Microprocessor Systems Lab 6 Report

Weiheng Zhuang Ziyu Zhu

Table of Contents

Task 1: USB Mouse

Introduction

Task 1 is a project using a Micro-B to USB OTG adapter plugged into CN15, detecting a USB mouse and displaying its connection status on the terminal. It should detect when a device has been plugged or unplugged and behave accordingly. Alternatively, the movement of the mouse can be used to control the terminal cursor, with the buttons providing other functions. In this Task, left click is designed to print an "o" on the terminal and right click removes the current cursor position.

USB is an asynchronous bus which uses differential signaling , USB 1.0, 1.1, and 2.0 use only 4 pins (VDD, D+, D-, GND). USB 3.0 bumped this up to 9 pins, adding RX1-, RX1+, TX1-, TX1+, and another GND to allow for full-duplex serial data transmission; with USB 3.2 additionally providing RX2-, RX2+, TX2-, TX2+, among some other pins. Multiple devices may be connected to the same USB host port as well, generally through the use of USB hubs, which essentially just duplicate the connections and possibly provide external power to supplement the host's limited supply. There also exist combination plugs like the Micro-AB port on the DISCO board (CN15) that signify the device can act as either a host or peripheral, though this is not very common. Dual-role Micro-USB devices (e.g., the DISCO, smartphones) are referred to as USB On-the-Go (USB OTG), which is a specification defining devices that can act as USB hosts but are predominantly meant to be USB peripherals.

To configure the STM32F769NI to enable the USB OTG HS connection in ULPI mode, many pins are necessary. Although USB is a serial protocol, the ULPI implementation used here first conveys the information (in bytes) to be transmitted over the USB link to the external PHY chip through a parallel interface via the eight USB_OTG_HS_ULPI_D# signals. Each of the pins should be configured with the OTG HS alternate function (GPIO_AFx _OTG_HS), in addition to being in alternate function mode (push-pull) with no pull up or down and in high speed mode.

Low Level Description

Mice and keyboards are known as Human Interface Devices (HID), which is a specific class of usb devices. Unlike HAL that was used before, the USB Host library is separated to initialize the pins. USBH_UserProcess is a built-in function, but can be modified in the code. It contains three cases: disconnection, active and connection. The program should print out a message when the mouse is connected or disconnected, so a line of print function should be written under the corresponding case.

```
    USBH_Init(&hUSBHost, USBH_UserProcess, 0);
    USBH_RegisterClass(&hUSBHost, USBH_HID_CLASS);
    USBH_Start(&hUSBHost);
```

Unlike the interrupt functions that were written before, USBH interrupt does not have IRQHandler functions. It only has the USBH_HID_EventCallback function. Inside the callback function, the USBH should update the mouse information when the data is changed. The data that is send from the mouse is a pointer HID_MOUSE_Info_TypeDef*. x and y are the distance the mouse has moved, and buttons[0] is left click, buttons[1] is right click, buttons[2] is middle click. They are 1 is clicked, and 0 is released.

```
   typedef struct _HID_MOUSE_Info
   { uint8_t              x;
     uint8_t              y;
     uint8_t              buttons[3];}
```

