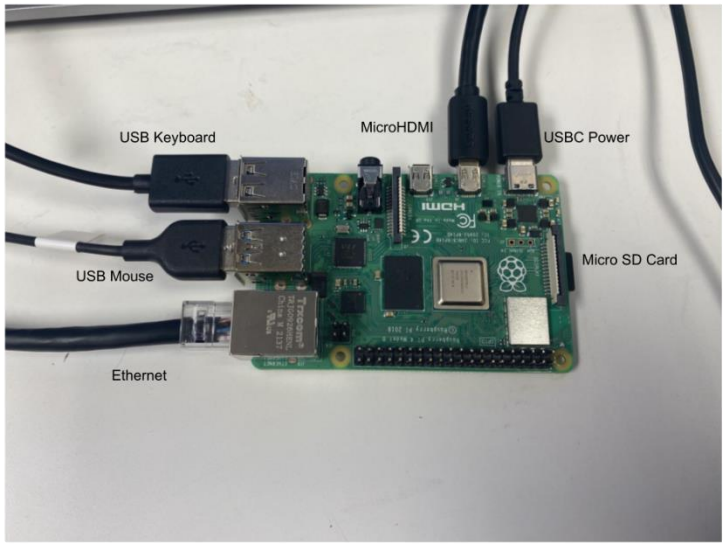
# FreeRTOS SPI Controller Documentation

FreeRTOS is a package which can allocate RAM to user-defined tasks and run them as real-time threads. While the processing and user-facing software can be run in Linux, the SPI commands can be run real-time, improving timing. With a combination of Ubuntu OS, C/C++ and FreeRTOS, timing is far more consistent, making operation with an Intan system possible. Installation instructions are provided below.

Hardware needed:

- Raspberry Pi 4B (at least 2GB of RAM)
- MicoSD card (at least 32GB RAM)
- PC with microSD card slot or adapter for microSD card.
- USBC cable for power
- MicroHDMI cable and HDMI monitor
- Ethernet cable and ethernet wall connection
- USB mouse and keyboard



Steps for installation and compilation:

1. Download Raspberry Pi Imager on your PC
2. Select Ubuntu Desktop 22.04.1 LTS (RPi 4/400) to be burned onto the SD under the "CHOOSE OS" tab on Imager, under CHOOSE STORAGE chose the SD card, and flash the OS onto the card.
3. Plug the SD card, ethernet cable, mouse, keyboard and micro HDMI-HDMI cable into the Raspberry pi. (HDMI should be connected to monitor). Then, plug the wall outlet in, a boot screen should appear on the screen.
4. Configure the Raspberry Pi system for preferred language and location
5. Choose a name for your Raspberry Pi
6. Choose a login name and password
    i. User name: heather
    ii. Password: AFAFAF
7. Choose "Automatically … button" and not "Use password"
8. Create an account. All our devices are using "AFAFAF" as a password. Go through the initialization process until you are on the home screen
9. Download Visual Studio Code for Ubuntu here onto your raspberry pi (.
    i. Select the arm x64 bit .deb version for installation on the raspberry pi 4
10. Go to this link and download the latest version of FreeRTOS (202112.00)
11. Open a terminal window
12. Execute these terminal commands in order:
    i. >>cd Downloads/
    ii. >>sudo apt install build-essential
    iii. >>sudo apt install./code_1.71.0-1662017130_arm64.deb(The debian file downloaded from the onedrive)
    iv. >>code
13. In visual studio code, go into settings and in the search bar look up "telemetry settings" and disable crash reporter and telemetry.
14. Back in terminal, execute the following commands:
    i. >>unzip FreeRTOSv202112.00.zip (should match the download file from the OneDrive)
    ii. >>sudo chown heather /opt (heather is username defined in step 6)
    iii. >>mv FreeRTOSv202112.00 ~/FreeRTOSv202112.00(The folder name of the unzipped package)
    iv. >>cd ..
    v. >>export FREERTOS_PATH=~/FreeRTOSv202112.00
    vi. >>export FREERTOS_PATH=/opt/FreeRTOSv202112.00
    vii. >>nano .profile
15. In the file that opens in a text editor, add "export FREERTOS_PATH=~/FreeRTOSv202112.00" to the bottom.
    i. Save with ctrl+x

ii. Enter “Y” to save
iii. Hit Enter to quit

16. Back in terminal, execute:
    i. >>sudo apt install git
    ii. >>cd Documents
    iii. >>mkdir Project_FreeRTOS
    iv. >>cd Project_FreeRTOS
    v. >>git clone https://github.com/vsserafim/twotasks-posix-gcc.git
    vi. >>code

In Visual studio code, click File => Open Folder and open Twotasks-posix-gcc.

17. The updated code is here, replace main.c code in the Visual Studio project with this code.
18. Open view>command pallet and execute “run build task”
19. Right click on “build” in explorer and select “open in integrated terminal”
20. Execute “sudo ./modelo-posix-gcc", this should run the current script.
21. For any changes to the script, change the main.c file in the /src folder, but remember to build before successive executions of the program

A note on customization:

When customizing main.c to your specific needs, it is important to keep in mind what happens inside FreeRTOS and what happens outside. If you need a real-time task, you must instantiate a unique task function and add it inside the task scheduler in main. Depending on the parameters to the scheduler, you can spend more/less time in a task or change between the tasks more/less often. The rest of the script, outside these functions and the schedule, will be executed in Ubuntu C/C++, and will not be beholden to FreeRTOS. Generally, timing-critical components like SPI commands should be instantiated as FreeRTOS tasks, and other functions like post-processing, frontend, etc. should be executed in the main script.