ECE 561 - Project 4

Milestone 1 – Software Design

“Doodle Jump”

Michael Meli

Thread Information

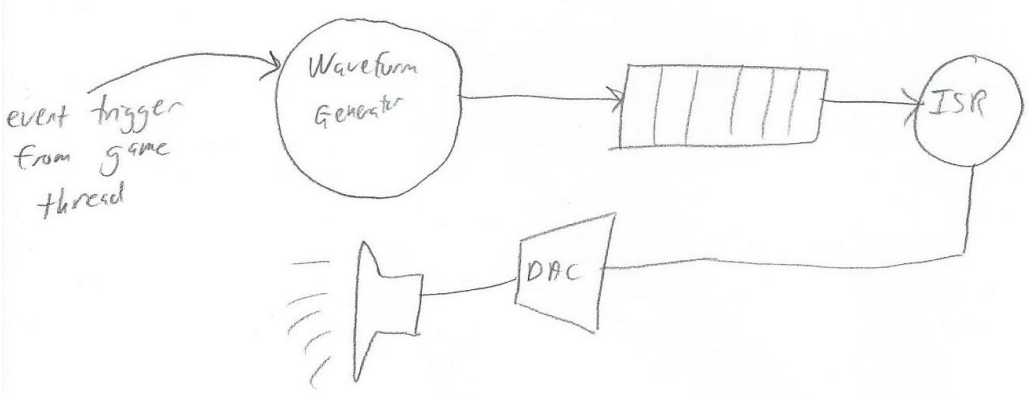
This design will have a total of five threads of differing priority. The threads are, in order of highest to lowest priority, producing sound, reading the accelerometer, reading the touch screen, updating the game state, and displaying system status (561).

Producing Sound

Triggering

The thread that produces sound will be triggered at a set interval as necessary while the sound is being played. Ideally, this thread will be generating a waveform to allow the DMA to copy data into the DAC. The frequency at which this thread needs to be entered will be determined by the buffer size. To avoid entering this thread unnecessarily when no sound is being played, this thread will wait until it is initially triggered by an event from the game thread. Once triggered, it will run normally until it is finished playing the sound.

Top Level Design



Complex/Critical Processing

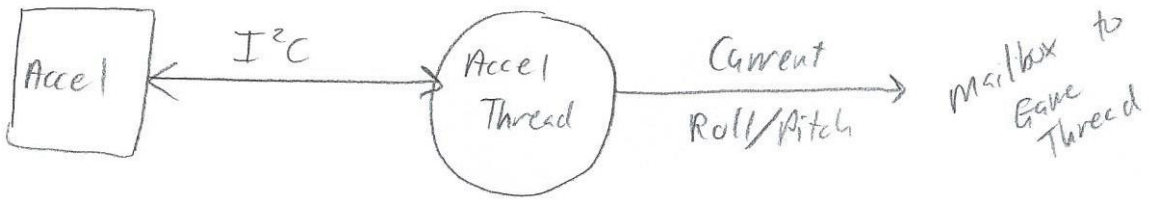
In order to improve performance, it would be wise to replace the ISR feeding the DAC with utilizing the DMA controller to copy data into the DAC asynchronously from the software running. This would be a large improvement and will be focused on.

Reading Accelerometer

Triggering

This thread will be triggered on a set interval throughout the run of the game. This thread needs a higher priority than the game thread to ensure that new roll and pitch data can continually be produced. The roll and pitch data are sent to the game thread via mailbox.

Top Level Design



Complex/Critical Processing

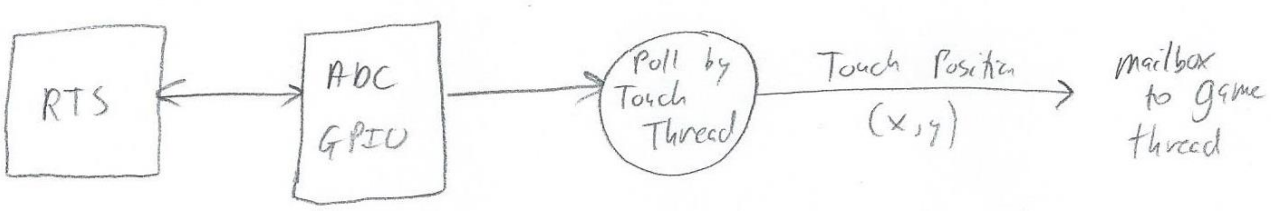
The processor will spend a lot of time in the I2C_WAIT function with this polling implementation of reading the accelerometer. In order to improve performance, interrupt based I2C communications would be desirable. The I2C interrupt will be triggered when it sends or receives information, so the design of this approach would be to track where in the process of communication the processor and accelerometer are so that every time the interrupt is entered, the appropriate action (sending or reading) can be taken. Now, instead of waiting through polling, other actions will be able to be taken.