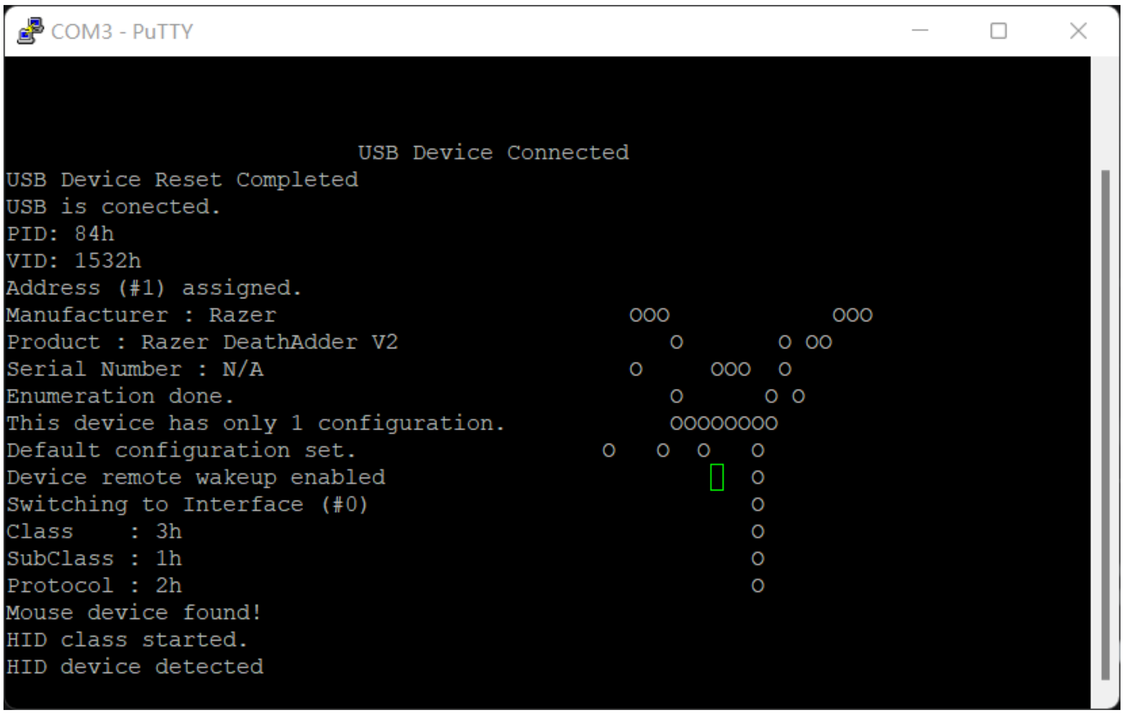
So, the callback function should first check if the interrupt is triggered by the mouse, then get the mouse info and update the parameter. Shown in the following codes.

```
   void USBH_HID_EventCallback(USBH_HandleTypeDef *phost) {
   if (HID_Handle->Init == USBH_HID_MouseInit)
       {   mouseinfo = USBH_HID_GetMouseInfo(phost);
           xnow = mouseinfo->x;
           ynow = mouseinfo->y;
```

Because the terminal is in default size 80x24, the sensitivity of horizontal and vertical movement is different, so the position should be corrected to move the cursor in a comfortable way.

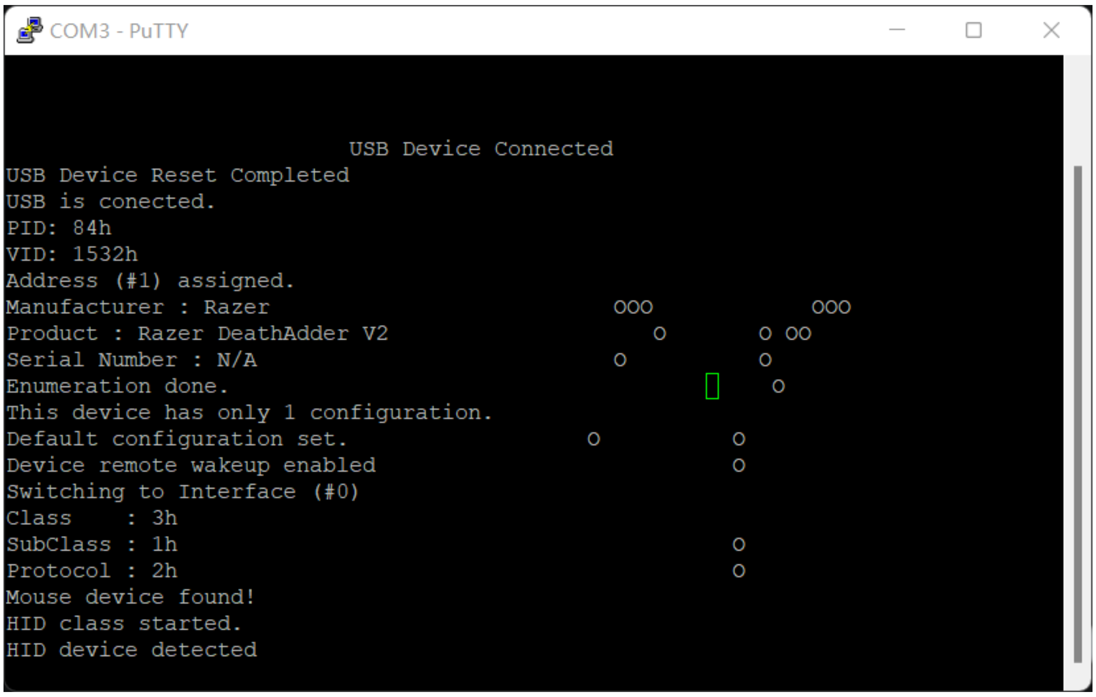
```
   xorigin += xnow * 0.02;
   yorigin += ynow * 0.0015;
   printf("\033[%d;%dH", (uint8_t)yorigin, (uint8_t)xorigin);
```

