Project 4: Longan Nano Pong Game/ Other Game

Computer Architecture I ShanghaiTech University

Project 3 Project 4

IMPORTANT INFO - PLEASE READ

The projects are part of your design project worth 2 credit points. As such they run in parallel to the actual course. So be aware that the due date for project and homework might be very close to each other! Start early and do not procrastinate.

Goal

In this project, we hope you can use RISC-V programming to implement a game using Longan Nano board.

You have two options: Either you implement a Pong game, or implement your own game if you want.

Also, we will only provide the basic framework as in the Lab 11 starter - we expect that you should be capable of doing a project from scratch. Of course, the library functions are attached.

Getting started

Make sure you read through the entire webpage before starting the project.

You will be using gitlab to collaborate with your group partner. Autolab will use the files from gitlab. Make sure that you have access to gitlab. In the group CS110_Projects you should have access to your project 4 repository. Also, in the group CS110, you should have access to the p4_framework.

Hardware and Software setup

Refer to Lab 11 for details. The following is a quick navigator:

- Software setup
- Hardware setup

Obtain your files

1. Clone your Project 4 repository from GitLab.
2. In the repository add a remote repo that contains the framework files:
   ```
   git remote add framework
   https://autolab.sist.shanghaitech.edu.cn/gitlab/cs110/p4_framework.git
   ```

3. Go and fetch the files:
   `git fetch framework`
4. Now merge those files with your master branch:
   `git merge framework/master`
5. The rest of the git commands work as usual.

Files

The framework contains the following files:

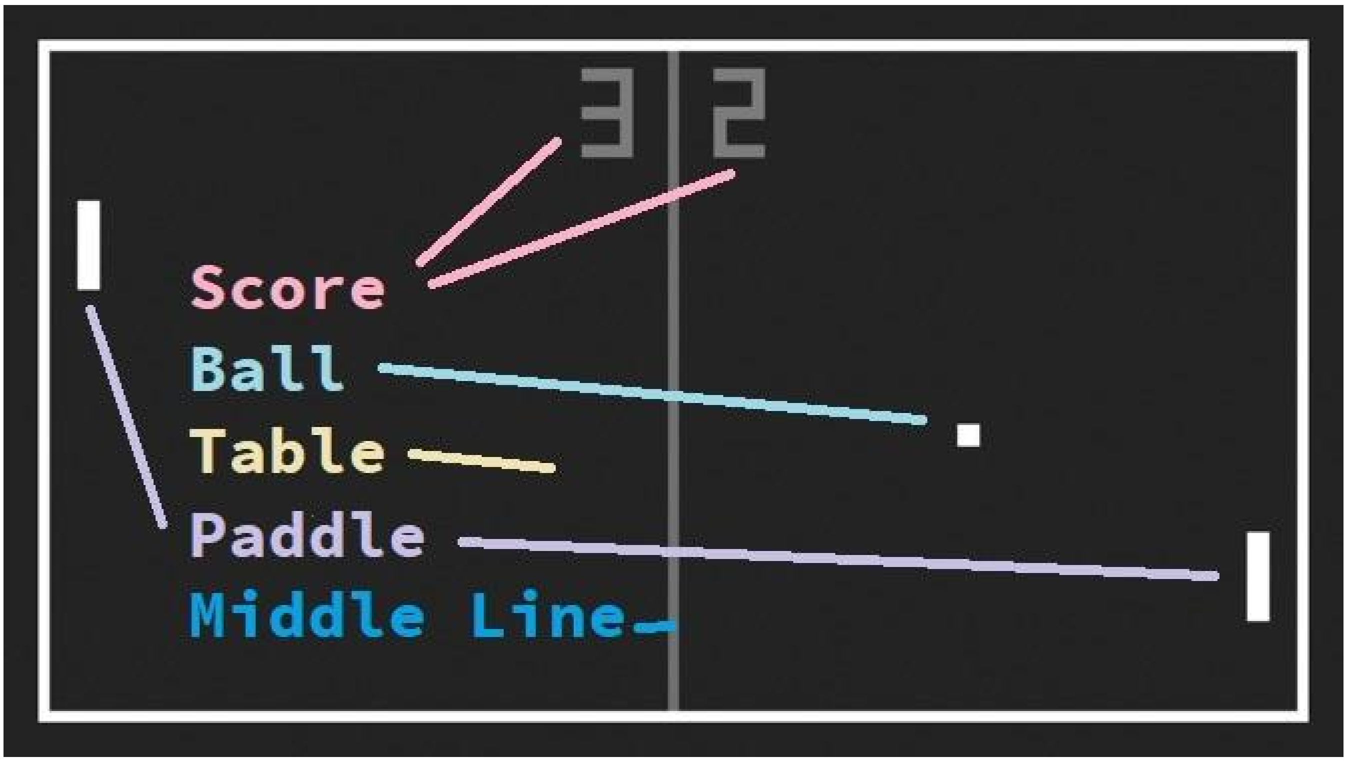
```
.
├── LICENSE
├── Makefile
├── README.md
├── dfu-util
├── include
│   ├── README
│   ├── fatfs
│   │   ├── diskio.h
│   │   ├── ff.h
│   │   ├── ffconf.h
│   │   └── tf_card.h
│   ├── gd32v_pjt_include.h
│   ├── gd32vf103_libopt.h
│   ├── lcd
│   │   ├── bmp.h
│   │   ├── lcd.h
│   │   └── oledfont.h
│   ├── systick.h
│   └── utils.h
├── platformio.ini
└── src
    ├── assembly
    │   └── example.S
    ├── fatfs
    │   ├── 00history.txt
    │   ├── 00readme.txt
    │   ├── ff.c
    │   ├── ffsystem.c
    │   ├── ffunicode.c
    │   └── tf_card.c
    ├── lcd
    │   └── lcd.c
    ├── main.c
    ├── systick.c
    └── utils.c
```

