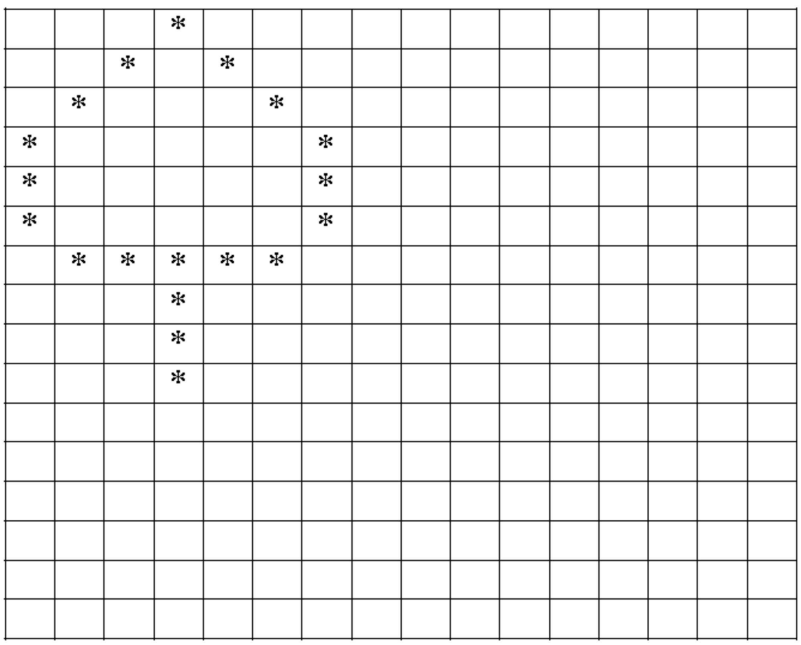
Embedded System Lecture Note 15
Term Project

1. Device drivers

A device driver is a software module that abstracts a physical (hardware) or virtual devices and provides a uniform (and sometimes standard) interface for an operating system to access the peripherals of a system. A **framebuffer** is a video output device that drives a video display from a memory buffer containing a complete frame of data. The information in the buffer typically consists of color values for every pixel (point that can be displayed) on the screen. Color values are commonly stored in 1-bit monochrome, 4-bit palettized, 8-bit palettized, 16-bit highcolor and 24-bit truecolor formats. [http://en.wikipedia.org/wiki/Framebuffer]. Below we simulate the monochrome framebuffer and single pixel represented by an asterisk '*' character. Assume a suitable size for the buffer (say, 256characters X 256 characters). Consider for example a display window (16 X16), try to draw your initials using asterisk or any other character. In your project this frame buffer will be further enhanced.



What to do?

You will implement the project in two phases:

Phase 1: Design and implement a simple user level framebuffer as shell command in order understand the function of a framebuffer. (20%)

Phase 2: Design and implement a framebuffer device driver that will enhance the xinu operating system with a device driver similar to tty driver. (80%)

2. Study Embedded XINU files

Embedded XINU documentation for the version of XINU given with the project assignment. This documentation is internally well linked to allow navigation among related code files. Examine the directory structure and understand the purpose of various directories. Specifically, study these files in the directories:

include (header files): **shell.h, device.h , interrupt.h, tty.h, uart.h**
shell (shell command files): shell.c, xsh_led.c, xsh_help.c
system (system operations): devtable.c, getc.c, putc.c, read.c, write.c, initialize.c, kprintf.c, open.c, close.c, startup.S
all the files in tty and uart directories.