Results and Analysis



```
                          USB Device Connected
USB Device Reset Completed
USB is conected.
PID: 84h
VID: 1532h
Address (#1) assigned.
Manufacturer : Razer                          OOO            OOO
Product : Razer DeathAdder V2                    O       O OO
Serial Number : N/A                           O     OOO  O
Enumeration done.                                O      O O
This device has only 1 configuration.            OOOOOOOO
Default configuration set.                  O   O  O   O
Device remote wakeup enabled                        ▯  O
Switching to Interface (#0)                            O
Class    : 3h                                          O
SubClass : 1h                                          O
Protocol : 2h                                          O
Mouse device found!
HID class started.
HID device detected
```

Figure 1: USB Mouse left click



```
                          USB Device Connected
USB Device Reset Completed
USB is conected.
PID: 84h
VID: 1532h
Address (#1) assigned.
Manufacturer : Razer                          OOO            OOO
Product : Razer DeathAdder V2                    O       O OO
Serial Number : N/A                           O          O
Enumeration done.                                    ▯    O
This device has only 1 configuration.
Default configuration set.                  O          O
Device remote wakeup enabled                           O
Switching to Interface (#0)
Class    : 3h
SubClass : 1h                                          O
Protocol : 2h                                          O
Mouse device found!
HID class started.
HID device detected
```

Figure 2: USB Mouse right click

As the figure shows, the mouse parameters are printed in the terminal, the mouse can control the cursor and print/erase in the terminal.

Task 2: USB Mass Storage

Introduction

FatFs is a general-purpose file system (FAT/exFAT) module for implementing FAT file systems in small embedded systems. The writing of FatFs components complies with ANSI C (C89) and is completely separated from the disk I/O layer, so it does not depend on the hardware platform. It can be embedded into resource-constrained microcontrollers such as 8051, PIC, AVR, ARM, Z80, RX, etc. without any modification. Since the FatFs module is completely separated from disk I/O layer, it requires the following functions to lower the layer to read/write physical disk and to get current time. The low level disk I/O module is not a part of FatFs module and it must be provided by the user. The sample drivers are also available in the resources.

To set up FatFs, the user must provide the functions for FatFs to perform diskio. These are in the struct Diskio drvTypeDef. An instance of this struct, with all the variables initialized, is passed to the function FATFS LinkDriver. The "path" argument should be "0:/" as the first drive connected. The FatFs drivers (Diskip_drvTypeDef) are provided by STM and the applicable one for USB is included in the FatFs/inc/ directory.

For this task, we need to use a Micro-B to USB OTG adapter plugged into CN15, read a FAT/FAT32 USB 2.0 (HS) flash drive (most flash drives would work) and display its contents on the serial terminal. This program also adds on to Task 1 such that the program will respond to the mouse or the flash drive without user intervention.

