



College of Engineering
UNIVERSITY OF GEORGIA

Library 1 Components

19 March, 2024

Olivia Beattie & Isaac Garon

Blocking & Deblocking

E4235_KYBdeblock

| int E4235_KYBdeblock (int state) | |
|------------------------------------|--|
| Overview | This function controls the blocking status of the standard input file |
| Parameters | state - the state of the standard input deblocking |
| Returns | error_code - an integer that describes if the function executed successfully or not 0 - deblocking successfully set 1 - state integer out of bounds error |

Description

This function controls the blocking status of the standard input file. Essentially, setting this function to ON results in standard input reads returning immediately. Setting this function to OFF results in a standard input read blocking the program from continuing until an input is read. If the function is successful, it returns a 0. If a number other than 0 or 1 is input, an integer out of bounds error is generated and a 1 is returned. This function is useful when the user wants to get input while continuously updating an output.

Examples

E4235_KYBdeblock(1)

Unblocks standard input. Any scanf or read calls will return immediately.

E4235_KYBdeblock(0)

Blocks standard input. Any scanf or read calls will wait to return until a newline character or carriage return is read.

What Am I

E4235_WhatAmI

| int E4235_WhatAmI () | |
|----------------------|--|
| Overview | This function returns the maximum frequency of the central processing unit (CPU) |
| Returns | frequency - the frequency of the processor in Hz |

Description

This function is used to get information about the central processing unit for the Raspberry Pi 4. Upon the call of the function, the program will return the maximum frequency of the CPU in Hertz.

Examples

E4235_WhatAmI()

Returns 1800000000.

Top Level

The library structure is essential to creating an environment for using the library components and updating future functionality. The header file, E4235.h, contains all of the required includes and defines for the library; each function's signature is contained in the header using the `extern` keyword. Additionally, the E4235 library uses directories to organize files used for both development and testing. The `src` directory contains all of the library functions - each written in assembly in its own file. The `test` directory stores both assembly and C programs that validate the functionality of each library function; the compiled test code is stored in the `bin` directory. Finally, the `obj` directory contains all of the object files created by compiling the source files. These object files are linked together, and the final library, `libE4235.a`, is created. In all, the library contains a `src` directory which stores every assembly function, and the `libE4235.a` and `E4235.h` files are created to link the library when it is called from C.

To call the library from C, the `libE4235.a` and `E4235.h` files must be generated and stored in the proper locations. A compile script compiles each individual program in the `src` folder and stores its object file in the `obj` directory. Then, the `libE4235.a` library is generated by linking every object file in the `obj` directory together. The install script then copies the library file into `/usr/local/lib` and copies the header file into `/usr/local/include`. This allows `gcc` to find the header and library files without explicitly specifying the path. To use the library in a C file, the header, `E4235.h`, must be included. Then, when compiling using `gcc`, the flag, `-lE4235`, must be added to the compilation command. This tells the compiler to link the `E4235.a` library stored in `/usr/local/lib` when compiling the current C file.

Compilation Example: `gcc test.c -o test -lE4235`

```
isaacgaron@IDGs-MacBook-Pro Library % tree
.
├── E4235.h
├── bin
│   ├── KYBdeblock_ASM_Test
│   ├── KYBdeblock_C_Test
│   ├── WhatAmI_ASM_Test
│   └── WhatAmI_C_Test
├── compile.sh
├── install.sh
├── libE4235.a
├── obj
│   ├── E4235_KYBdeblock.o
│   └── E4235_WhatAmI.o
├── src
│   ├── E4235_KYBdeblock.s
│   └── E4235_WhatAmI.s
└── test
    ├── KYBdeblock_ASM_Test.s
    ├── KYBdeblock_C_Test.c
    ├── WhatAmI.png
    ├── WhatAmI_ASM_Test.s
    └── WhatAmI_C_Test.c
```

Figure 1. Library Top Level Structure

Test Plan

KYBdeblock

The following test cases will be executed to verify the functionality of the KYBdeblock function. The tests will check the function under both nominal conditions and edge cases.

Case 1: Deblock OFF

This case checks that the function can be used to set the terminal to BLOCKING mode. In BLOCKING mode, the program will wait for input from the user to continue.

Implementation:

The deblock off functionality was tested by writing a program that checks for user input and then prints a string to the terminal. If the program detects an input of 'b' the program will switch the terminal to blocking mode using the function calls below. While the terminal is set to blocking mode, the program will only print to the terminal once each time a key is entered on the keyboard. The program can be quit by entering 'q'.

Assembly Function Call: `mov r0, #0`
`bl KYBdeblock`

C Function Call: `E4235_KYBdeblock(0)`

Verification:

- (1) Run the program
- (2) Enter 'b' to enter blocking mode
- (3) Ensure that the program prints "hello there general kenobi" to the terminal each time the enter key is pressed
- (4) Enter 'q' to end the program

Results: **PASSED**

When the program was run and 'b' was input, the program only printed "hello there general kenobi" after a key was pressed. When 'q' was entered, the program quit, verifying that it received the input.

