Implementing Optimized Brick Breaker in CC3200 with OLED and Accelerometer from Scratch

Eric Li (Xiaoyi)
University of California, Davis
Davis, U.S.A.
ercli@ucdavis.edu

***Abstract*—We can implement games with a CC3200 Launch Pad. However, a naive implementation will suffer from performance problems, such as limitation on SPI transfer speed. In this paper, I will go though how to implement and optimize a brick breaker game in CC3200. The game will use accelerometer, OLED, and the WiFi module. UART can be optionally used for debugging.**

***Index Terms*—embedded system, game, OLED**

I. Introduction

The CC3200 Launch Pad and OLED used for UC Davis EEC 172 instruction can be used to implement simple games. The accelerometer built in to CC3200 can be used as main input, the OLED can be used as main display of the game. The two push buttons in CC3200 can serve as game control (e.g. pause, restart). The CC3200 also provides WiFi capability, which allows us to do basic communication with the IoT services for score reporting etc.

II. Motivation

UC Davis EEC 172 labs requires implementing a simple game using OLED and accelerometer. However, students cannot easily see how a real-world embedded system game will face challenges on performance and memory limitations. Thus, I decide to implement a much complex game in order to reveal these difficulties. I will also incorporate knowledge from linear algebra, numerical methods, and computer vision to optimize the game. I will also add features like screen rotation and score fetching in this project.

III. Approach

This project uses the code from EEC 172 labs as templates. In the beginning of this project there are two goals in parallel. The first is to have a proof of concept work showing that all components used are functional and there are no pin conflicts when using all of them together. The second is to develop the logic for the game, which involves mainly software engineering.

After a naive implementation of the game, I can see performance problems of the game. For example, An analysis in Section VI-A shows that the provided graphics library contains severe inefficiencies when drawing characters in the OLED. So optimization on the code should be performed. For example only re-draw updates to a frame and caching the OLED state.

Finally, some additional features like screen rotation and score fetching can be added.

IV. Components Used

All components used in this project are provided in EEC 172 Lab. Specifically,

- CC3200 Launch Pad [1]
  - Accelerometer [2]
  - WiFi module
  - UART port (optional)
  - Push buttons
  - Slow Clock [3]
- OEL Display Module (OLED) [4]

A code similar to lab 4 is developed (not submitted) to prove that all the components can work independently without interfering each other. This becomes the "proof of concept" part of this project.

A. Communication Protocols

The CC3200 CPU communicates with the Accelerometer using I2C Protocol.

The CC3200 CPU communicates with the OLED using the SPI protocol.

B. Pin Mux Configuration

Pin Mux configuration is done using the TI PinMux tool.

- GPIO
  - GPIO09: Pin 64, output, for WiFi LED
  - GPIO13: Pin 4, input, for SW3
  - GPIO22: Pin 15, input, for SW2
  - GPIO25: Pin 21, output, for OLED R
  - GPIO28: Pin 18, output, for OLED OC
  - GPIO31: Pin 45, output, for OLED DC
- I^2C
  - SCL: Pin 1, for accelerometer
  - SDA: Pin 2, for accelerometer
- SPI
  - CLK: Pin 5, for OLED CL
  - MOSI: Pin 7, for OLED SI
- UART
  - RX: Pin 57, for UART
  - TX: Pin 55, for UART

TABLE I
JUMPER WIRE CONNECTIONS

CC3200			OLED
Pin	Location	Ref	Pin
VCC	J20		+
GND	J20		G
P05	P1	7	CL
P18	P2	2	OC
P45	P2	4	DC
P07	P2	6	SI
P21	P2	8	R

C. Jumper Wire Connections

Extra connections need to be made between the CC3200 Launch Pad and the OLED. 7 Jumper wires are required in total. See Table I for details.

V. GAME LOGIC

The game main logic is implemented in the `project_main()` function. It fetches the maximum score from Amazon AWS Shadow, and then call the `game()` function. After game ends it sends the score and displays texts such as "Game Over".

The `game()` function implements the game. It first initializes all variables such as the location and velocity of the ball and the tray. The bricks are initialized in two ways: pre-defined pattern or random blocks. It also draws the OLED screen. Then it enters a while loop that iterates once per frame.

1) Compute the new location of the ball using Euler method in numerical methods, and compute collision between the board and the tray, the bricks, and the border. This part of the code extensively uses simple linear algebra computations for 2D vectors. The vectors are defined as a C struct called `vect_t` and helper functions like `normalize` and `dot_prod` are written.
2) Draw the updated OLED screen, which overwrites the original position of the ball, the tray, and the broken bricks, and draws the new positions and the texts (e.g. score).
3) Check input (accelerometer reading, push button status).
4) Wait until the correct timing for next frame by polling the Slow Clock.

VI. OPTIMIZATIONS

A. Analysis on Provided Graphics Library

An analysis to the provided graphics library in EEC 172 Lab 2 shows that there are a number of inefficiencies. One example is the `drawChar()` function. A character of size 1 occupies 48 pixels, and to draw one pixel `drawChar()` calls `fillRect()` of size 1 48 times. Each time `fillRect()` need to transfer 9 bytes to the OLED (7 bytes for setting location, 2 bytes for the color of the rectangle). In total, $9 \times 48 = 432$ bytes need to be transferred for each character.

However, if we just use 7 bytes to set the location and then use $2 \times 48 = 96$ bytes to specify the pattern of the 8×6 rectangle, only $7 + 96 = 103$ bytes need to be transferred, saving more than 75% of the time.