Low Level Description

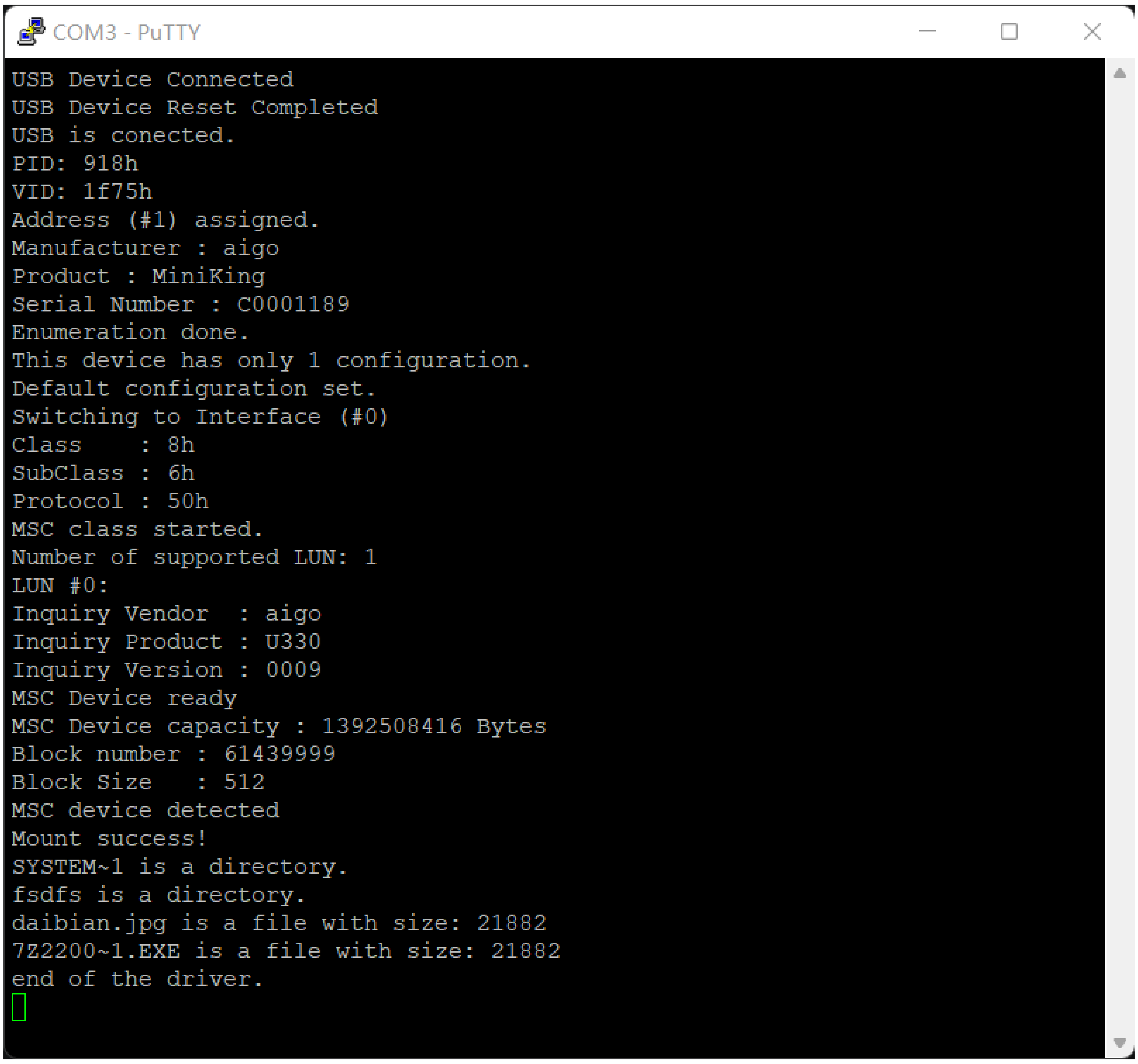
For this task, we have the experience of the previous task, so we can directly complete the USBH_RegisterClass function to initialize the startup class of the hard disk. Then follow the steps in the lab documentation. Note that a flash drive in USB terms is a Mass Storage Class (MSC). First declare an array of size 4 of type char for the "path" variable that will be used frequently later. then just use the FATFS_LinkDriver function to substitute our path parameter. In this way, we successfully link the USBH driver handle with the FatFs driver.

```
    FATFS_LinkDriver(&USBH_Driver, mynewdiskPath);
    USBH_RegisterClass(&husbh, USBH_HID_CLASS);
    USBH_RegisterClass(&husbh, USBH_MSC_CLASS);
    USBH_Start(&husbh);
```