Reading Touch Screen

Triggering

This thread will be triggered on a set interval throughout the run of the game. This thread needs a higher priority than the game thread to ensure that new touch location information can continually be detected. The position information is sent to the game thread via mailbox.

Top Level Design



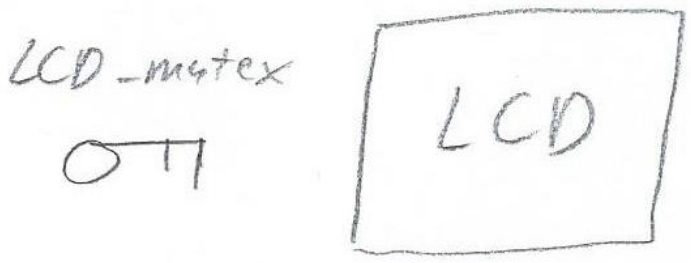
Updating the Game State and Display

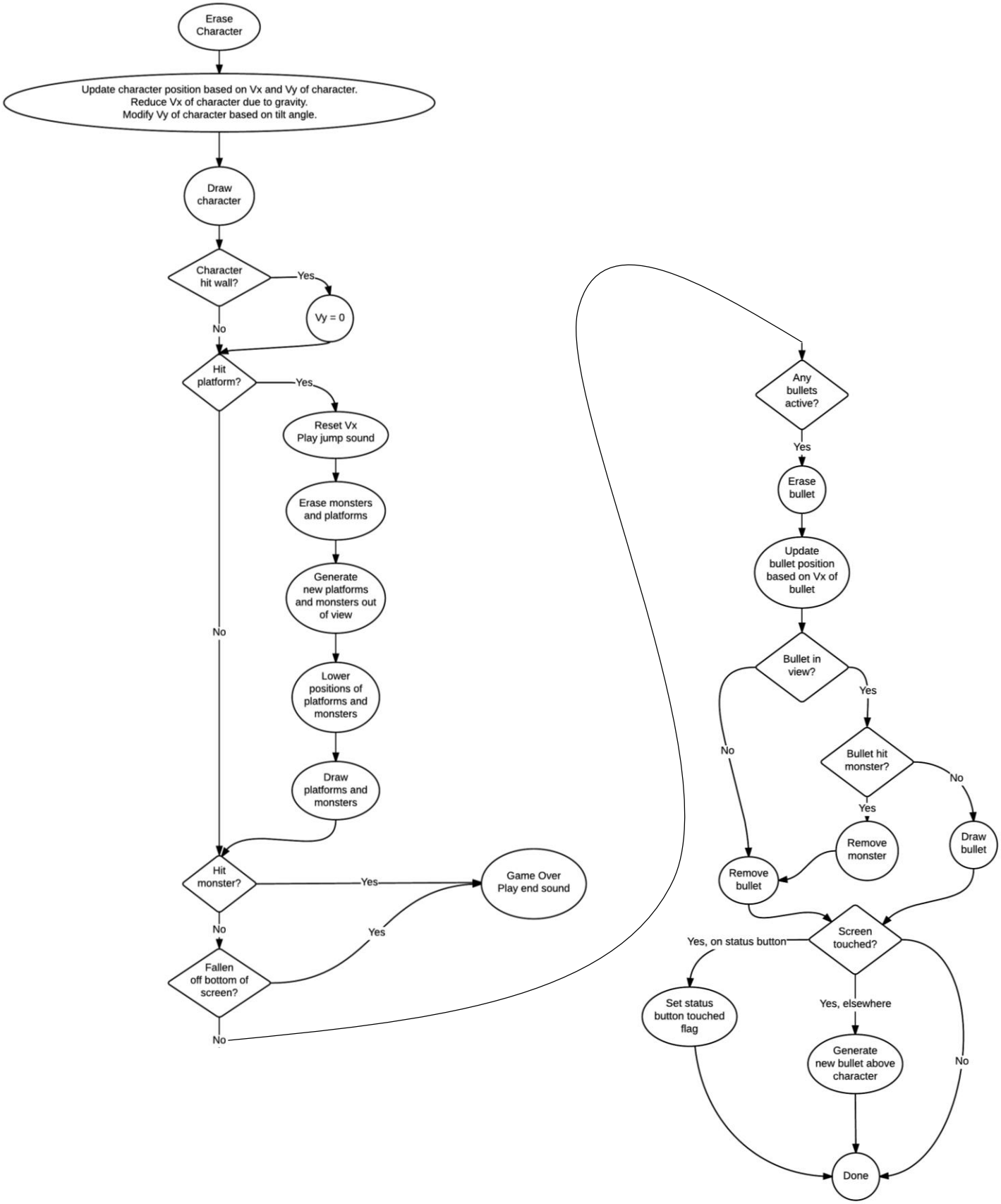
Triggering

This thread will be triggered on a set interval throughout the run of the game. This thread receives roll/pitch and touch screen information via mailbox from the corresponding threads and acts according to the flow chart below. The display is updated at the end of this thread based on the updated game state. A mutex is needed to protect the LCD.

Top Level Design

See the flow chart on the next page for the game state flow.



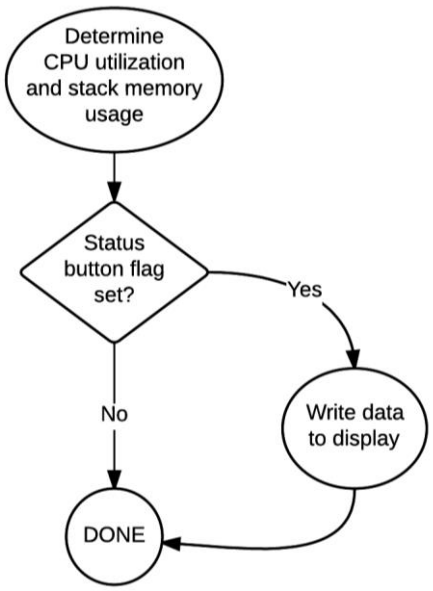

Erase Character
Update character position based on Vx and Vy of character.
Reduce Vx of character due to gravity.
Modify Vy of character based on tilt angle.
Draw character
Character hit wall?
Yes
Vy = 0
No
Hit platform?
Yes
Reset Vx
Play jump sound
Erase monsters and platforms
Generate new platforms and monsters out of view
Lower positions of platforms and monsters
Draw platforms and monsters
