GSoC'21 Final Report

AXIOM Remote: Firmware Improvement And Extension

Aman Singh

Mentors: Sebastian, Andrej and Priya

19th August, 2021

Content

Overview

My work involves advancing and extending the firmware of AXIOM Remote. During the GSoC coding period, I have been involved in the tasks mentioned below:

- Adding transition animations in the UI of AXIOM Remote
- Add a text input GUI
- Set up a timer to measure performance of the remote.

Because of some hurdles along the way, I have not been able to finish, but I am currently working on tasks and in the preceding time, I will get them merged.

My organization has supported me and has helped in every way to speed up things. They have sent me AXIOM Remote hardware and helped me set it up. While setting up the hardware, the FT4323H mini-module was not working, so we had to debug and order a new one, which took a lot of time. But after setting the hardware I have been able to finish as much as possible and because of the hardware, I can work on the PIC32 MCU properly and enjoy the process.

In upcoming sections, I have explained my work and the code accompanying it.

Task 1 - Adding Transition Animation

AXIOM remote firmware has various screens as per the UI/UX requirements, it is required to add transition animation, while we switch between different screens.

Progress Pre-GSoC Period

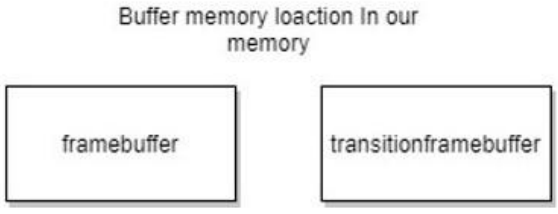
1. Tried an approach based on transition animation from the previous version of AXIOM Remote.
2. Extended the virtual display driver to work with transition animation**.**

Progress During GSoC Period

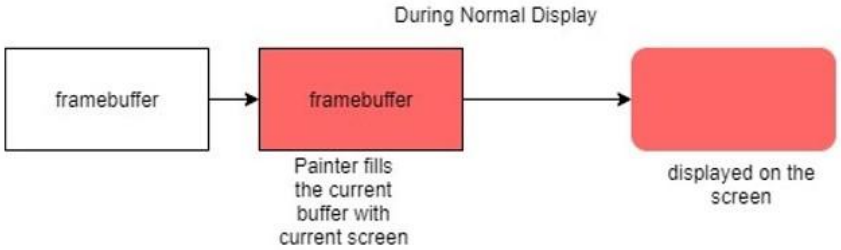
1. Implemented my pre gsoc approach, completely in the firmware and ILI9341 display driver.
2. Tested the approach remotely, and found it to be heavy on CPU.
3. Reorganized and re-structured the approach and changed the basic crux to make it proper.
4. Debugged the version of firmware with transition animation to look at what was causing the crash.
5. Added Push left, Push right, Push up, Push down animation so it works with hardware.