Then we can start to write the equation to read the internal files of the U disk. First of all, we need to mount the target hard disk, call the function f_mount, if its return value is FR_OK, it means we can enter the next stage. Then we need to call the f_opendir equation, if its return value is also FR_OK, it means that our stage is also successful. It is worth noting that we need to declare in advance a variable whose data structure is DIR to carry the return value of the f_opendir equation. The variable of the DIR data structure carries the directory object structure. The next thing we have to do is to read all the files in the root directory by calling the f_readdir function in a loop. The f_readdir function returns a variable of fno data type. This variable carries the file name, creation date, size, attributes and many other variables. By consulting the official documentation of FatFs Driver, we know that fno.fattrib & AM_DIR can be used to determine whether the fno data is a directory. Through a simple if to judge whether it is a directory, and then call fno.fname to print out the file name, we have basically completed all the tasks of reading the flash file. Because it needs to be compatible with the content of task1, we can add the distinction between HID devices and MSC devices in the USBH_UserProcess equation. By using the built-in parameter pActiveClass of USBH_HandleTypeDef, we can know the device class currently in use, and then we can judge the device information through the open variable set in the main equation.

Results and Analysis

The screenshots show the output interface of this task. It works perfectly.



```
USB Device Connected
USB Device Reset Completed
USB is conected.
PID: 918h
VID: 1f75h
Address (#1) assigned.
Manufacturer : aigo
Product : MiniKing
Serial Number : C0001189
Enumeration done.
This device has only 1 configuration.
Default configuration set.
Switching to Interface (#0)
Class    : 8h
SubClass : 6h
Protocol : 50h
MSC class started.
Number of supported LUN: 1
LUN #0:
Inquiry Vendor  : aigo
Inquiry Product : U330
Inquiry Version : 0009
MSC Device ready
MSC Device capacity : 1392508416 Bytes
Block number : 61439999
Block Size   : 512
MSC device detected
Mount success!
SYSTEM~1 is a directory.
fsdfs is a directory.
daibian.jpg is a file with size: 21882
7Z2200~1.EXE is a file with size: 21882
end of the driver.
```

Figure 3: output results

Conclusions

Through this lab, we focused on learning the example operations of USB devices on STM32F769NI, as well as the basic reading of files with FatFs of USB protocol.

USB

The DMA method is to directly rely on the hardware to transfer data between the main memory and the device, and does not require CPU intervention during this period. It is characterized by randomness, higher parallelism, fast transmission speed and simple operation. Applications:

(1) Data block transfer for high-speed external memory such as disks;

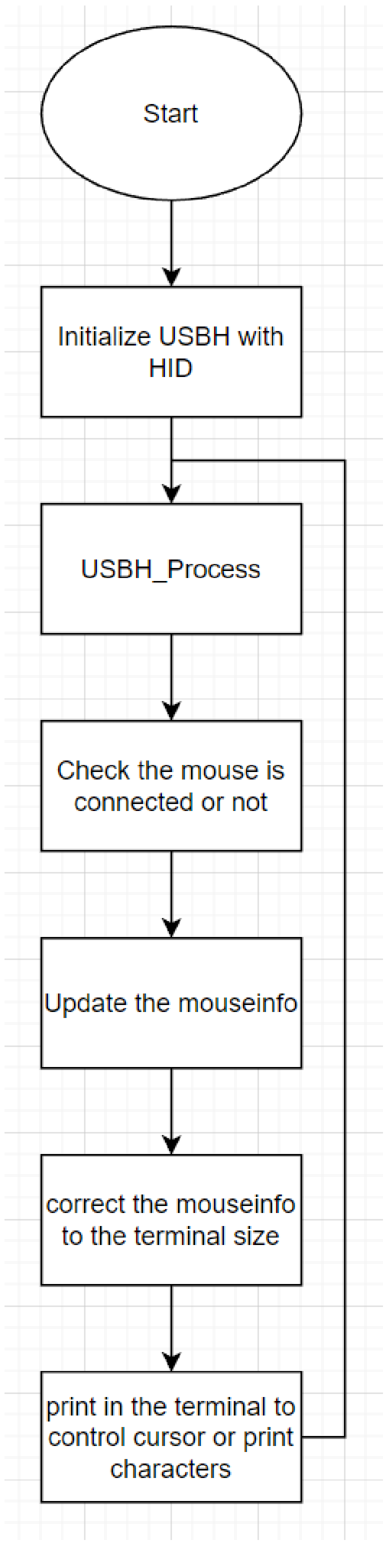
(2) Data frame transfer for high-speed communication equipment;

(3) High-speed data acquisition;

