating (10A). From the B-H curve provided by the datasheet, we can see whether the inductor can still operate in linear mode or not. However, finding an inductor with a detailed datasheet in the domestic market is proven to be quite hard, so the inductor's theoretical saturation current cannot be calculated. To solve this issue, an inductor core with a very low μ_R (10) is selected in hopes that the inductor's saturation current is high enough to operate in 10A.

To reduce the current and voltage spike that occur in every switching cycle, there is a need for a voltage snubber. The snubber circuit is placed in parallel with the switching device *Q*. A few combinations of R and C values are tested to find an optimum value to reduce the voltage spikes. After selecting a resistor value of 31Ω and a capacitor value of 100nF, the voltage spikes are reduced significantly.

There are two phases that needs to be done when charging a Li-Ion battery, that is constant current phase (CC), and constant voltage phase (CV) [10]. When a battery's SoC is low, the charging phase is CC, where the battery's voltage will increase as the SoC increases. The charging phase will change into CV when the nominal voltage of the battery is near its fully-charged voltage. For our battery pack, the voltage is 65.5V. In this phase, the output voltage is held constant, which causes the current flow to slowly decrease until the battery is fully charged. For the constant current phase, a proportional controller with some minor modification is used to control the output current of the buck converter. The current sense block will provide feedback information to the designed controller. Proportional controllers tend to have a steady state error which can cause the steady state current flow to be different than the setpoint current. We implemented an extra comparison algorithm to the P controller so that the error can be minimized.

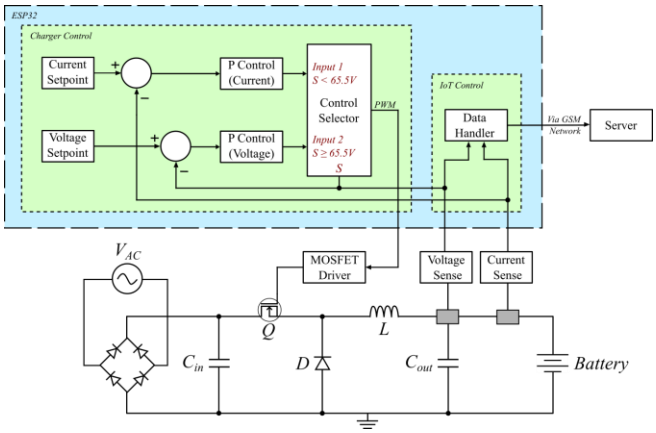


Figure 1. System architecture

The voltage sense block will detect the nominal voltage of the battery. When the voltage is at 65.5V, the P control no longer tries to compensate for the steady state error. Rather, it will keep the duty cycle value constant. This causes the current output to drop slowly as the battery's voltage increases. This goes on until the current output is small enough, and the charging is complete. In other words, the maximum output power will be at 655 watts.

For the case where the charger is used in an area with limited power available, two charging modes are available in the charger, which are slow charging or fast charging. Slow charging limits the output current to 5A, which effectively halves the power usage of the charger. Fast charging allows the charger to have an output current of 10A.

There is also a temperature sensor that is placed near the switching device *Q* to monitor its temperature when the buck converter is charging a battery. A simple failsafe can be implemented which stops the charging operation if the temperature has risen to above the safety threshold.

C. IoT Subsystem

This subsystem also encompasses some things outside the portable charger, such as the remote server. Going forward, we will refer the designed system as *client*, and the remote server as *server*. The client sends a data packet only if a certain event is triggered, such as:

1. A battery pack is inserted to one of the two slots
2. A battery pack is finished charging
3. A battery pack is removed from the slot
4. The charging mode switch is changed

The hardware components that is used on the client is a GSM module (SIM800L) that can connect to the 2G cell network. Its primary purpose is to end data payloads to a server via MQTT protocol. In order to use the MQTT protocol, the system as a whole (client and server) needs to publish and subscribe to certain topics in order to communicate properly between each other. Seen in Figure 2, the client and server are publishing and subscribing to various topics, each with different payloads. In MQTT, there is a *wildcard* option to subscribe to a parent topic. Detailed description for each topic's payloads can be seen in Table 1

