Portable Nintendo Entertainment System (NES) emulator on ESP32

Sunday 26th May, 2019

Akhsarbek Gozoev

***Abstract*—Some of us was lucky enough to own a NES, some wasn't. In this article i will try to describe how one can create a system emulating original NES for roughly 1/25 of its price while being portable and giving hours of gameplay. All this possible thanks to Espressif Systems's ESP32 microcontroller for being powerful enough without draining too much energy.**

I. Introduction

After reading this article you will be introduced to the insights of how to create portable NES emulator and our end-product will look like this:

The code is heavily based on Espressif own port of NES emulator located on Github: esp32-nesemu.

II. Hardware

The complete system consists of **ESP32 microcontroller** as the main brains, **ILI9341-based LCD** for image output, vintage **SNES controller** as input and some source of power, either **LiPo-battery** or Micro-USB charger can be used.

A. ESP32 vs Arduino

ESP32 microcontroller is developed and manufactured by Espressif Systems with a release date being September 6, 2016. The reason i chose ESP32 over **Arduino** platform is simple. Arduino boards are mainly based on **AVR microcontrollers**, which does not have the processing power required to comfortably emulate NES, nor does it have enough RAM to do so. ESP32 on the other hand is equipped with:

1) **CPU**: Xtensa dual-core 32-bit LX6 microprocessor, operating at 160 or 240 MHz and performing at up to 600 DMIPS
2) **Memory**: 520 KiB SRAM
3) **Wi-Fi**: 802.11 b/g/n
4) **Bluetooth**: v4.2 BR/EDR and BLE

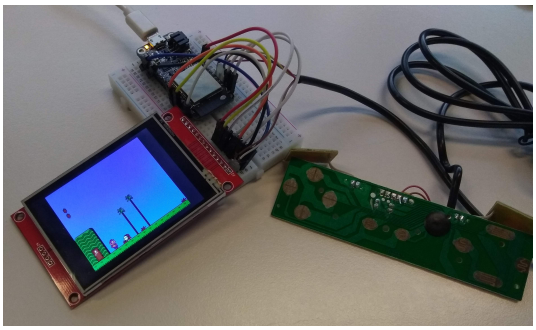
And while I don't use Wi-Fi and Bluetooth yet, they may be used for remote ROM uploading and connecting multiple devices to be able to run 2- or even 4-player games.

B. Why not RaspberryPi

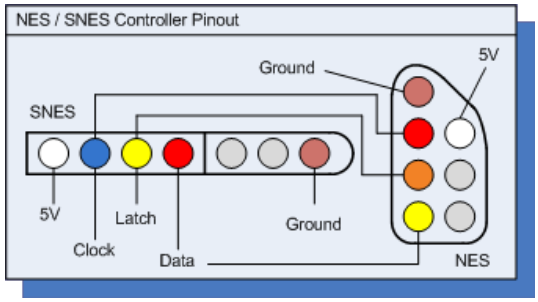
Another contender which comes to mind is RaspberryPi (rPi), as it seems everybody's 1st project with rPI are retro game emulator system. This would perhaps be too easy as there are thousands of tutorials on the Internet. Another big concern of mine about using RaspberryPi and it's analogues was their power consumption. These small computers were designed to be used as desktop/office computers and that's the power that's always there, eating up precious energy, while we can't utilize it fully while playing NES games. We could of course try disabling all unnecessary modules and maybe end up with the same result as using ESP32 but the main focus of my work then would be stripping rPi of its features, rather than working on NES emulator. And that's not nearly as fun.

C. Gamepad

Ideally NES controller should be used as we are emulation NES system, but I didn't had any laying around. SNES controller has 12 buttons (D-pad with 4 directions, Start, Select, A, B, X, Y and side buttons L and R), while NES game pad only had D-pad, Start Select A and B, which makes it 8. We can decide how to use 4 buttons left out. In my case i chose to use side buttons to reset emulator, and X, Y just mirrored A and B. In future when this emulator will have more functionality these buttons may be remapped.



As noted earlier there are 12 buttons, but only 5 wires coming out of the gamepad. Those would be Vcc, Clock, Latch, Clock and Ground. Below is the image illustrating it. Image is taken from Hackaday article [1] about how to connect Snes controller to Arduino, but since the code is simple and self-explaining it is not hard to port it to ESP32 platform. The protocol used is already described in the article so I'm not going to repeat it again here.



D. Screen

Original code used ILI9341-based LCD screen. There are most commonly used color screens in hobby electronic projects. They use SPI-protocol for communication, more often then not these modules come with build in SD card reader, and i will discuss later how i may use SD card in the future. The whole code related to outputting game image to the screen resides in one file 'spi_lcd.c' which may be modified to adapt this emulator to use screens with different lcd-drivers.

E. Power source

During development power via USB cable was used. Mainly to avoid needing to recharge the battery over and over again. In the final version 830 mAh Li-Po battery was used and it lasted for 3 hours before the experiment had to be canceled due to the lack of time. Following this tutorial[2] it should be possible to measure battery percentage Even if the author used MongooseOS, we are using bare C and we should be able to implement everything MongooseOS can.

III. Case

The case was developed in **Fusuion360** and printed on **Ultimaker 2+** over the night. It's fairly simple design where i recreated Snes controllers layout while leaving some space for the screen on top.

IV. Future Work

A. Multiple ROMs on SD-card

Most of ILI9341-based LCD screens have build-in SD- / microSD-card readers. It should be possible to rewrite original code to be able to list and load ROMs from SD-card. This gives us the possibility to have dozens and dozens of games in our pocket rather than only one as it is right now.

B. Second Player

It doesn't take long for us to understand how game AI's work and beating them over and over again. At this point a second player is required for us to enjoy the same old games. And it is always more fun to play together with some friends.

C. TV-out

When playing with multiple people it's not hard to predict the screen size will be the biggest holdback in never ending battles. The first thing popping into the mind is creating some sort of output to bigger screen (eg. TV). The most used protocols today are HDMI and older AV-signal. HDMI is licensed by HDMI LLC and maybe difficult to implement on ESP32. AV on the other hand is old well known format and every TV have one. It should not be a problem to stream our games to big screen, and even if we experience any lag we can always disable the original ILI9341 screen output as most likely no one will be using it when connected to bigger TV-screen

D. Sound

When writing game system emulators sound is often implemented last as the main focus is always what happens on the screen. but without sound we are unable to recreate those nostalgic moments from the past as sound is our second most used and developed sense and there are a lot of wonderful sound effects in our favorite games.

E. Multiple systems

Implement support for multiple systems, kinda like duo-booting, like Sega Genesis, GameCube etc.

V. Conclusion

It was possible to recreate a fully functional system emulating NES console which runs on same or close to real system's speed not allowing us to notice any lag, thus discomfort. The system is fairly portable as it does not require to be plugged in while operating and can give at least 2 hours of gameplay, which is not bad taking into the account that no optimization was done, and both Bluetooth and Wi-Fi modules are on during up-time.

[1] SNES Controller Arduino Adapter: https://hackaday.io/project/7498-snes-controller-arduino-adapter

[2] Battery Monitoring on the Adafruit Huzzah32 ESP32 with MongooseOS: http://cuddletech.com/?p=1030