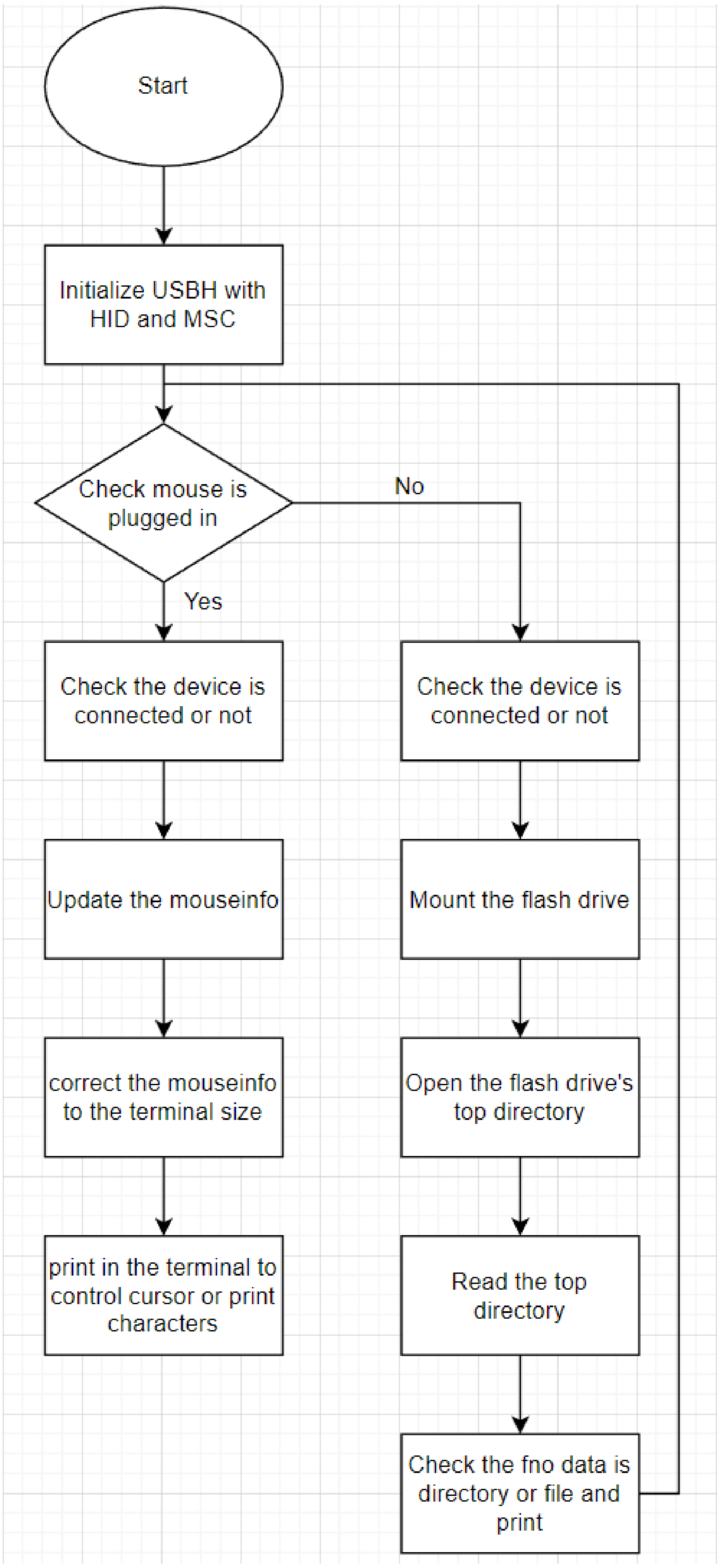
(4) Dynamic memory refresh.

Appendix

[1] Task 1 flowchart.