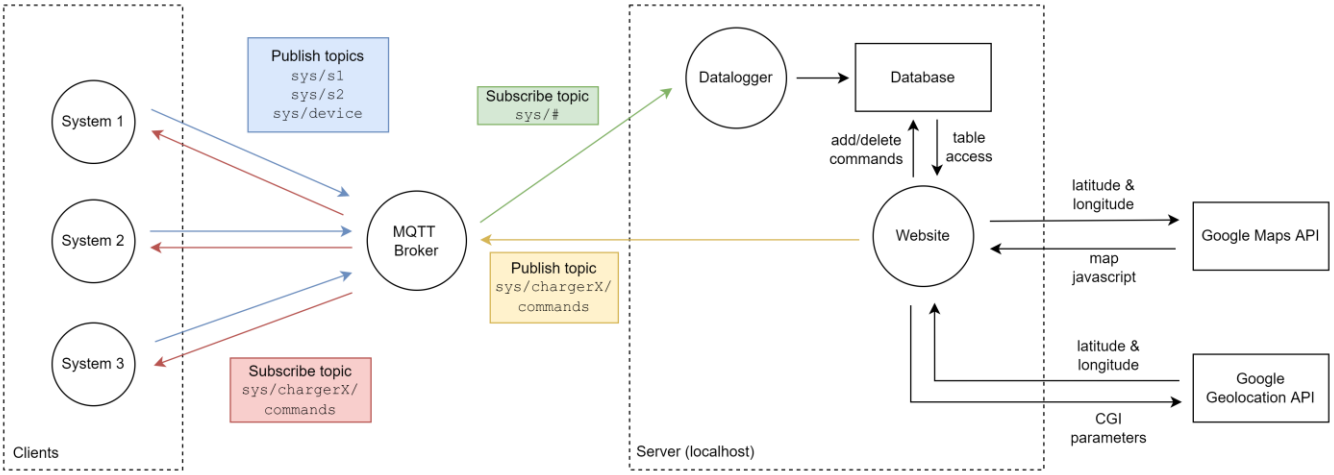


Figure 2 Data structure of the IoT Subsystem

1) Client-side Processing

At the client side, this subsystem's primary task is to publish and subscribe to a few topics. In order to do this, the GSM module needs a SIM card with a valid data plan, which enables it to connect to the MQTT broker. We've chosen Mosquitto as our MQTT broker. This subsystem also needs to be able to get its current location, and the current time. This can be done by sending a few serial commands to the GSM module [11]. As there is no urgency in pinpointing the exact location of the client, our system uses a tracking method based on readily-available cell phone network towers. By sending a series of serial commands to the GSM module, we can get 4 critical parameters, also known as Cell Global Identity (CGI) parameters. Said parameters are MCC, MNC, LAC, and Cell ID. From these parameters, the server then can get the latitude and longitude coordinates of the client. The process to these coordinates will be elaborated further in the document

In general, there are 3 main parts of the server, the MQTT listener, the database, and the webserver. The server needs a MQTT listener that's subscribed to every topic that are published by each client. This is achieved by making a python script that's connected to the MQTT broker

From Table 1, we can see that there are 3 topics which the clients are publishing to, each with different payload types. An SQLite database is used, containing 3 tables to store each topic's payloads, and a lookup table containing each client's current status. This extra table is used to enable/disable each client. SQLite is chosen for the database framework due to its portability.

The webserver is also made with Python. The back-end is made using Flask, a web framework for Python. The front-end is made with a combination of HTML, JS, and Jinja2. The backend processes primarily focus on accessing and editing the database, as well as processing the CGI parameters to get the latitude and longitude. Database access is done simply enough by querying the SQLite database to do various CRUD (create, read, update, or delete) actions based on user inputs at the frontend.

The frontend processes focus on displaying various information to the user. In general, the webserver is made to monitor each client's status, location, and usage history. In addition, it's also made to control (enable/disable) each client, and to monitor every battery pack that has been connected to our network of clients. There's also a user logon system, created to keep the database secure. To make things brief, only some of the webpage will be shown and explained, that is the home page, and the page to see usage history of one of the client

