COMP 2659 Course Project – Stage 8: Miscellaneous Revisions

Released: Wednesday, March 13, 2019
Target Completion Date: Friday, March 20, 2019 (at the latest)

Overview

This stage adds very little new functionality to the game. Instead, it provides an opportunity to review and complete the core features of the 1-player game prior to advancing to stage 9. The only new functionality added in this stage is a "splash screen"-style main menu. Also, reliance on the operating system will continue to be reduced.

Review and Completion of All Stage 1-7 Deliverables

Before continuing, it is **critical** that you take this opportunity to review your work in all previous stages. Ensure that all requirements described in those stages are met. This includes ensuring each of the following:

- that all aspects of the graphics are complete, including flicker/tear-free double buffered animation;
- that the rendering is efficient and the animation is smooth, at or approaching 70fps;
- that all aspects of the music and sound effects are complete;
- that the core 1-player game play and functionality are finished, from the user's perspective;
- that the game specification document from stage 1 is in-sync with the game itself (update the spec. and/or the game as necessary);
- that the code is well designed and clearly written, and conforms to all the code and documentation standards described in earlier stages.

Stages 9 and onward assume that all of the above are finished. Due to the nature of the remaining stages, there will be very limited opportunity to add new functionality or correct earlier design/coding deficiencies.

Background: Atari ST Video Hardware Summary (Monochrome)

As discussed previously, the frame buffer (FB) is a 16,000 word region of RAM. It holds the bitmap for the video image to display on the monitor's screen. The ST's video hardware repeatedly indexes through FB, word-by-word. The SHIFTER chip serializes each successive word to produce a video intensity signal, which is output on a pin of the video port.

Actually, the SHIFTER does not index memory directly. Another chip, called the MMU (Memory Management Unit), is responsible for addressing RAM. Each indexed word is placed on the data bus by RAM, and is then latched by the SHIFTER.

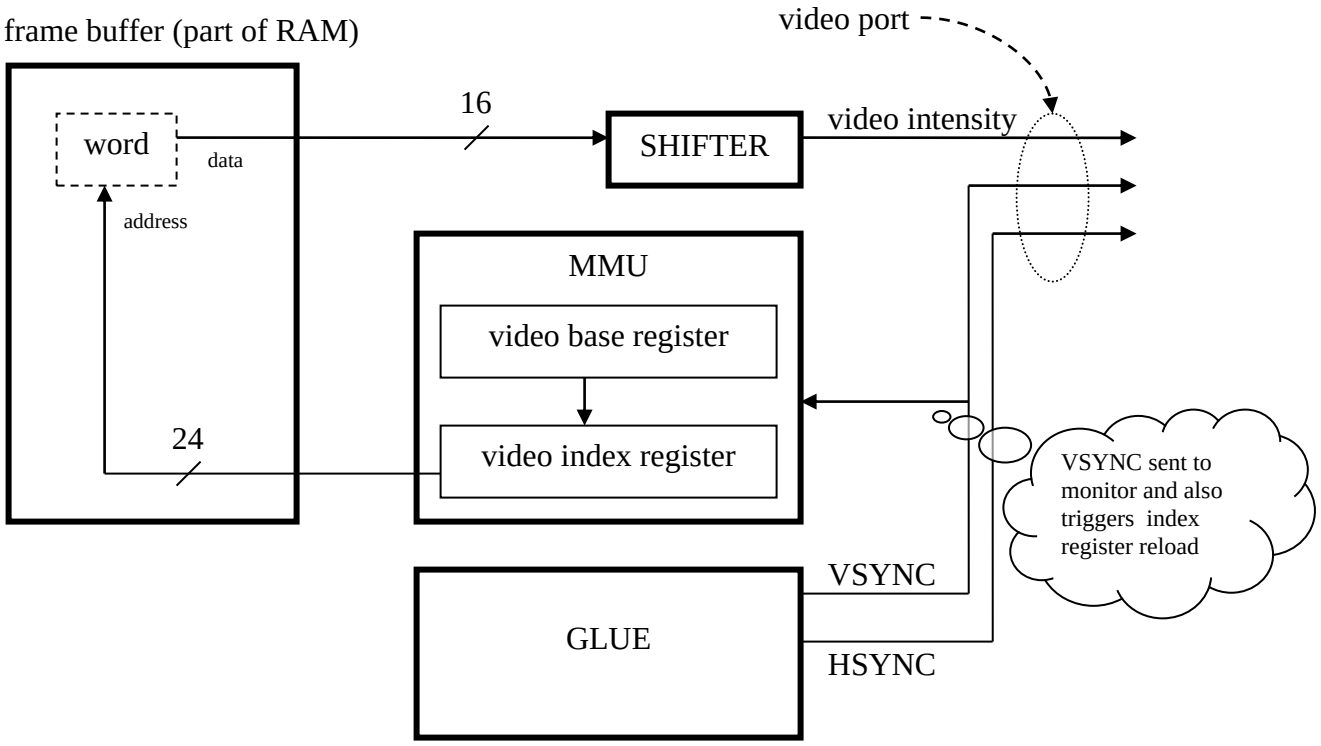
Yet another chip, the GLUE, is responsible for generating the horizontal and vertical synchronization signals (HSYNC and VSYNC) for the monitor. They specify the timing of horizontal and vertical scans. These two signals are output on the video port along with the video intensity signal. The entire screen is refreshed at a rate of 70 Hz.

