**Library 1**

ECSE4235: Embedded Systems II

Small Project – 2024 – Timothy Million, Sreya Bitra



**Documentation**

**Top Level Library Structure**

**Library**

**E4235.h - header file**

**src - stores function assembly code**

**bin - contains the .o files for the functions**

**tests - stores test code**

**testbin - contains .o files for the test code**

**makefile**

**libe4235.a - static library formed from the .o files of the function assembly (not working yet)**

**Makefile**

**make library** - remake the static library in case new functions are added

**make build [name of test] -** will build one specified test, either in assembly or c

**make run [name of test]** - will run one specified test

**Ex.** make build whatami\_asmtest

make run whatami\_asmtest

> will build and run an assembly file that uses the E4235\_whatami() function call

**make clean** - clean all files inside of testbin

**Blocking/Deblocking**

The blocking / deblocking function is designed to offer a way of enabling or disabling the function on the terminal. The primary purpose of this function is a switch function to allow the user to easily turn off or on blocking on the terminal so they can decide whether they want their program to wait for an input or to continue.

In Assembly:

mov r0, #input

bl E4235\_deblock

In C:

int E4235\_deblock(input);

**deblocking:**

This code toggles the terminal’s mode between blocking and non-blocking. It does so by interacting with file control settings via the fcntl system call.

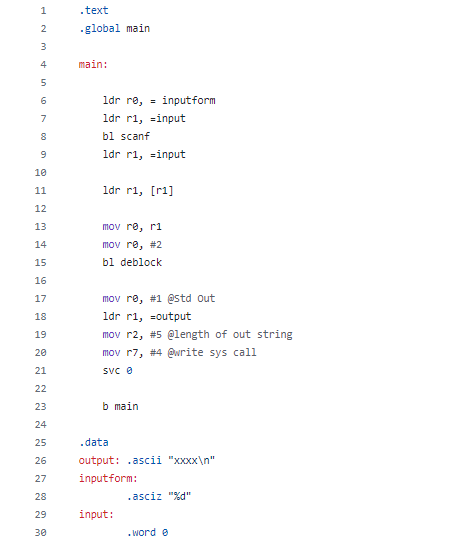
**Parameters:**

R0, the register used to specify the desired mode for the terminal. A value of 0 sets the terminal to blocking mode, a value of 1 sets it to non-blocking mode.

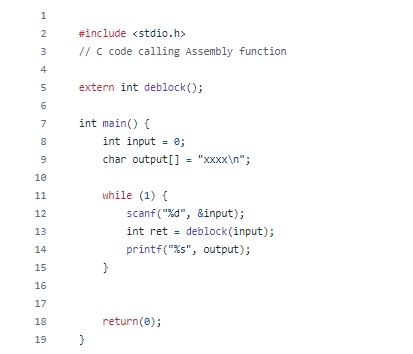
**Returns:**

R0, the result of the last fcntl call. If the input value is neither 0 nor 1, it does not modify the terminal's mode and sets R0 to -1, indicating an invalid input.

**Testing**



**FIGURE X: Calling Assembly**

****

**FIGURE X: Calling C**

**Objective:**

The objective of the test plan is to test the toggle functionality, what happens when the toggle function is called repeatedly, and to test the function’s error handling.

**Toggle Functionality:**

Confirm the function successfully toggles the system's blocking state.

**Procedure:**

Ensure that system is in a known blocking state initially. Then invoke the library function to toggle the state to non-blocking. Verify that the state has changed and then call it again to return its state back to the original blocking state. The expected outcome is that the function correctly toggles the state of the deblocking switch.

**Repeated Toggle:**

Validate that repeated toggling does not lead to system instability or incorrect states.

**Procedure:**

Repeatedly call the function around 10 times, making sure that the function is correctly toggling between the blocking and non-blocking states. After each toggle, the state is verified to be as expected.

**Error Handling:**

Ensure the function properly handles situations where toggling the blocking state is not possible

**Procedure:**

Create a scenario where toggling the blocking state should fail, attempt to toggle to the blocking state, and make sure an error is returned. Making sure to verify that the state remains unchanged after the error.

**Whatami()**

The whatami function is an assembly language library that has the primary purpose of enabling users to determine various recorded stats of the processor. This function is intended for users who need to find a specific statistic to use in their program.

**Whatami():**

The whatami() function in ARM Assembly is designed to provide specific statistics about the Raspberry Pi 4 it's running on.

**Parameters:**

R0, the integer that specifies which statistic to return.