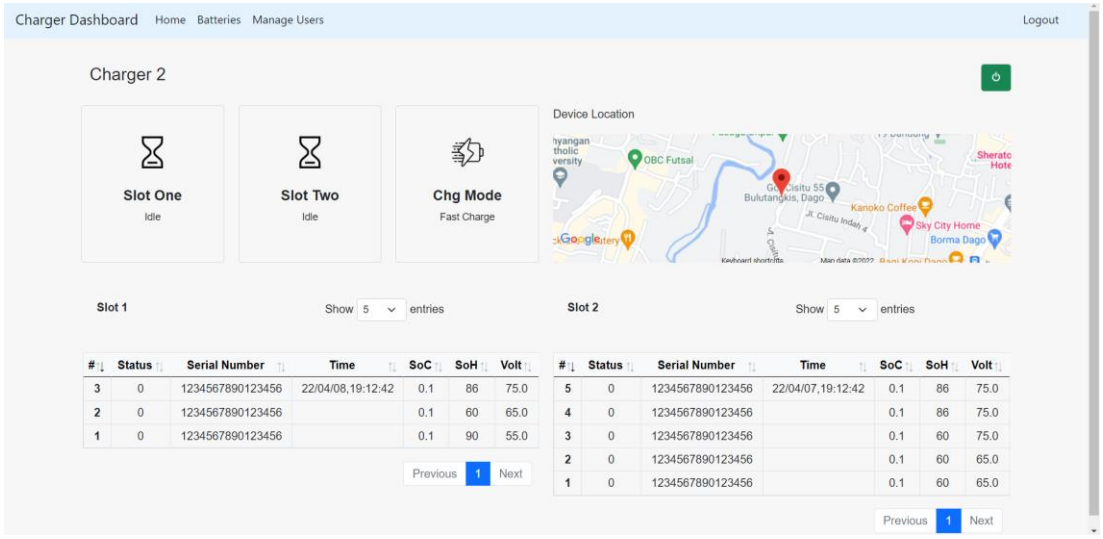


Figure 3 Information and history of a specific client (client 2)

Table 1 MQTT Topic payloads

Num	Topic	Payload contents
1	sys/s1	System ID, status, serial number, time, SoC, SoH, voltage from slot 1 or slot 2.
2	sys/s2	
3	sys/device	System ID, charging mode, and product location
4	sys/chargerX/ commands	Command to enable or disable the client

`After successfully logging in, users are then redirected to the homepage. The homepage itself contains the latest status of each client in the network, such as the latest status of both slots, the latest update time, and its charging mode. From the homepage, users can choose to see a complete history of a corresponding client, as can be seen in Figure 3. In it, an information regarding the latest status of each slot and client's current charging mode is displayed on the top left-hand side of the dashboard. On the top right side, there's a map showing the location of the client. And on the bottom side, there's the full usage history of each slot.

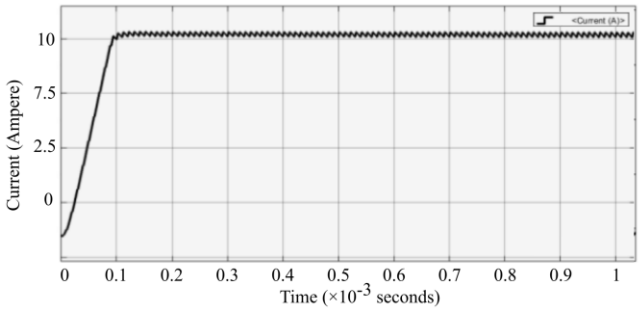
III. SYSTEM IMPLEMENTATION AND RESULT

In this section, we will discuss the development results of both the charging and the IoT subsystem. After that, a verification of each specification that is listed in the Introduction section will be conducted.

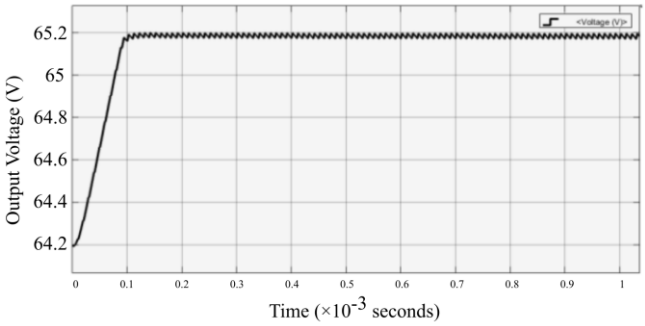
A. Charger Subsystem

Before proceeding with making a physical buck converter circuit, a simulation is done to see whether the designed charger can work properly with the component values that are selected, as well as the designed controller. The simulation is done using MATLAB Simulink. Seen from Figure 4, the constant current phase has been simulated to charge a battery pack with 10A of current. We can also see that the response time of the designed controller is quick enough for our use case.

From Figure 5, the transition from constant current phase to the constant voltage phase can be seen. The simulation starts at 80% SoC level. At nearly 85% SoC, the charging current drops steadily from 10A all the way to 0A, and the voltage level is kept constant. This proves that the designed charger and control could charge the battery with a CC-CV method.