Obviously, the MMU maintains a "video index register" to keep track of the current FB word to address. This 24-bit counter is periodically incremented by 2, so that it steps through the addresses of FB memory. The MMU also maintains a "video base register". This holds a pointer to the start address of the FB. At

the end of each pass through the FB, the index register is reloaded with the contents of the base register in preparation for the next screen refresh.

Important: it is by altering the contents of the video base register that the frame buffer start address can be changed. This is how page flipping is accomplished when performing double buffered graphics. **Note carefully:** when a new value is written into the base register, the value isn't loaded into the video index register until the next VSYNC! So, the page flip does not occur immediately. It occurs some time within the next 1/70th of a second.

The diagram below summarizes this arrangement.



One final comment for the curious: on the ST, the MMU handles *all* RAM accesses. If the 68000 places a valid RAM address on the address bus, the MMU forwards this address along to RAM. If the SHIFTER needs another word of data, again, the MMU addresses RAM at the next video index. The latter happens repeatedly, without the CPU's involvement. If the CPU and the SHIFTER both require memory to be accessed at roughly the same time, the MMU alternates memory access cycles between them.

<u>Requirement: Removal of More TOS System Calls</u>

By the end of this stage, the game code must not rely in any way on TOS, with the exception of the following:

- vertical blank counter (longword at address 0x462) ← will be removed in the next stage
- `Cconis` and `Cnecin` system calls ← will be removed in the next stage
- `Super` system call ← will be removed in the last stage

The above implies that the code does not issue any system calls other than the three above, nor does it rely on any operating system variables/tables other than the one above. The TOS font table and key table are examples of the latter, and are therefore strictly disallowed (unless you've "ripped" your own copies).

The above implies that any library functions which cause TOS system calls to be invoked indirectly are also forbidden. In particular, you may not use I/O functions such as `printf` or `getchar`. You must also avoid `malloc` and `free`, since they may occasionally need to interact with the operating system. The implementations of `srand` and `rand` do not invoke the operating system, so their usage is safe. Similarly, string manipulation functions such as `strcpy` and `strcmp` are safe. If in doubt about a particular function, seek guidance from your professor before continuing to use it.

In this stage, you must remove the `Physbase` and `Setscreen` system calls, and replace them with your own implementations.

Write your own routine for determining the current frame buffer start address:

```
UINT16 *get_video_base();
```

Replace the call to `Physbase` with a call to your own function.

If you consult a memory map, you will discover that there is a 16-bit video base register (spread over two 8-bit addresses) which holds the frame buffer start address. These two bytes represent the high and middle bytes of the 24-bit video base. The low 8 bits are always zero, and are not stored[1].

Once the above is working, write your own routine for changing the frame buffer start address:

```
void set_video_base(UINT16 *);
```

Replace the call to `Setscreen` with a call to your own function.

Place these two functions in the `raster` module.

Implementing these two functions in C does not require much code. However, to be fully correct, the latter function must really be implemented in 68000 assembly language using the `movep` instruction. This is so the two bytes of the 16-bit video register are updated at the same time. Not doing so can trigger a bug if VSYNC happens midway through the `set_video_base` algorithm's execution, at the point when the two 8-bit portions of the video base register are inconsistent.

Requirement: Splash Screen with Main Menu

You must provide a welcome splash screen which provides a main menu. At start-up, this allows the user to select between "1 player" and "quit" options at a minimum. Later, the splash screen will include a "2 player" option. When the game is over, the program may optionally return to this menu.

Display each menu option as a rectangular button. Later, it will be possible to make this menu mouse-driven. At this stage, the keyboard is to be used for menu selection.

[1] This is why frame buffer addresses on the ST must be 256-byte aligned.

The main menu must not be handled by the main game loop. Instead, the program's `main` function must call one function for handling the splash screen, and then a second function for handling the game itself. The latter function must contain the main game loop.

Time permitting, you may also wish to display a background image, high score list, etc. on this screen. But, these are not core features of the project.