Current Approach

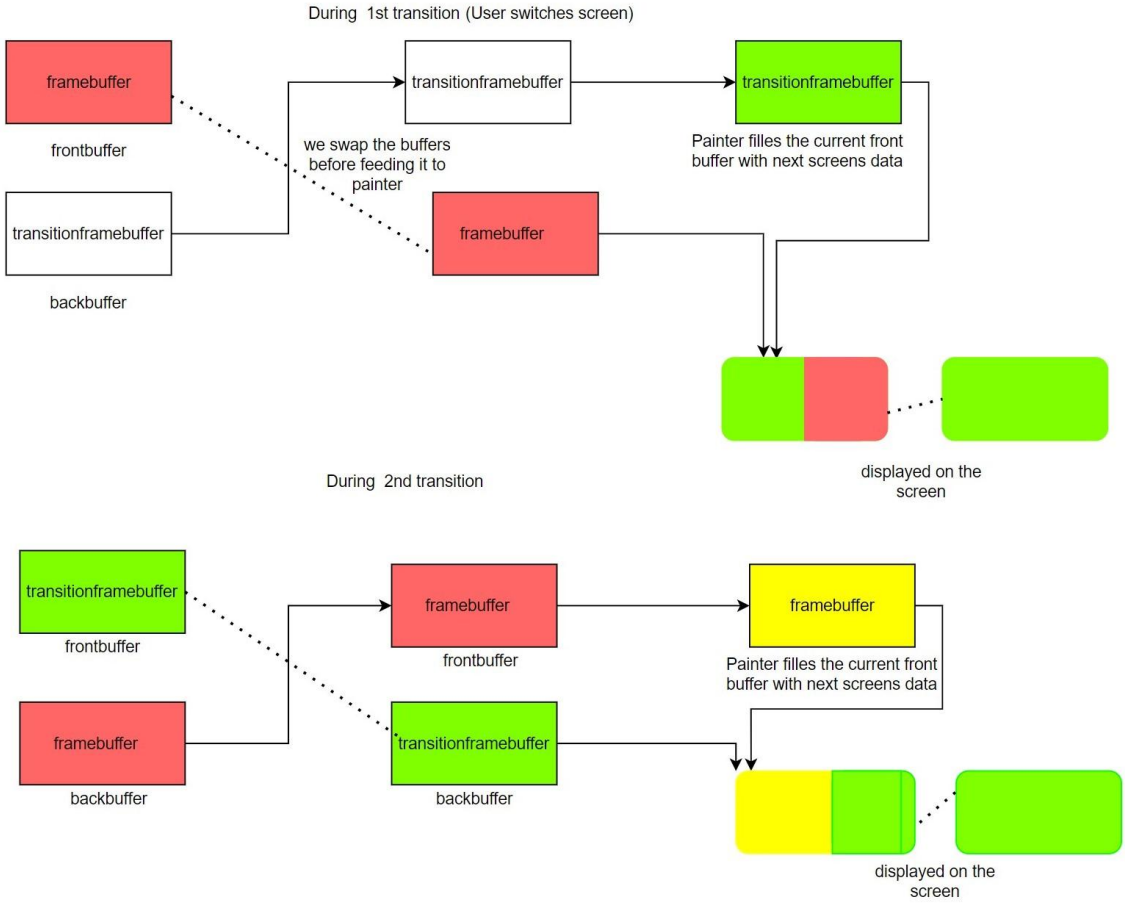
- We have two memory region, named "*framebuffer*" and "*transitionframebuffer*" *representing two framebuffer.*



- *During normal operation, Painter fills a buffer, which is being displayed on screen constantly*



- *During transition, we swap the buffer, so our active buffer becomes our back buffer and back buffer becomes our active buffer, which is filled with a screen that needs to be displayed. Advantage of this, being, we don't have to copy the current content of the buffer to another.*



Modifications in codebase for Various animations - Link
Modifications in codebase needed for buffer swap while changing screen - Link
Link of video of transition animation on AXIOM Remote - Link
Link to complete codebase Repository - Link

Task 2 - Text input GUI

In Axiom Remote we need input from the user, so need a proper input method. I have made a keyboard and widget accompanying it.

Progress During GSoC Period

- completed the text input screen to accomodate a keyboard that is a widget.
- Made a keyboard widget.
- Started writing unit tests for keyboard.
- Modified and debugged the keyboard to work properly in remote hardware.

The text input is through a QWERTY based keyboard layout and can be used to input lower case alphabets, upper case alphabets, numbers and special characters. My previous work was crashing on real hardware. I have updated and debugged it, so it works well in AXIOM Remote hardware.

Working video of Text input GUI on the AXIOM Remote hardware - Link

Screenshots of text input screen

Modifications in Codebase for keyboard Widget - Link
Link to complete codebase Repository - Link

Task 3 - Timer in PIC32, for performance measurement in firmware

It was decided to implement the core timer of PIC32, to accommodate various ways we can measure performance in firmware. After discussion, I had made the modifications, but the same thing could have been achieved easier with a 16 bit timer, with appropriate alterations in Peripheral bus clock 3. Below is how we are using timers and why core timer is not used.