B. Rewriting Graphics Library

I rewrote the graphics library provided. The only thing I reused is the `glcdfont.h`, which defines the fonts. Since most objects in the game are square-shaped, drawing them is straight forward. The characters are drawn using the method discussed in Section VI-A. The ball is drawn by calculating the euclidean distance from the pixel to the center of the ball.

C. Caching OLED memory

The OLED is 128×128, and each pixel is represented by 2 bytes. So to store the entire OLED memory, only $128 \times 128 \times 2 = 32768$ bytes are needed, which is 32 KiB, which is small enough to be stored in the memory.

The normal way the screen is initialized is: draw the black background, draw each brick, draw the ball, etc. This creates inefficiencies because the same pixel may be drawn multiple times.

Notice that the multiple writes are going through the SPI protocol, which is the bottleneck for performance. So we can cache the OLED state. Specifically, we compute the new OLED screen in memory (i.e. cache), and then flush it to OLED once the computation is done. We can also optimize by only flushing the part that are updated.

In the implementation, the OLED memory is cached as a global variable `virt_oled` (in the data section). The `render()` function flushes the data to OLED memory. The programmer can specify a rectangular region for `render()` so that only the changed part of the screen can be updated.

VII. EXTRA FEATURES

A. Score Fetching

The game communicates with Amazon AWS service for storing and loading score.

It uses HTTP GET method to get the maximum score from AWS each time the CC3200 Launch Pad starts. There is a simple parser implemented using `strstr()` and `sscanf()` to extract the maximum score from the HTTP response.

It uses HTTP POST method to send the current score and updated maximum score to AWS, and the settings in AWS can allow subscribers to receive text messages for score updates.

As a result, the user can see the maximum score in the board.

B. Screen Rotation

Using the spirit of the adapter method, caching OLED makes the communication with the OLED only through the `render()` function. Thus, we can do simple tweaks to this function in order to support the 4 screen attitudes.

At the start of the game, the accelerometer is read and which attitude to use is computed.

We also need to do some changes to the logic converting the accelerometer reading to the tray speed.

VIII. Reproducing

Here are the procedures to reproduce this work

A. Requirements

- CC3200 Launch Pad (CC3200-LAUNCHXL)
  - USB cable
- Adafruit 1.5" SSD1351 128x128 RGB OLED
- 7 jumper wires
- A WiFi access point
- An Amazon AWS account
- Software such as Code Composer Studio, CCS UniFlash

B. Wire Connections

- Connect the OLED screen with the CC3200 Launch Pad. See Table I.
- Connect the CC3200 to a computer using the USB cable.

C. Setting Up AWS

1) Go to AWS IoT [5], create a Thing. Create certificates for it and activate it. Also download Amazon AWS's Root CA.
2) Copy the REST API Endpoint host name from the Interact tab of the thing.
3) Go to IoT Policies and add a policy for allowing `iot:GetThingShadow` and `iot:UpdateThingShadow`.
4) Attach this policy to the certificates you have generated.
5) Use OpenSSL to convert the certificates to der format.
6) Create an SNS topic and create a subscription for it (with your phone number).
7) Create an IoT Rule that sends a message as an SNS push notification when the shadow updates.

D. Updating Code

In `main.c`, Update macro definitions for `DATE`, `MONTH`, `YEAR`, etc. using current date and time. Update `SERVER_NAME`, `POSTHEADER`, `GETHEADER`, `HOSTHEADER` using AWS settings in Section VIII-C.

You also need to update your WiFi access point information in `common.h` in the CC3200 SDK.

E. Flashing Program

The flashing configuration is saved in `project.usf`. You need to change `/sys/mcuimg.bin` to use the project binary, `/cert/rootCA.der`, `/cert/client.der`, `/cert/private.der` to use the files from Section VIII-C. After you are done click the "Program" button.

IX. Future Work

- Add more features to the game, like animation of brick disappearing.
- Restructure the code to allow it easily be ported to other platforms (e.g. computers).
- Allow users to select game difficulty.
- Allow multiple users to exchange their game scores in AWS (currently if multiple players are playing the game there will be a race condition on score updating).

X. Conclusion

This project implements a realistic brick breaker game from scratch using the CC3200 Launch Pad and OLED provided in UC Davis EEC 172 class. The game makes use of the accelerometer connected through I^2C, the OLED connected through SPI, the WiFi module, the UART port, and Slow Clock. It contains game logic and collision detection algorithms, and includes highly optimized code for graphics operations. It also have features like rotating screen and fetching scores.

Acknowledgment

I would like to thank the UC Davis Electrical and Computer Engineering Department for providing equipments (CC3200 Launch Pad, OLED) for this project. I would also like to thank Professor Homayoun and EEC 172 TAs for their instructions.

References

[1] Texas Instruments, "CC3200 SimpleLink Wi-Fi and Internet of Things Solution With MCU LaunchPad Hardware User's Guide," June 2014, Revised March 2020.

[2] Bosch Sensortec, "BMA 222 Digital, triaxial acceleration sensor Data sheet," May 2012.

[3] Texas Instruments, "CC3200 SimpleLink Wi-Fi and Internet-of-Things Solution, a Single Chip Wireless MCU Technical Reference Manual," June 2014, Revised May 2018.

[4] Univision Technology Inc, "OEL Display Module Product Specification," October 2008.

[5] Amazon Web Services, "AWS IoT Developer Guide," April 2020.