Case 2: Deblock ON

This case checks that the function can be used to set the terminal to NON-BLOCKING mode. In NON-BLOCKING mode, the program will not wait input from the user to continue.

Implementation:

The deblock on functionality was tested by writing a program that checks for user input and then prints a string to the terminal. If the program detects an input of 'd' the program will switch the

terminal to non-blocking mode using the function calls below. While the terminal is set to non-blocking mode, the program will continuously print to the terminal while also accepting keyboard input. The program can be quit by entering 'q'.

Assembly Function Call: `mov r0, #1`
`bl KYBdeblock`

C Function Call: `E4235_KYBdeblock(1)`

Verification:

- (1) Run the program
- (2) Enter 'd' to enter non-blocking mode
- (3) Ensure that the program prints "hello there general kenobi" to the terminal continuously
- (4) Enter 'q' to end the program

Results: **PASSED**

When the program was first run, the terminal only printed "hello there general kenobi" when the enter key was pressed. After 'd' was input, the program continuously printed "hello there general kenobi". Finally, when 'q' was entered, the program quit, verifying that it still received input while in the non-blocking state.

Case 3: Invalid Input

This case checks that the function will return an error code if the user enters an invalid input.

Implementation:

For a successful execution, the program will return 0; if the function detects an error it will return 1 to indicate that the input argument was out of range. The invalid input case was checked by calling the function with an input of 5 using the function calls below. The program will compile because 5 is an int and follows the input parameter guidelines, but the function will return the error code.

Assembly Function Call: `mov r0, #5`
`bl E4235_KYBdeblock`

C Function Call: `int return = E4235_KYBdeblock(5)`

Verification:

- (1) Run the program
- (2) The function should return an error code of 1
 - (a) In assembly, use GDB to view the registers and check the return register
 - (b) In C, check the return value of the function

Results: **PASSED**

After the execution of the assembly program, a value of 1 was stored in the return register R0. When the C program was run, the function returned a value of 1.

WhatAmI

The following test cases will be executed to verify the functionality of the WhatAmI function. The tests will check the function under both nominal conditions and edge cases.

Case 1: Calling the Function

This case checks that the function will return the maximum frequency of the CPU when called.

Implementation:

The return value functionality was checked by writing a program to call the WhatAmI function using the function calls below and print the maximum CPU frequency to the terminal. For the Raspberry Pi 4, the user can expect to see either 1.5 GHz or 1.8 GHz.

Assembly Function Call: `bl E4235_WhatAmI`

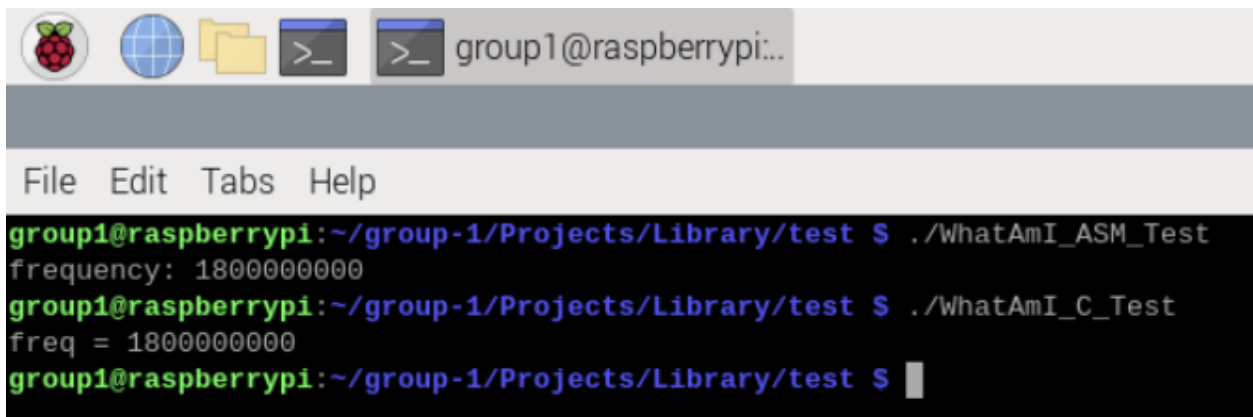
C Function Call: `int freq = E4235_WhatAmI()`

Verification:

- (1) Run the program
- (2) Ensure that the frequency printed to the terminal is in range

Results: **PASSED**

Upon execution of the program, a maximum frequency of 1.8 GHz was printed to the terminal. The results for both the C and ASM programs are shown in Figure 2.



```
group1@raspberrypi:~/group-1/Projects/Library/test $ ./WhatAmI_ASM_Test
frequency: 1800000000
group1@raspberrypi:~/group-1/Projects/Library/test $ ./WhatAmI_C_Test
freq = 1800000000
group1@raspberrypi:~/group-1/Projects/Library/test $
```

Figure 2. WhatAmI Verification