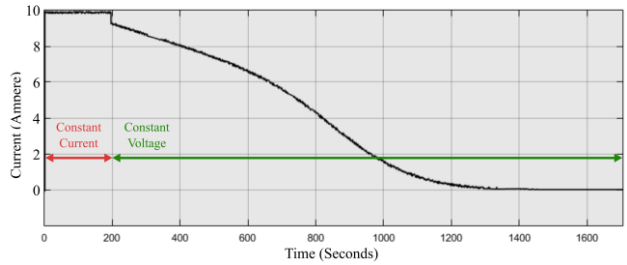


(a)

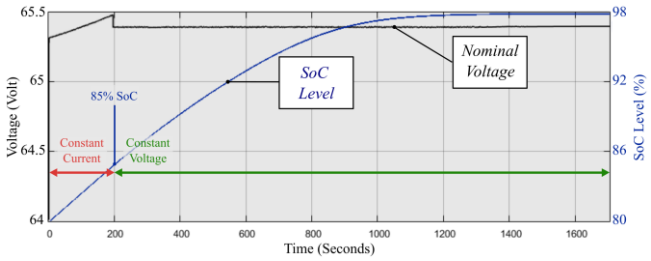


(b)

Figure 4 Charging simulation of the CC Phase (a) current output, (b) voltage output



(a)



(b)

Figure 5 Charging Simulation of the CC-CV Phase Transition (a) current output, (b) voltage and SoC level

After implementing the circuit in a PCB, a simple test is done to see the effectiveness of the designed charger. Figure 6 shows I_D waveform when the circuit is set to give an output of 10A. When the output is 10A, the current flow in I_D is only at 15.2A, way below the MOSFET's maximum rated current (40A). Thermal conditions also showed great results, with the hottest component being the diode with around 50°C when operating at 10A. With that, the buck converter could be tested thoroughly to charge a battery pack.

1) Overheat Protection

Overheat protection testing is done by heating the temperature sensor until the temperature reaches 70°C. At a temperature of 70.3°C, the temperature sensor detects overheating, and the charging process stops. After the temperature returns to normal, the charging process starts back again. This shows that the overheating protection has been running properly.

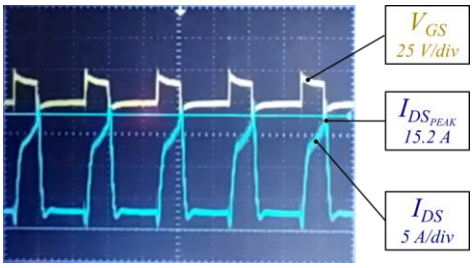


Figure 6 Waveform of V_{GS} (top) and I_{DS} (bottom) when I_O is 10A

2) Rated Output Voltage

This specification is intended to make sure that the system can charge a battery pack to its full capacity. A voltage measurement of the system's output voltage is done as it is currently charging a battery pack. Figure 7 shows that the system's output voltage is rising as the time goes. The charging mode used for this verification is slow charging (5A) From this observation, it is concluded that the output voltage is able to adjust according to the battery's SoC so the charging process can be done.

3) Rated output current and power

The system's current output is measured for both charging modes to verify this specification. The current measurement results. For slow charging, the output current is near 5A, with a tolerance range of about 80mA. For fast charging, the output is a little bit above 10A, with a tolerance range of about 26mA. We also tested the charging duration for both charging modes, and it is found that the battery needs around 94 minutes to fully charge in the slow charging mode, and around 53 minutes in the fast charging mode.

However, it should be noted that the initial SOC for the verification is not 0%. If a mathematical interpolation is done, the time needed to charge the battery in slow charging mode is 122 minutes, and in fast charging is 59 minutes. These figures are very close to our initial specifications, with only 2 minutes in difference. From this observation, it is concluded that the charger can charge the battery pack successfully with a reasonable amount of time.

B. IoT Subsystem

For this subsystem, throughout testing is done after the other subsystems are operational, mainly the charger and sensing subsystem. In general, the test conducted is to operate the system as normal. A series of events was done to trigger the system to send data to the server, such as the insertion or removal of a battery pack, or a change in the charging mode. Details about the test can be seen below.