3. Write a framebuffer driver (Phase 2)

The **framebuffer** is a graphic hardware-independent abstraction layer to show graphics on a console without relying on system-specific libraries or the heavy overhead of a Window display system. The framebuffer driver will be implemented as a virtual device on top of uart device driver. The tty driver is indeed a virtual driver that is implemented on top of uart driver. Study tty driver to get an idea of the approach. For the framebuffer device driver suggested steps are as follows:

1. Update **device.h and devtable.c** to include a framebuffer driver.

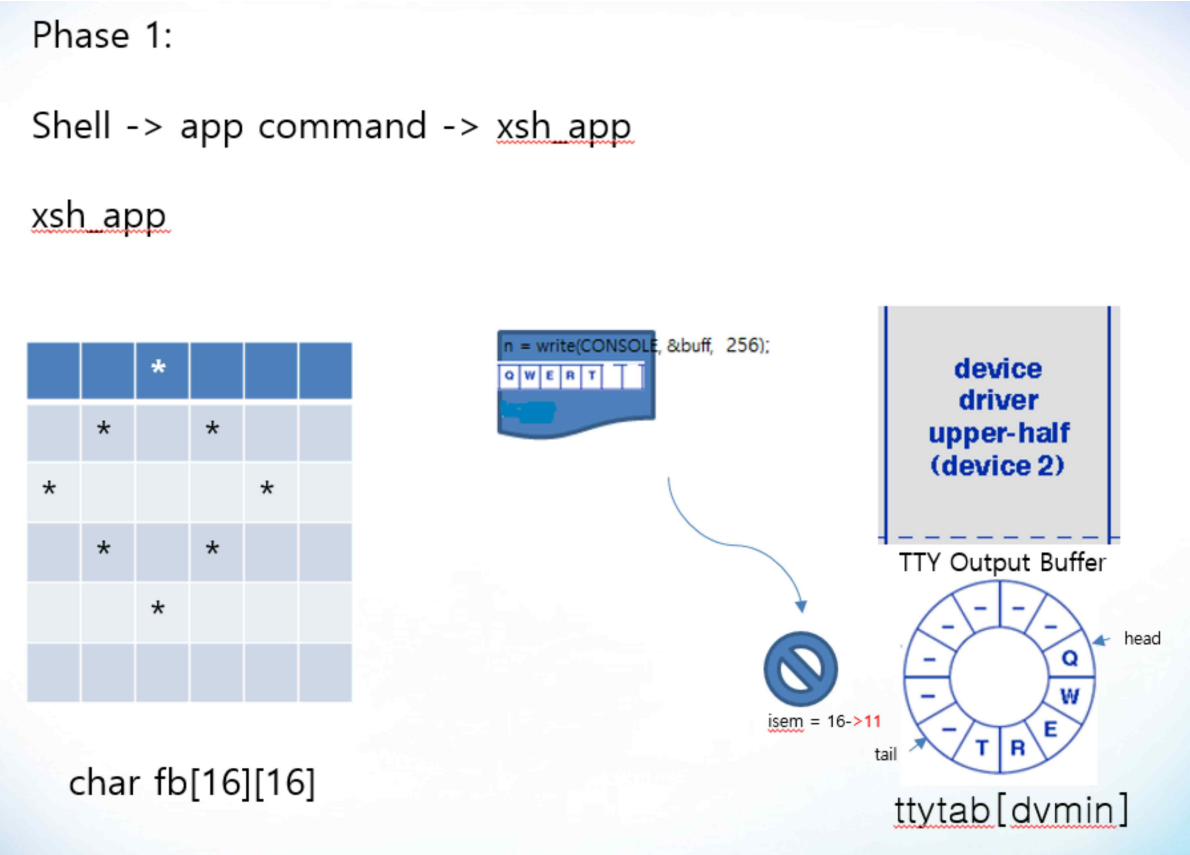
2. Create a **framebuf.h** that specifies the operations of the framebuffer: **init, control, open, close, write (or putChar), other functions needed.**
3. Create a directory **framebuf** and implement all the functions.
4. Update **Makefile** in the **compile** directory to include instructions to compile and link framebuf driver to the xinu boot.
5. Update other files such as the **initialize.c**, if needed.
6. Load the xinu boot and test the operation of the framebuffer driver.