No
Hit monster?
Yes
Game Over
Play end sound
No
Fallen off bottom of screen?
Yes
No
Any bullets active?
Yes
Erase bullet
Update bullet position based on Vx of bullet
Bullet in view?
Yes
No
Bullet hit monster?
Yes
No
Remove monster
Draw bullet
Remove bullet
Screen touched?
Yes, on status button
Yes, elsewhere
No
Set status button touched flag
Generate new bullet above character
Done

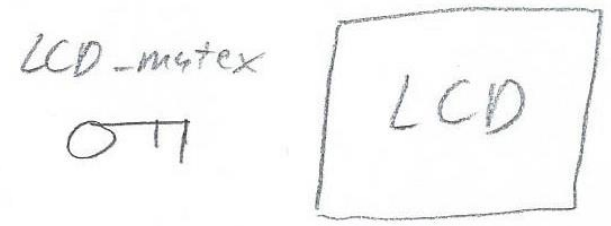
Displaying System Status

Triggering

This thread will be triggered on a set interval to continually check the CPU time utilization and stack memory use. When the option to display this information is selected by clicking on a “Status” button on the screen, the information will be textually displayed on the screen. A mutex is needed to protect the LCD.

Top Level Design





Complex/Critical Processing

Whether or not the “Status” button is pressed will be detected in the game thread for simplicity as that thread will already be checking this information. As a result, a state variable will need to be set to indicate whether the button is pressed. To avoid a race condition, this variable shall be protected by a mutex.

Inter-Thread Communication

The pre-thread information listed out the necessary events, mutexes, and mailboxes that were needed. They shall be repeated here.