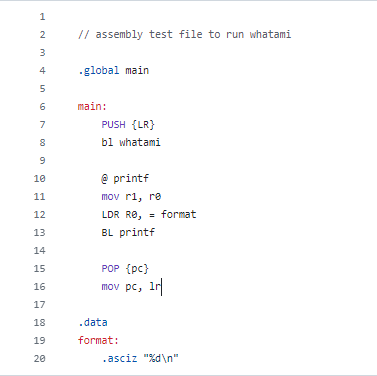
Options:

1 = CPU Frequency in Hz

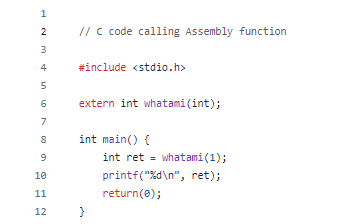
**Returns:**

R0, if the input is 1, the CPU frequency in Hz. If the input was not recognized it returns -1 to indicate an invalid request.

**Testing**

****

**FIGURE X: Calling Assembly**

****

**FIGURE X: Calling C**

**Objective:**

The objective with the test plans was to test the basic functionality of retrieving the CPU’s maximum frequency, as well as a stability test to ensure that the library consistently reported the correct frequency after multiple executions.

**Basic Functionality Test:**

Validate that the whatami function’s Get Frequency function correctly identifies and returns the RP4's maximum CPU frequency as a positive integer value.

**Procedure:**

Write the code to navigate to the RP4’s system files to locate the scaling\_max\_freq file. Invoke this function using both ARM Assembly and C. Open the file and output the data collected to standard output.

**Stability Test:**

Confirm the Get Frequency function's reliability by repeatedly invoking it and verifying it consistently reports the same maximum CPU frequency.

**Procedure:**

Invoke the get frequency function 10 times making sure that each iteration returns the frequency recorded in the pi. Record each output making sure that it is matching.

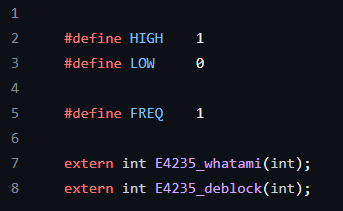
**Current Issues/Drawbacks to fix**

Only works in a certain level of hierarchy, need to change to look for the file regardless of function’s position in the hierarchy