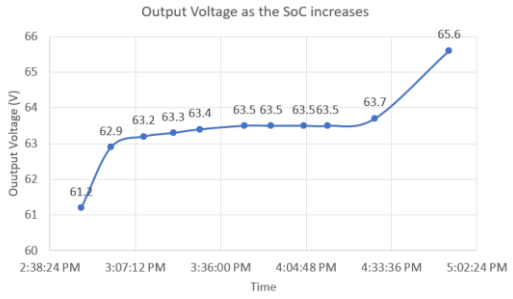


Figure 7 System's output voltage plotted as the battery's SoC increases

In general, there will be 2 sets of pictures, the left side represents the client's current state, and left side represents the dashboard view of the corresponding client. Some dashboard views are zoomed-in to convey a particular test result. To see the full view of the dashboard below, refer to Figure 3

In Figure 8 (a), both slots in the system are in the idle state, shown by the red LED, as there's no battery connected to any of the slot. As such, the database seen in Figure 8 (b) also shows 'Idle' at both of the status. There's also a location pin that can be seen on the right side of the dashboard. On the top right-hand corner, there's also a button to enable/disable the system. Refer to Figure 12 to see this functionality being tested. In Figure 9 (a), the left slot has entered the charging state, shown by the blue LED. The dashboard in Figure 9 (b) in turn also shows that slot one (left slot) is in the charging state. In Figure 10 (a), the charging mode switch is flicked to change the charging mode from fast charging, to slow charging. The dashboard in Figure 10 (b) also reflects that change, seeing that the charging mode is now slow charging.

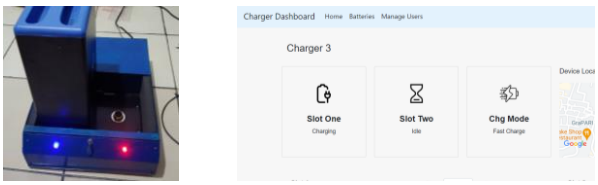
After the battery is fully charged, the system enters the finished charging state (green LED), seen in Figure 11 (a). The dashboard in Figure 11 (b) also shows that slot one is fully charged. If the user chooses to unplug the battery, the system will change its state to idle, akin to pictures in Figure 8. Next, Figure 12 (a) shows that the system is disabled via the remote server, indicated from the blinking red LED lights. The button in Figure 12 (b) also changed colors, with a description saying the system is disabled. To re-enable the system, users can click the button again, and the system will revert back to the idle state, as seen in Figure 8

C. System Integration

After verifying the functionality for each of the 6 subsystems, said subsystems are now ready for integration. A steel-framed case is chosen as the system's housing to provide extra ruggedness to the system, seeing the fact that it needs to support two 10 kg battery packs. Figure 13 and Figure 14 shows the system's prototype which has been inserted into the steel case.

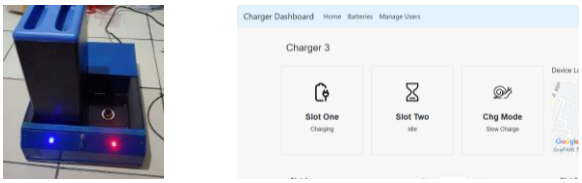
(a) **(b)**

Figure 8 Initial client startup
(a) system state. (b) dashboard view



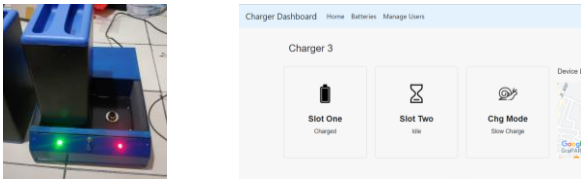
(a) **(b)**

Figure 9 Battery inserted to left slot.
(a) system state. (b) zoomed-in view of the dashboard



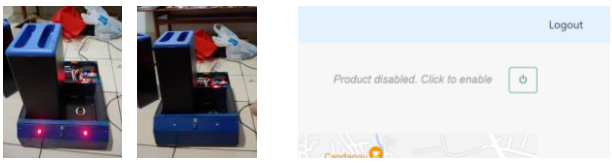
(a) **(b)**

Figure 10 Charging mode changed to slow charging. (a) system state. (b) zoomed-in view of the dashboard



(a) **(b)**

Figure 11 Battery is fully charged. (a) system state. (b) zoomed-in view of the dashboard



(a) **(b)**

Figure 12 System is disabled. (a) system state (Red LED blinking). (b) zoomed-in view of the disable button on the dashboard

