Live Cryptocurrency Price Display

Jonathan Chesney (Student)
Department of Electrical and Computer Engineering
University of California, Davis
Davis, USA
jcchesney@ucdavis.edu

***Abstract*—This project is a live cryptocurrency price display. It is configured to automatically update every minute, where it will download the latest 20 minutes of a certain cryptocurrency's price data. It will draw a line graph on a 128x128 OLED display, headed by the cryptocurrency's symbol and price in US Dollars. Users may push a button to change which cryptocurrency is displayed. Another button will queue an email to be sent with a recent price of the currently displayed cryptocurrency, via AWS.**

I. Introduction

This project was inspired by the recent price surge of various cryptocurrencies. They have gained a significant amount attention in recent months and have become popular. The primary motivation of this project is the popularity of cryptocurrency discussion on the internet. This project sought to make an IoT device dedicated to monitoring the price of various cryptocurrencies. It achieved this by integrating a CC3200 Launchpad with Wi-Fi capability, a 128x128 OLED display, and Amazon Web Services. The device is meant to be like a clock: it is constantly displaying current information. Rather than perpetually displaying the time, the device perpetually displays cryptocurrency data. The displayed data can be changed via user input. In addition, user input can ultimately send an email with the most-recently downloaded price.

II. Methodologies

A. Components Used

- CC3200 Launchpad
- Adafruit 128x128 OLED Display (SSD1351)
- Amazon Web Services
    - IoT Core
    - Lambda
    - Simple Notification Service
- Crypto Compare Min API (www.cryptocompare.com)
- MikroElektronika GLCD Font Creator
- REST API

B. Approach

The first step was to set up the AWS Device Shadows. In total, there are three shadows: *BTC_Shadow*, *ETH_Shadow*, and *ADA_Shadow*. Each shadow stores the 20 floating point values that correlate to the 20 most recent prices of each cryptocurrency. It was necessary for them to be self-updating. This was achieved via 3 respective lambda functions, each with an EventBridge trigger enabling them to run every minute. Each time a lambda function ran, it deleted the least recent price and added the newest price to the shadow. It was necessary to use boto3 to update the shadow. IAM permissions needed to be granted for a successful UpdateThingShadow(). In addition to the lambda functions, policies had to be attached to each Shadow, in order to allow the CC3200 Launchpad to call UpdateThingShadow() and GetThingShadow() with the REST API.

Programming the launchpad had multiple parts: setting up the launchpad to readily receive and upload data via AWS, writing code to interpret the data received from AWS, writing functions for drawing on the OLED display, creating custom font characters for the numbers and certain letters, polling SW2 and SW3 for user input, and coming up with the ultimate process of the algorithm.

To set up the launchpad, the AWS root, client, and private certificates needed to be converter to .der format and flashed as User Files on the launchpad. Flashing them ensured that they could be consistently accessed as the program ran.

The function drawGraph() drew a the line-plot in a succession – starting from the least recent value until the most recent. The function drawPrice(double price) drew the crypocurrency symbol at the top left of the screen and the most recent price at the top right. A design decision was made to draw on the OLED display 90 degrees from its normal upright encoding. This decision was unnecessary and required custom font characters to be created.

MikroElektronika GLCD Font Creator was used to create the custom characters at 90 degrees. All digits from 0 – 9 were replaced as well as the characters A, B, C, D, E, and T.

To access the dynamic data via AWS, http_get() was written to use a rest API call (GET) to get the full state of a shadow. This function parsed the returned string and loaded it into a global double array.