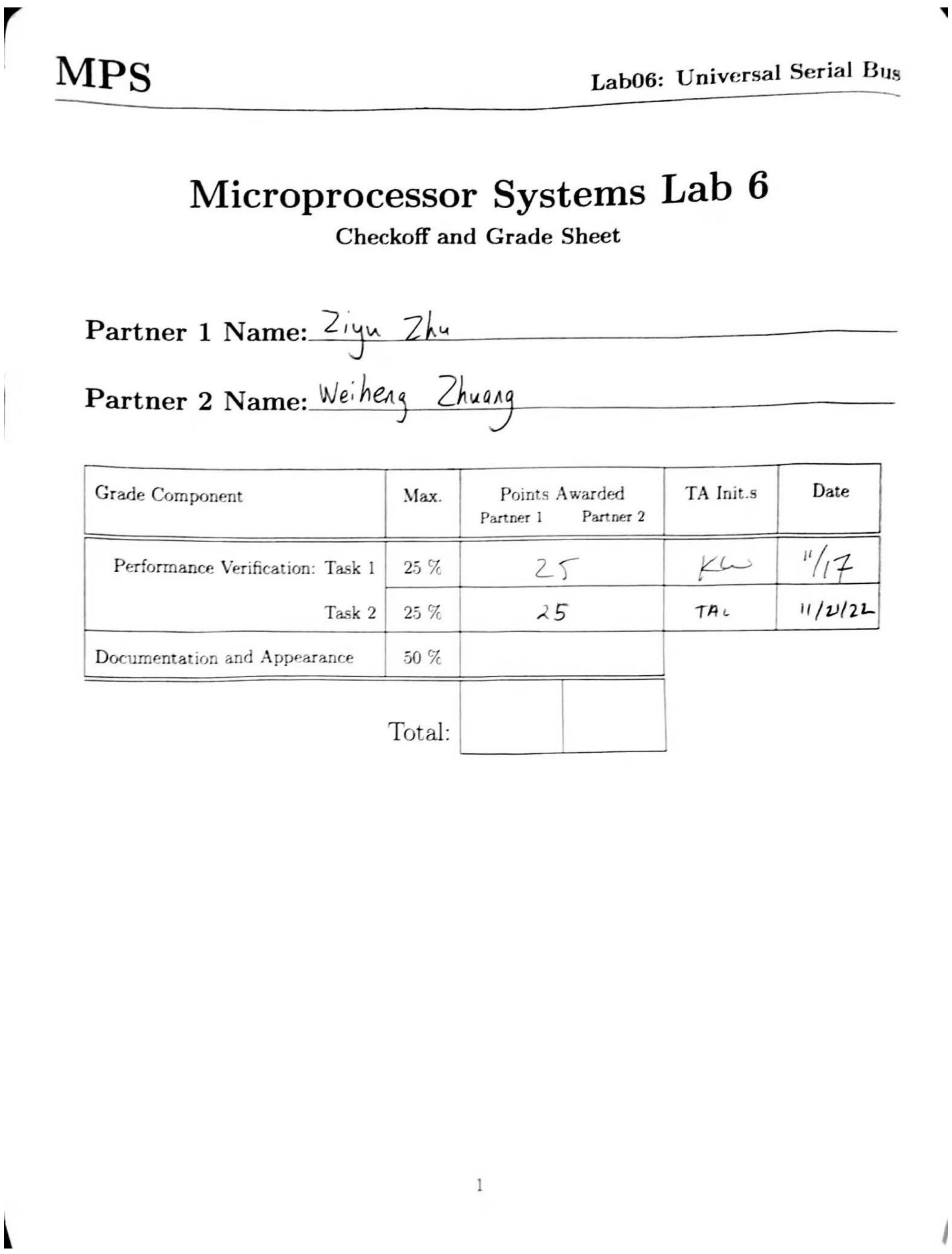

Start
Initialize USBH with HID
USBH_Process
Check the mouse is connected or not
Update the mouseinfo
correct the mouseinfo to the terminal size
print in the terminal to control cursor or print characters

[2] Task 2 flowchart.



Start
Initialize USBH with HID and MSC
Check mouse is plugged in
No
Yes
Check the device is connected or not
Update the mouseinfo
correct the mouseinfo to the terminal size
print in the terminal to control cursor or print characters
Check the device is connected or not
Mount the flash drive
Open the flash drive's top directory
Read the top directory
Check the fno data is directory or file and print

[3] Checkoff and Grade Sheet



Microprocessor Systems Lab 6

Checkoff and Grade Sheet

Partner 1 Name: Ziyu Zhu

Partner 2 Name: Weiheng Zhuang

Grade Component	Max.	Points Awarded		TA Init.s	Date
		Partner 1	Partner 2		
Performance Verification: Task 1	25 %	25		KW	11/17
Task 2	25 %	25		TAL	11/21/22
Documentation and Appearance	50 %				
Total:					

1