Please read `src/utils.c`, `src/assembly/example.S`, `src/systick.c` and `src/lcd/lcd.c` carefully. They contain helper functions and/or infomation that may not be mentioned in this webpage.

Pong Game Introduction

The first Pong game was manufactured in the year of 1972, by the founders of the ATARI company. There was a story that only several hours after the game console was placed, it broke down because users can insert no more coins - The storage was full of coins.

The original game looked like this:



And there is an online demo here. For details, please refer to this link.

Requirements

Coding

We expect you to program something in RISC-V on a real machine - So we require that:

- You implement at least 3 RISC-V functions, and the total LoC is larger than 300 (comments excluded).
- For tidiness, please put all the assembly functions (.S files) in `src/assembly`. No other files or directories should appear in that folder.
- For assembly code: You need to have comments not less than 40% of the total lines of RISC-V code you wrote. A comment is defined by a sentence followed by #. Comments have to be **meaningful** and **in English**.
    - Actually, `riscv-nuclei-elf-as` allows you to write comments by using `//` and `/* ... */`; but we don't want you to do this for tidiness. Those comments will not count.
- The project can be successfully compiled via PlatformIO.

Game

There is no solid restriction for the Pong game - the details of the game are up to you.

However, we're saying "Pong Game" here - we expect that, if you want to implement a Pong, at least your game should look like Pong. We want your Pong game to "look good" - Otherwise your score will be deducted.

The above has shown a standard Pong game and a online demo. Your Pong game should more or less look like that. Here are a few basic requirements for a nice-looking Pong game:

- Two players: A human player, and another program-controlled "CPU" player. (The demo game has the human player on the right, but you're free to determine this.)
- The "ping-pong table" has to be big enough - it cannot just occupy a size of 10x5 pixels!
- There should be some kind of "middle line" on the table - It has to be right in the center of the screen.
- The "table" is a rectangle; when the ball hits the edge of the table, it should be bounced back.
- When someone failed to bounce the ball back with his paddle, his opponent gets 1 score.
- There should be a scoreboard showing your and your opponent's score real-time.

Hints

- You don't have to emulate the complex physics when the ball hits someone's paddle - you can let it bounce randomly in that case.
- When implementing RISC-V functions, recall the calling conventions that you've learned - which registers should be saved by th caller and which by the callee?
- Think twice before you write something - it is not convenient to debug on this board. We recommend you write a framework first, and add functionalities one by one.
- If you want to make use of advanced features of the board, here is a bunch of links that will help you:
  - Official document
  - Official download site (contains documents, specifications, HDK, SDK, etc)
  - Official example project (bad apple)
- If you face problems that can't be solved by your group, feel free to come to TAs' office hour for help.

Grading Policy

If you did not pass the check on Autolab, you will get zero score.

We will grade your project based on the latest commit that passed the Autolab check. You'll get 80% of score if you implemented a Pong that works (can correctly respond to the button input, no matter ugly or not); you'll get the rest 20% if we think that your Pong game is a nice-looking one.

Grading Policy Other Game

If you did not pass the check on Autolab, you will get zero score.

We will grade your project based on the latest commit that passed the Autolab check. You'll get 80% of score if you implemented a game that works (can correctly respond to the button input, no matter ugly or not) and that at least reaches the complexity of a Pong game; you'll get the rest 20% if we think that your Game is a nice-looking one.

Submission

Submit changes to your remote GitLab repo by running the following commands in your local repo folder:

```
$ git commit -a
$ git push origin master:master
```

How to Autolab

Similar to previous projects, upload your `autolab.txt` to Autolab to submit your project.

Submission Time Announcement

The last time of your submission to the git repo will count towards your submission time - also with respect to slip days. So do not commit to this git after the due date, unless you want to use slip days or are OK with getting fewer points.

Collaborative Coding and Frequent Pushing

You have to work at this project as a team. We invite you to use all of the features of gitlab for your project, for example branches, issues, wiki, milestones, etc.

We require you to push very frequently to gitlab. In your commits we want to see how the code evolved. We do NOT want to see the working code suddenly appear - this will make us suspicious.

We also require that all group members do substantial contributions to the project. This also means that one group member should not finish the project all by himself, but distribute the work among all group members!

At the end of Project 4 we will interview all group members and discuss their contributions, to see if we need to modify the score for certain group members.

Schwertfeger, Sören <`soerensch` AT `shanghaitech.edu.cn`>
Chundong Wang <`wangchd` AT `shanghaitech.edu.cn`>
TA: Ze Song <`songze` AT `shanghaitech.edu.cn`>
Last modified: 2020-06-03