Alexandria University
Faculty of Engineering
Computer and Systems Engineering Department
CS 401 – Graduation Project

Graduation Project
General-Purpose Computer and Game Console

Firmware Technical Reference

Student:

- Mostafa Abd El-Aziz (65)

Supervisor:

- Prof. Dr. Layla Abo Hadid

Abstract:

This document describes the firmware program used in the proposed MIPS computer. Firmware is a piece of software that provides control over the electronic system, and stored in non-volatile memory chip that is attached to the system.

The firmware described in this document can be seen as a complete interactive operating system for our MIPS computer. It has device drivers for the PS/2 keyboard and VGA interface, and includes an embedded shell that allows running some utilities.

Source code organization:

The firmware system is mainly written in C (and little assembly). All source code files exist under firmware/ directory. The following is a listing for the files in firmware/ directory, and their description:

<u>C files:</u>

```
calc.c          Calculator code.
kbd.c           PS/2 Keyboard device driver.
main.c          Contains main() routine.
mem.c           Utility to read/write from/to memory.
shell.c         Firmware embedded shell.
string.c        String processing routines.
vga.c           VGA device driver.
```

Header files:

`calc.h`	Header for calc.c
`kbd.h`	Header for kbd.c
`mem.h`	Header for mem.c
`shell.h`	Header for shell.c
`string.h`	Header for string.c
`vga.h`	Header for vga.c

Assembly files:

`start.s`	Code executed by CPU when the machine starts up.
`test.s`	Assembly code to print "Hello World" on screen.

Linker scripts:

`firmware.ld`	Linker script processed by GNU linker (ld).

Make scripts:

`Makefile`	Makefile processed by make command.

Compiling, Linking, and Loading:

Makefile automates the compilation process. All you have to do is to type “make” in terminal to compile the program. Compilation process requires gcc for MIPS and binutils for MIPS to be installed on your machine. Some GNU/Linux distributions provide prebuilt packages for mipsel-gcc and mipsel-bintuils.

The compiler and assembler generate object files for the C files and assembly files listed above. The linker then links them together into a single binary file called “firmware.bin”. The object code in start.s is allocated in the beginning of the binary file.

After “firmware.bin” is generated, the file is to be flashed onto the ROM on FPGA board, or used as firmware image for the emulator.

Initialization:

When the computer is powered on, the CPU starts executing the instructions on the ROM, starting from address 0. As explained above, the linker allocates first instructions in the ROM images for the object code of start.s. The code in start.s simply initializes the stack, then calls main() routine in main.c.

The main procedure calls clear_screen() from vga.c to clear and repaint VGA screen, then calls kbd_init() from kbd.c to initialize the keyboard driver. After printing some information about the machine, the code calls shell() from shell.c to enter shell cycle.

VGA Device Driver:

VGA text buffer appears in the memory region (0x18000 to 0x1FFFF). vga.c file contains routines to interact with the buffer and provides an API with functionality that is similar to (but very simplified) printing functions in C stdio library:

- **print_char()**: used to print character on screen and updates cursor location.
- **print_int()**: prints decimal integer on screen and updates cursor location.
- **print_hex()**: prints hexadecimal integer on screen and updates cursor location.
- **print_str()**: prints string and updates cursor location.
- **print_hf()**: prints string in the middle of a line on the screen.
- **print_fmt()**: like printf() in C.

PS/2 Keyboard Device Driver:

The PS/2 keyboard controller is accessed through two registers at 0xFFF00000 (**KBDDATA**) and 0xFFF00004 (**KBDSTS**). KBDSTS always reads 0, unless somebody has pressed on some keyboard key. As long as the key is in the controller buffer, KBDSTS reads 1. Once the key is removed from buffer (by reading KBDDATA register to catch the key), KBDSTS returns back to 0 state.

The driver provides routines to interact with KBDDATA and KBDSTS registers. get_from_buf() keeps polling KBSTS until it is 1, which means that keypress event has taken place. The function then calls scproc() which gets the scancode of the pressed key by reading KBDDATA, then translates the scancode into ASCII code, and handles special keys like shift, alt, numpad, numlock, etc...

The driver also provides an API with functionality that is similar to (but very simplified) scanning functions in C stdio library:

- **scan_char()**: simply calls get_from_buf() to get a single ASCII character.
- **scan_str()**: returns a complete string, terminated with NULL terminator.
- **scan_int()**: similar to scanf("%d", &ret_val) in C.

String Library:

Several parts of the firmware program uses string manipulation routines to handle strings. string.c provides an API with functionality that is similar to (but very simplified) string functions in C string library:

- **str_len**(): returns length of C string.
- **str_cmp**(): compares to C strings.
- **str_to_int**(): converts a string that contains ASCII digits into integer data type.

Shell:

The shell code in shell.c allows user to interact with the firmware through keyboard and screen. The life cycle of the shell program consists of 5 states:

1. Print shell prompt on screen.
2. Use scan_str() in kbd.c to scan a command line.
3. Parse the command.
4. Execute command.
5. Go to 1.

Currently, the shell program only supports these simple commands:

- **help**: prints listing of supported commands.
- **version**: prints firmware version information.
- **calc**: loads a simple calculator (calc.c).
- **mem**: loads a simple memory debugging program (mem.c).
- **test**: executes code in test.s.
- **clear**: clears screen.
- **reboot**: restarts the firmware program.
- **shutdown**: stops the firmware program.