Core timer

Core timer is implemented in the form of two co-processor registers: the Count register, and the Compare register. The Count register is incremented every two system clock (SYSCLK) cycles. The Compare register is used to cause a timer interrupt if desired. An interrupt is generated when the Compare register matches the Count register. An interrupt is taken only if it is enabled in the Interrupt Controller module.

Our SYSCLK frequency in remote - 192 MHz
Core timer uses a 32 bit register (max 4,294,967,295) and counts up at SYSCLK/2, that is our core timer will tick for every 10.41 ns, and will go up to 257.6976 sec. So, for 1 sec timer interrupt, we would need a compare register to be set to 96000000.

Core timer is visible through Coprocessor 0, or CP0.
Register no. 9 - Count Register (Count Processor cycle count)
Register no. 11 - Compare Register (core timer interrupt controller)

We can access the registers in CP0 using coprocessor instructions, such as mfcx (move from coprocessor x register) and mtcx (move to coprocessor x register).

Interrupt Source(1)	XC32 Vector Name	IRQ #	Vector #	Interrupt Bit Location				Persistent Interrupt
				Flag	Enable	Priority	Sub-priority	
Highest Natural Order Priority								
Core Timer Interrupt	_CORE_TIMER_VECTOR	0	OFF000<17:1>	IFS0<0>	IEC0<0>	IPC0<4:2>	IPC0<1:0>	No
Core Software Interrupt 0	_CORE_SOFTWARE_0_VECTOR	1	OFF001<17:1>	IFS0<1>	IEC0<1>	IPC0<12:10>	IPC0<9:8>	No
Core Software Interrupt 1	_CORE_SOFTWARE_1_VECTOR	2	OFF002<17:1>	IFS0<2>	IEC0<2>	IPC0<20:18>	IPC0<17:16>	No
External Interrupt	_EXTERNAL_0_VECTOR	3	OFF003<17:1>	IFS0<3>	IEC0<3>	IPC0<28:26>	IPC0<25:24>	No

The difference between core timer and normal timer module is that, core timer's Count register doesn't reset to 0, after it matches with compare register, so, we have to use delta method i.e.,

difference between different values of count register to measure time taken and for fps measurements we have handle, it in the ISR of the core timer.

So, using just timer 1 would be better, where we do not have such limitations

Using timer 1 (16 bit) to generate 1 sec timer interrupts

Timers in PIC32 are controlled by Peripheral bus clock 3, and our system clock runs at 192hz. We set the P.B clock 3's frequency to SYSCLK freq/12 = 16Mhz, allowing us to get 1 second interrupts with a period value of 62500 (in the range of 16 bit timer) .

Link to Timer function - Link

Task 4 - Documenting the Process of setting up the AXIOM Remote

Organization had sent me AXIOM Remote hardware. I had updated the apertus wiki with documentation about the Process of setting up the AXIOM Remote hardware for development purposes, and flashing the firmware in PIC32 and PIC16s.

Link to AXIOM Remote development notes - Link

Link to changes in development notes in apertus wiki - Link

Post GSoC Plans

At the beginning of January, I had talked about various tasks available with AXIOM Remote, and I wasn't able to do much because I didn't have any prior experience in that field. Since then, I have come a long way. I now understand PIC32 structure and embedded systems. I have AXIOM Remote hardware that speeds up things a lot.

I had many hurdles along the way during the GSoC period, so have not been able to complete many things from my proposal. I will finish all my GSoC tasks properly and get them merged. I will continue working on AXIOM Remote. I would like to be a sub mentor next year, as my mentors have helped me with many things, this was my first ever professional experience, and I have grown a lot.