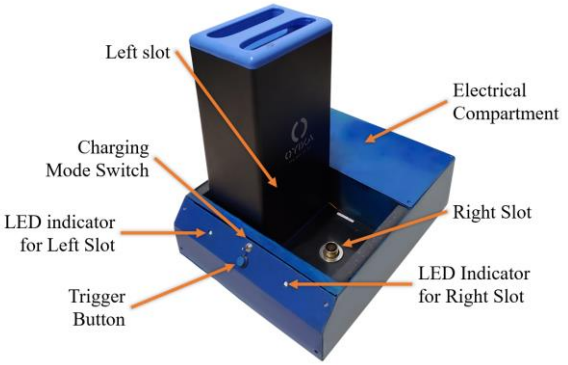


Figure 13 Prototype of the designed system (front side)

Figure 14 Prototype of the designed system (rear side)

IV. CONCLUSION

A portable battery charger with IoT capabilities has been designed successfully. A digital microcontroller-based (ESP32) DC-DC buck converter has been designed to charge our selected battery pack (60.2V, 11.2Ah), and a remote monitoring system has also been made to monitor the charger's current location and usage. The designed system is able to charge one battery with a current of 10A, or charge two batteries simultaneously, with each battery receiving 5A of current. The maximum output power of the designed system is around 650W. Users can change the charging mode to 5A if the so decide.

Besides that, the data acquisition and data logging also give appropriate results. The system could send various data, as well as being controlled from the remote server. Thanks to the usage of a microcontroller for the charging control, the output voltage and current can be changed by changing some parameters in the designed firmware, giving flexibility if the design is to be used to charge a battery with a different rated voltage than our initial battery pack.

V. ACKNOWLEDGEMENT

The authors would like to thank PT. Oyika Powered Solutions and Putra Agung Rekayasa for supporting the required funding and equipments needed for this research

REFERENCES

[1] M. N. Yuniarto, I. Sidharta, S. E. Wiratno, Y. U. Nugraha and D. A. Asfani, "Indonesian Electric Motorcycle Development: Lessons from innovation-based concept implementation on the design and production of the first Indonesian electric motorcycle," in IEEE Electrification Magazine, vol. 10, no. 1, pp. 65-74, March 2022, doi: 10.1109/MELE.2021.3139247.

[2] E. Drummond et al., "Design and Construction of an Electric Motorcycle," 2019 Systems and Information Engineering Design Symposium (SIEDS), 2019, pp. 1-6, doi: 10.1109/SIEDS.2019.8735634.

[3] S. S. Thakur, E. Ankit Roy, S. K. Dhakad and E. Alpesh jain, "Design of Electric Motorcycle," 2020 IEEE International Students' Conference on Electrical,Electronics and Computer Science (SCEECS), 2020, pp. 1-4, doi: 10.1109/SCEECS48394.2020.116.

[4] J. M. Kharade, A. A. Patil, N. V. Yadav, B. D. Kamble and A. B. Virbhadre, "Dual Battery Charger System for Electric Vehicle," 2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC), 2021, pp. 157-161, doi: 10.1109/ICESC51422.2021.9532862.

[5] T. Kaur, J. Gambhir and S. Kumar, "Arduino based solar powered battery charging system for rural SHS," 2016 7th India International Conference on Power Electronics (IICPE), 2016, pp. 1-5, doi: 10.1109/IICPE.2016.8079373.

[6] K. E. Hammoumi, R. E. Bachtiri, M. Boussetta and M. Khanfara, "Arduino Based Platform for Managing a PV Battery Charge," 2019 7th International Renewable and Sustainable Energy Conference (IRSEC), 2019, pp. 1-4, doi: 10.1109/IRSEC48032.2019.9078303.

[7] Ron Stull. "Isolated vs Non-Isolated Power Converters." (2019). Accessed: Oct. 29 2021. [Online]. Available: https://www.cui.com/blog/isolated-vs-non-isolated-power-converters

[8] C. Tsai, et al., "Designing a Fast Battery Charger for Electric Bikes," presented at the International Conference on System Science and Engineering, Taipei, Taiwan, July. 1-3

[9] A. Banks and R. Gupta. 'OASIS standard'. Accessed on 26 May 2022. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718099

[10] B. Arabsalmanabadi, N. Tashakor, A. Javadi and K. Al-Haddad, "Charging Techniques in Lithium-Ion Battery Charger: Review and New Solution," IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society, 2018, pp. 5731-5738, doi: 10.1109/IECON.2018.8591173.

[11] B. Mishra and A. Kertesz, 'The Use of MQTT in M2M and IoT Systems: A Survey', in IEEE Access, vol. 8, pp. 201071-201086, 2020, doi: 10.1109/ACCESS.2020.3035849.