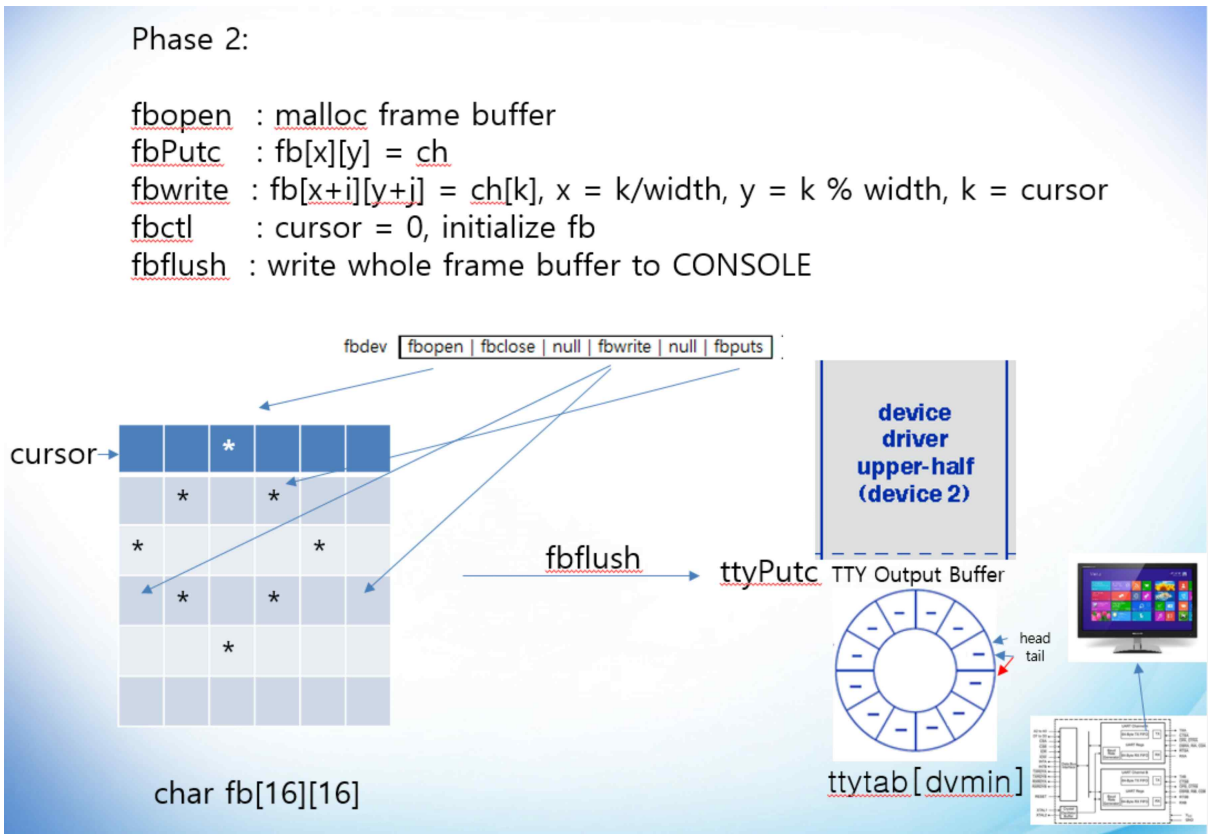
Test it using a shell command that will use this driver. In other words, you ask the shell command to "write" to the screen via the framebuffer.

What to submit?

Zip the Phase 1 files as FBPhase1.zip

Zip the phase 2 files as FBPhase2.zip





Terminal ESC Sequence Examples

A. putchar('₩033'); putchar('['); putchar('0m'); // **Reset All Attributes**

B. putchar('₩033'); putchar('['); putchar('2'); putchar('J'); // **Erase Screen**

C. putchar('₩033'); putchar('['); putchar(';') putchar(H); // **Cursor Home**

APPENDIX

ANSI/VT100 Terminal Control Escape Sequences

https://www2.ccs.neu.edu/research/gpc/VonaUtils/vona/terminal/vtansi.htm

Many computer terminals and terminal emulators support colour and cursor control through a system of escape sequences. One such standard is commonly referred to as ANSI Colour. Several terminal specifications are based on the ANSI colour standard, including VT100.

The following is a partial listing of the VT100 control set.

