Library 1

ECSE4235: Embedded Systems II

Small Project – 2024 – Timothy Million, Sreya Bitra

UNIVERSITY OF
GEORGIA

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

```
1     .text
2     .global main
3
4     main:
5
6         ldr r0, = inputform
7         ldr r1, =input
8         bl scanf
9         ldr r1, =input
10
11        ldr r1, [r1]
12
13        mov r0, r1
14        mov r0, #2
15        bl deblock
16
17        mov r0, #1 @Std Out
18        ldr r1, =output
19        mov r2, #5 @length of out string
20        mov r7, #4 @write sys call
21        svc 0
22
23        b main
24
25    .data
26    output: .ascii "xxxx\n"
27    inputform:
28            .asciz "%d"
29    input:
30            .word 0
```

FIGURE X: Calling Assembly

```c

#include <stdio.h>
// C code calling Assembly function

extern int deblock();

int main() {
    int input = 0;
    char output[] = "xxxx\n";

    while (1) {
        scanf("%d", &input);
        int ret = deblock(input);
        printf("%s", output);
    }


    return(0);
}
```

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

```
1
2   // assembly test file to run whatami
3
4   .global main
5
6   main:
7       PUSH {LR}
8       bl whatami
9
10      @ printf
11      mov r1, r0
12      LDR R0, = format
13      BL printf
14
15      POP {pc}
16      mov pc, lr
17
18  .data
19  format:
20      .asciz "%d\n"
```

FIGURE X: Calling Assembly

```
1
2   // C code calling Assembly function
3
4   #include <stdio.h>
5
6   extern int whatami(int);
7
8   int main() {
9       int ret = whatami(1);
10      printf("%d\n", ret);
11      return(0);
12  }
```

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

```
1
2       #define HIGH    1
3       #define LOW     0
4
5       #define FREQ    1
6
7       extern int E4235_whatami(int);
8       extern int E4235_deblock(int);
```

makefile

```
2       ifeq (build, $(firstword $(MAKECMDGOALS)))
3               ifeq (,$(word 2, $(MAKECMDGOALS)))
4                       PROG =
5               else
6                       PROG = $(word 2, $(MAKECMDGOALS))
7               endif
8       endif
9
10      ifeq (run, $(firstword $(MAKECMDGOALS)))
11              ifeq (,$(word 2, $(MAKECMDGOALS)))
12                      PROG =
13              else
14                      PROG = $(word 2, $(MAKECMDGOALS))
15              endif
16      endif
17
18      OBJS = ${PROG}.o
19
20      DEBUG = 1
21
22      ifdef DEBUG
23      DEBUGFLGS = -g
24
25      else
26      DEBUGFLGS =
27      endif
28      library:
29              as -o bin/E4235_whatami.o src/E4235_whatami.s
30              as -o bin/E4235_deblock.o src/E4235_deblock.s
31              ar cr libe4253.a bin/*.o
```

```
32
33
34      build:
35      ifeq (asm,$(patsubst %asmtest,asm,$(PROG)))
36              as $(DEBUGFLGS) -o testbin/$(OBJS) tests/$(PROG).s
37              gcc -o testbin/$(PROG) testbin/$(OBJS) src/* -lc
38      else
39              gcc -I Library -o testbin/${PROG} tests/${PROG}.c src/*.s
40      endif
41
42      debug:
43              gcc -g -o ${PROG} ${PROG}.s whatami.s
44
45      run:
46              testbin/$(PROG)
47
48      #cleanup
49      clean:
50              rm testbin/*
```

whatami() Code:

```
2       @ Assembly program to return statistics of the rp4 the
3       @ program is being run on
4       @
5       @ int whatami(int)
6       @
7       @ input:
8       @   R0 - integer denoted which statistic
9       @           1 = cpu frequency
10      @
11      @ output:
12      @   R0 - statistic
13      @           -1 = invalid
14
15      .global whatami
16
17      whatami:
18          push {r1 - r7}
19
20          // check which statistic
21          cmp r0, #1
22          beq frequency
23          // insert other constants
24          bne invalid // if none of them, return invalid
25
26      frequency:
27          ldr r0, =file // arg 1: path
28          mov r1, #0 // arg 2: flags
29          mov r7, #5 //open
30          svc 0 // file descriptor stored in r0
```

```
32          ldr r1, =text // arg 2: pointer to place to store data
33          mov r7, #3 //read
34          svc 0 //if successful r0 has num bytes read
35
36          // translates hex to integer
37          mov r3, #0 // final place
38          mov r4, #0 // incrementer
39          sub r0, #1 // num of bytes from read function (last byte is \n so take it out)
40          b hextoint
41      multiply:
42          mov r6, #10
43          mul r3, r6, r5
44      hextoint:
45          ldrb r5, [r1, r4]
46          sub r5, r5, #48 // 48 = ASCII 0 (zero) = 0x30
47          add r5, r3, r5
48          mov r3, r5
49          add r4, #1
50          subs r7, r0, r4
51          bne multiply
52
53          // multiply by #1000 to get GHz
54          ldr r4, =#1000
55          mul r3, r4, r3
56
57          mov r0, r3
58      // insert other stats to find //
59      end:
```

```
60          pop {r1 - r7}
61          bx lr
62      invalid:
63          mov r0, #-1
64          b end
65
66      .data
67      file: .ascii "../../../../../sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq\000"
68      text: .asciz "%d"
```

Blocking/Deblocking Code:

```
@ Changes terminal from blocking to non-blocking and vice versa
@
@ int deblock(int)
@
@ input:
@   R0 - integer dentoing blocking or non-blocking
@          0 = blocking
@          1 = non-blocking
@
@ output:
@   R0 - validity (comes from fcntl)
@          -1 = invalid

.text
.global deblock

deblock:
        push {r1 - r7,lr}
        mov r4, r0

        mov r0, #0      @Std input
        mov r1, #3      @get F_GETFL
        bl fcntl
        mov r2, #2048   @2048 = NON_BLOCKING constant in fcnt
        mov r0, #-1     @ will output if invalid input

        cmp r4, #0
        beq blocking
        cmp r4, #1
        beq nonblocking
        bne _skip
nonblocking:
        orr r1, r1, r2  @combine flags
        b changeflg
blocking:
        add r2, r2, #1 // ~NON-BLOCKING = -2049
        neg r2, r2
        and r1, r1, r2 //combine flags
changeflg:
        mov r2, r1
        mov r0, #0      @Std input
        mov r1, #4      @set F_SETFL
        bl fcntl
_skip:
        pop {r1 - r7, lr}
        bx lr
```

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.