ECE 391 PS 2

Name: Yu Meng NetID: yumeng5

Name: Shuijing Liu NetID: sliu105

Name: Zhikai Guo NetID: zguo16

Name: Jin Yan NetID: jinyan5

**Problem 1.**

a.

VGA is capable of dividing the screen horizontally into two windows with different display memory address.

* Register setting:

Pixel panning node (bit 5 on attribute node) = 1

The Line Compare field in the VGA, of which bit 9 is in the Maximum Scan Line Register, bit 8 is in the Overflow Register, and bits 7-0 are in the Line Compare Register, specifies the scan line address of the horizontal division.

For bottom window, preset row scan field is always 0 in preset row scan register.

In order to allow the upper window to pan while the lower one remains fixed, we need to set the Pixel Panning field to 1, which is bit 5 of the Attribute Mode Control Register. When a successful line compare occurs, Pixel Shift Count and Byte Panning fields are reset to 0 for the rest of display cycle.

* How VGA will act upon:

The value of preset row scan field specifies how many scan lines to scroll the display upwards. Valid values range from 0 to the value of the Maximum Scan Line field. If we set value to 0, this means we want a non-scrolling status bar.

* Relevant Constraint that must be obeyed:

Bottom window’s display memory address is starting at 0 and is a fixed convention, which implies that bottom screen must be located first in the memory and then locate the top screen. There are two cases. Either both top and bottom windows panning value are the same amount, or when top window pans, bottom window panning value remains at 0. In addition, the bottom window has a Preset Row Scan value of 0, and preset row scan field can only apply to top window.

b.

In order to write the VGA, first we need to deliver the value of palette entry index to PEL Address Write Mode Register, port 3C8h, and output the entry index to DAC Read address filed for read operations. Then, we need to output red, green, blue (in that order) component value in the PEL data register, port 3C9h. The internal write address updates automatically, allowing the next RGB values to be loaded without programming the DAC address write mode register. The entire palette can be loaded in a single write operation.

outp(0x03c8, index);

outp(0x03c9, red);

outp(0x03c9, green);

outp(0x03c9, blue);

**Problem 2**

a.

MTCP\_BIOC\_ON: The message is sent when the button interrupt-on-change is to be enabled. This command will enable the button interrupt-on-change mode. MTCP\_ACK is returned.

MTCP\_LED\_SET: The message should be sent when user is trying to set LED display values. It sets the user-set LED display values. These will be displayed on the LED displays when the LED display is in USR mode. The first byte of argument specifies which of the LED’s to set, and also determines how many bytes will follow – one byte for each LED to set. MTCP\_ACK is returned.

b.

MTCP\_ACK: It is sent when the MTC successfully completes a command. It serves as a return value when messages such as MTCP\_BIOC\_ON, MTCP\_BIOC\_OFF, and MTCP\_DBG\_OFF is sent.

MTCP\_BIOC\_EVENT: It is generated when the Button Interrupt-on-change mode is enabled and a button is either pressed or released. The packet contains information corresponding to C, B, A and START on the LED display (Byte 1) and right, down, left and up buttons (Byte 2).

MTCP\_RESET: It is generated when the device re-initializes itself after a power-up, a RESET button press, or an MTCP\_RESET\_DEV command. Byte 0 is the message MTCP\_RESET; byte 1 and byte 2 are reserved.

c.

The function which calls it (i.e. tuxctl\_ldisc\_data\_callback) is called from an interrupt context, thus it cannot wait or take too much time.

**Problem 3**

#define BOY 0

#define GIRL 1

typedef struct {

spinlock\_t lock;

int count[2];

} boygirl\_lock\_t;

void boygirl\_lock\_init(boygirl\_lock\_t \*lock) {

lock->lock = SPIN\_LOCK\_UNLOCKED;

lock->count[BOY] = 0;

lock->count[GIRL] = 0;

return;

}

void boygirl\_lock(boygirl\_lock\_t \*lock, int gender) {

spin\_lock (&(lock->lock));

while (lock->count[1-gender]) {

spin\_unlock (&(lock->lock));

spin\_lock (&(lock->lock));

}

lock->count[gender] ++;

spin\_unlock (&(lock->lock));

return;

}

void boygirl\_unlock(boygirl\_lock\_t \*lock, int gender) {

spin\_lock (&(lock->lock));

lock->count[gender] --;

spin\_unlock (&(lock->lock));

return;

}