<ESC> represents the ASCII "escape" character, 0x1B. Bracketed tags represent modifiable decimal parameters; eg. {ROW} would be replaced by a row number.

Terminal Setup

The h and l codes are used for setting terminal/display mode, and vary depending on the implementation. Line Wrap is one of the few setup codes that tend to be used consistently:

Reset Device <ESC>c

- Reset all terminal settings to default.

Enable Line Wrap <ESC>[7h

- Text wraps to next line if longer than the length of the display area.

`Disable Line Wrap` `<ESC>[7l`

- Disables line wrapping.

Cursor Control

`Cursor Home` `<ESC>[{ROW};{COLUMN}H`

- Sets the cursor position where subsequent text will begin. If no row/column parameters are provided (ie. `<ESC>[H`), the cursor will move to the *home* position, at the upper left of the screen.

`Cursor Up` `<ESC>[{COUNT}A`

- Moves the cursor up by *COUNT* rows; the default count is 1.

`Cursor Down` `<ESC>[{COUNT}B`

- Moves the cursor down by *COUNT* rows; the default count is 1.

`Cursor Forward` `<ESC>[{COUNT}C`

- Moves the cursor *forward* by *COUNT* columns; the default count is 1.

`Cursor Backward` `<ESC>[{COUNT}D`

- Moves the cursor *backward* by *COUNT* columns; the default count is 1.

`Force Cursor Position` `<ESC>[{ROW};{COLUMN}f`

- Identical to `Cursor Home`.

`Save Cursor` `<ESC>[s`

- Save current cursor position.

Unsave Cursor <ESC>[u

- Restores cursor position after a Save Cursor.

Save Cursor & Attrs <ESC>7

- Save current cursor position.

Restore Cursor & Attrs <ESC>8

- Restores cursor position after a Save Cursor.

Scrolling

Scroll Screen <ESC>[r

- Enable scrolling for entire display.

Scroll Screen <ESC>[{start};{end}r

- Enable scrolling from row {start} to row {end}.

Scroll Down <ESC>D

- Scroll display down one line.

Scroll Up <ESC>M

- Scroll display up one line.

Tab Control

Set Tab <ESC>H

- Sets a tab at the current position.

Clear Tab <ESC>[g

- Clears tab at the current position.

Clear All Tabs <ESC>[3g

- Clears all tabs.

Erasing Text

Erase End of Line <ESC>[K

- Erases from the current cursor position to the end of the current line.

Erase Start of Line <ESC>[1K

- Erases from the current cursor position to the start of the current line.

Erase Line <ESC>[2K

- Erases the entire current line.

Erase Down <ESC>[J

- Erases the screen from the current line down to the bottom of the screen.

Erase Up <ESC>[1J

- Erases the screen from the current line up to the top of the screen.

Erase Screen <ESC>[2J

- Erases the screen with the background colour and moves the cursor to *home*.

Printing

Some terminals support local printing:

`Print Screen` `<ESC>[i`

- Print the current screen.

`Print Line` `<ESC>[1i`

- Print the current line.

`Stop Print Log` `<ESC>[4i`

- Disable log.

`Start Print Log` `<ESC>[5i`

- Start log; all received text is echoed to a printer.

Define Key

`Set Key Definition` `<ESC>[{key};"{string}"p`

- Associates a *string* of text to a keyboard key. `{key}` indicates the key by its ASCII value in decimal.

Set Display Attributes

Set Attribute Mode <ESC>[{attr1};...;{attrn}m

- Sets multiple display attribute settings. The following lists standard attributes:

- 0 Reset all attributes
- 1 Bright
- 2 Dim
- 4 Underscore
- 5 Blink
- 7 Reverse
- 8 Hidden
-
- Foreground Colours
- 30 Black
- 31 Red
- 32 Green
- 33 Yellow
- 34 Blue
- 35 Magenta
- 36 Cyan
- 37 White
-
- Background Colours
- 40 Black
- 41 Red
- 42 Green
- 43 Yellow
- 44 Blue
- 45 Magenta
- 46 Cyan
  47 White