C. Algorithm

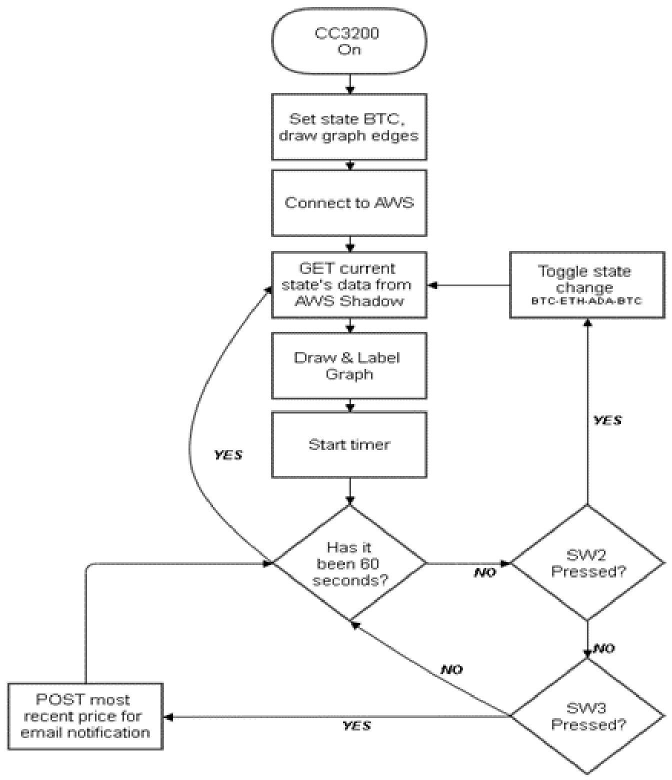
CC3200 On
Set state BTC, draw graph edges
Connect to AWS
GET current state's data from AWS Shadow
Toggle state change BTC-ETH-ADA-BTC
Draw & Label Graph
Start timer
YES
YES
Has it been 60 seconds?
NO
SW2 Pressed?
NO
NO
POST most recent price for email notification
YES
SW3 Pressed?

Figure 1: Flowchart for the algorithm at high-level

The algorithm begins by initializing the board, configuring the pinmux settings, and setting the state of the device to BTC. SPI is enabled and the OLED display is initialized. The screen is filled with black and the edges of the graph are drawn. UART is configured for serial monitoring via a terminal. The device connects to WiFi, sets the time, and then connects to the AWS server.

The device enters a while loop that checks the state and downloads the shadow data from AWS. The crypto symbol is drawn on the board, with the recent price. Followed by the line plot of the previous 20 minutes. The algorithm enters a second while loop that breaks after 60 seconds. Within this loop, SW2 and SW3 are polled for user input. If SW2 is pressed, the state changes and the main while loop advances. If SW3 is pressed, a message is posted to AWS, queuing an email, of the most recent price for the displayed cryptocurrency, to be sent to the user. The while loop repeats itself, self-updating every 60 seconds.

D. Protocols and Libraries

- Boto3 (Python): Used in each lambda function for directly reading and updating the shadow.
- Json (Python): Used in each lambda function for formatting the payload to write to the shadow and reading the payload from the shadow.
- Urllib (Python): Used in each lambda function for formatting data from the Crypto Compare API.
- Stdio, Stdint, Stdlib (C): Used in microcontroller for various operations.
- Time (C): Used for counting to 60 seconds before refreshing the OLED display.
- SPI: Communication with OLED display.
- UART: Monitoring Launchpad output via a terminal.

III. Results

With the implementation of the algorithm and configuration of AWS, the IoT device runs successfully. However, there are ways in which the device could be improved.

A. Areas for Improvement

- The buttons could be implemented with interrupts rather than polling. Given the frequency at which the buttons are expected to be pressed, they would be more efficient as interrupts. This would remove the need for the Time library (C).
- The device could automatically send the price via SMS or email after a massive price swing (~10% over 2 hours).
- A smaller microcontroller could be used to decrease the size of the device. In addition, a larger display could be used, where multiple cryptocurrencies could be displayed at once.

B. Potential Bugs

- The price value is displayed from left to right in a fixed number of characters. When a price eventually increases its number significant digits, it will be rendered incorrectly.
    - Solution: Count significant figures and align from right to left.
- The displayed symbols are hardcoded into the controller program. If they ever changed, they would stay the same on the device, rendering incorrectly.
    - Solution: Infrequently download the symbols via API. This will work as long as the API has its own keys for symbols.

IV. Conclusion

Ultimately, the device works as intended as an implementation of a CC3200 Launchpad, OLED display, and Amazon Web Services. It successfully, automatically updates every minute and responds properly to user input on the buttons. The buttons can switch the device between displaying Bitcoin, Ethereum, and Cardano data. The design, however, can certainly be improved with the implementation of interrupts instead of polling, automatic notifications, a smaller controller, a larger screen, and by increasing the adaptability to eliminate potential bugs.

REFERENCES

[1] R.Chloe Gregory, “Live Cryptocurrency Monitor using AWS Lambda,” Initial State.

[2] A. Havens, Beginners guide to creating a REST API, September 2021, Andrew Havens.