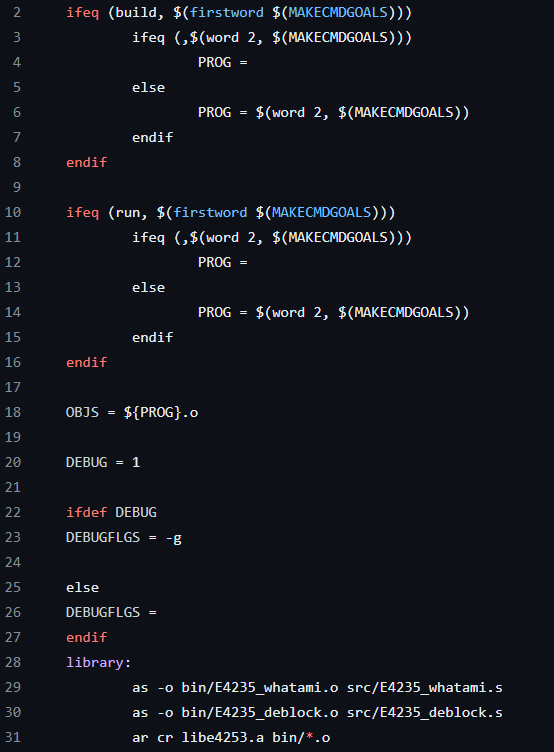
**Appendix**

**Section 1 - Code**

**E4235.h**

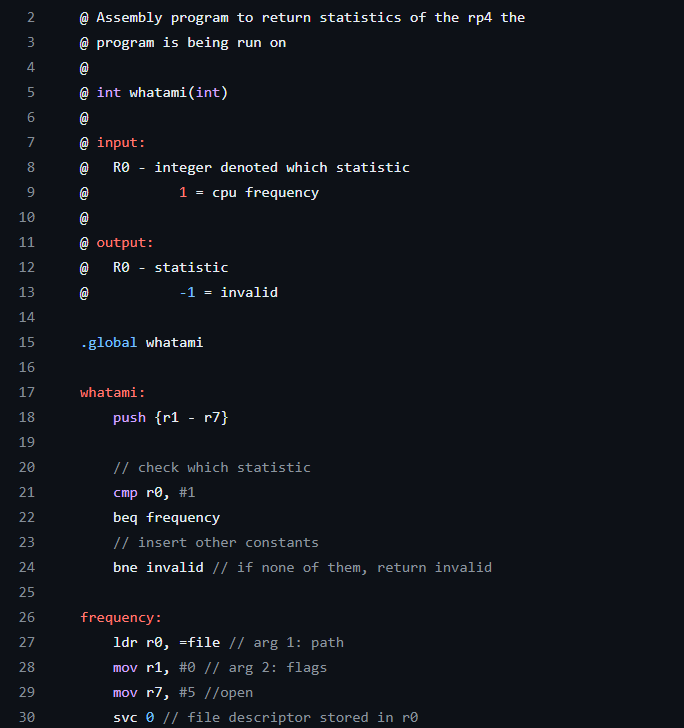
****

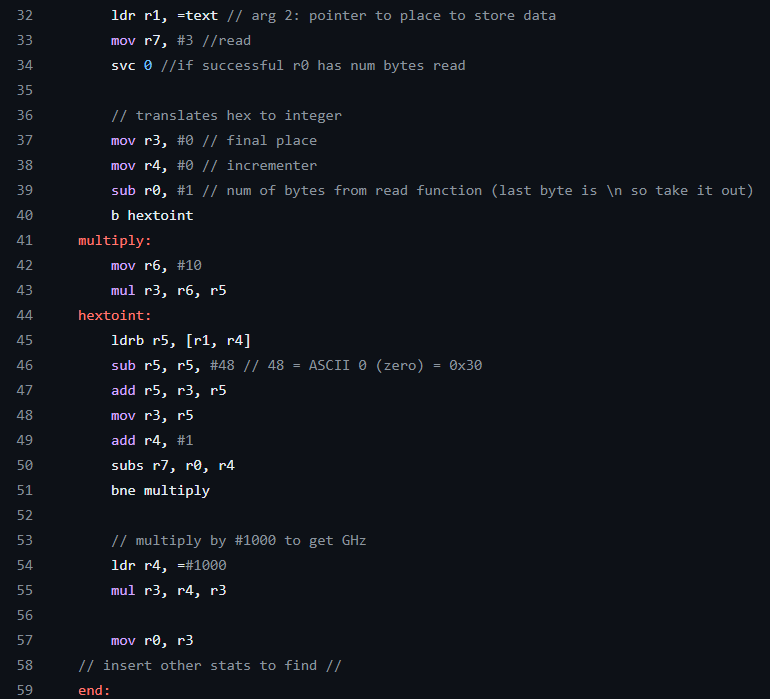
**makefile**

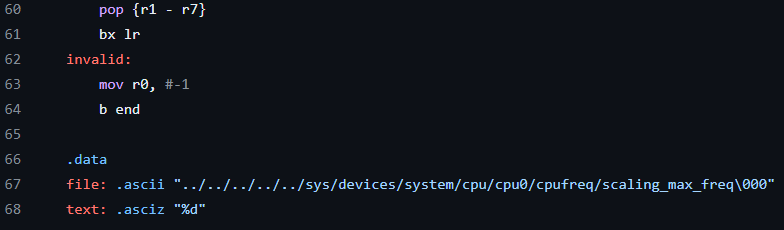
****

****

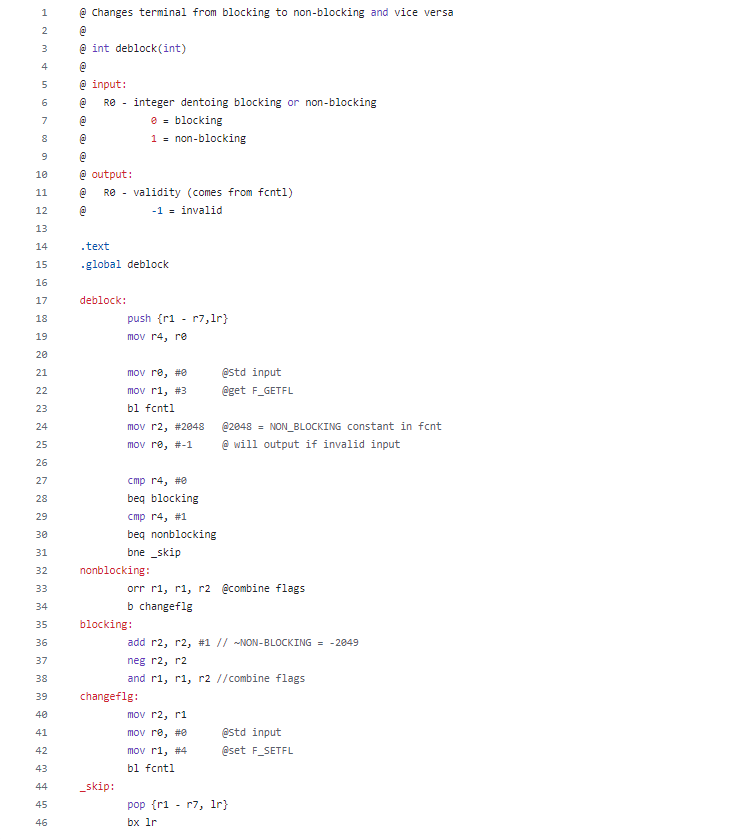
**whatami() Code:**

****

****

****

**Blocking/Deblocking Code:**

****

**References**

1. “BCM2711." Raspberry Pi Documentation, Raspberry Pi Foundation, <https://www.raspberrypi.com/documentation/computers/processors.html#bcm2711>.

2. <https://stackoverflow.com/questions/47910759/what-is-the-difference-between-ranlib-ar-and-ld-for-making-libraries#:~:text=ar%20creates%20or%20updates%20a,misleading%20to%20give%20them%20the%20>.