Events

- The game thread shall raise an event to trigger the audio thread when a sound needs to begin to play.

Mutexes

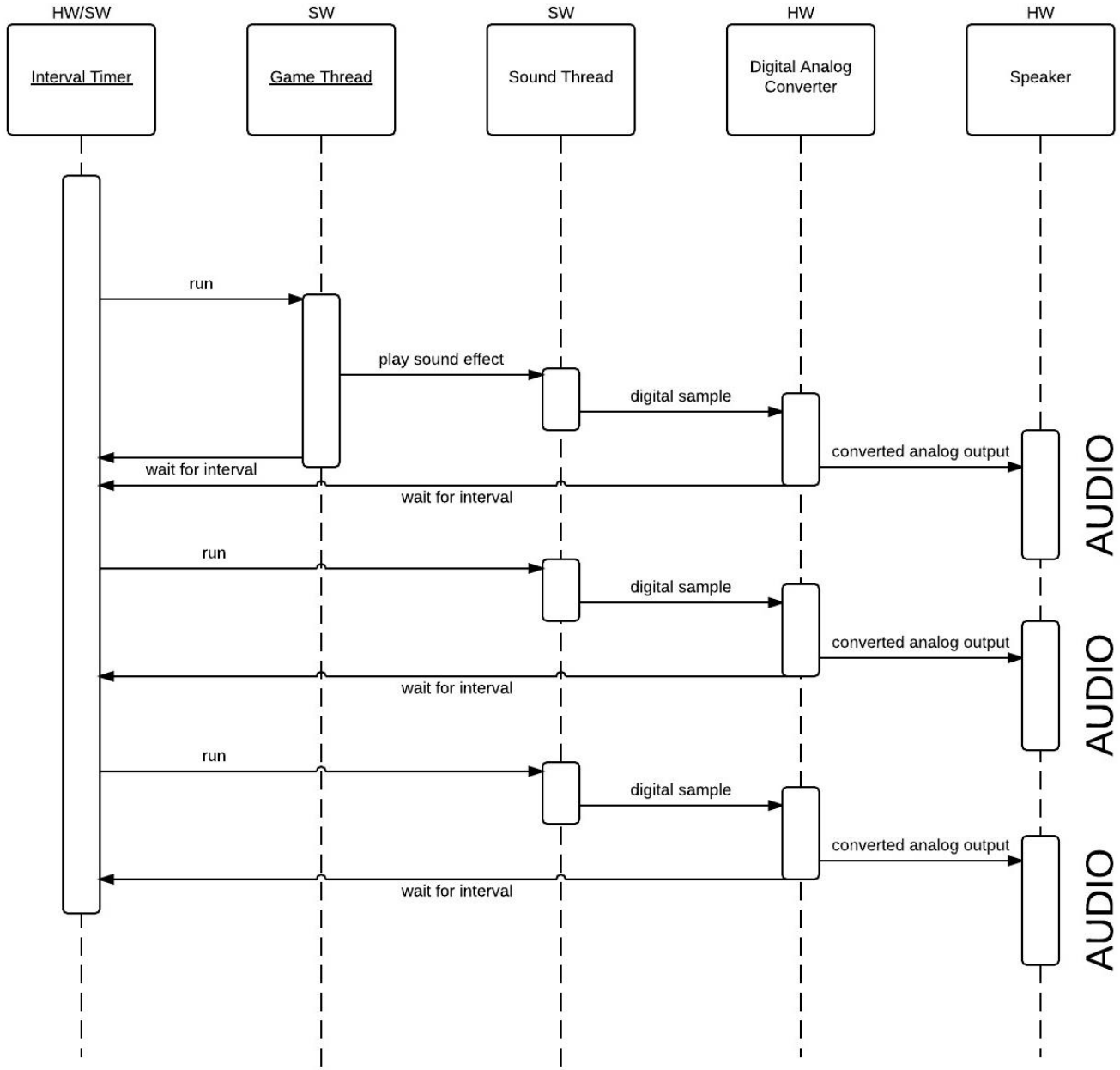
- The LCD needs to be protected by a mutex so multiple threads do not attempt writing to it at the same time.
- The state variable that indicates whether system status is displayed needs to be protected by a mutex. This variable will be written to in the game thread and read in the system status thread.

Mailboxes

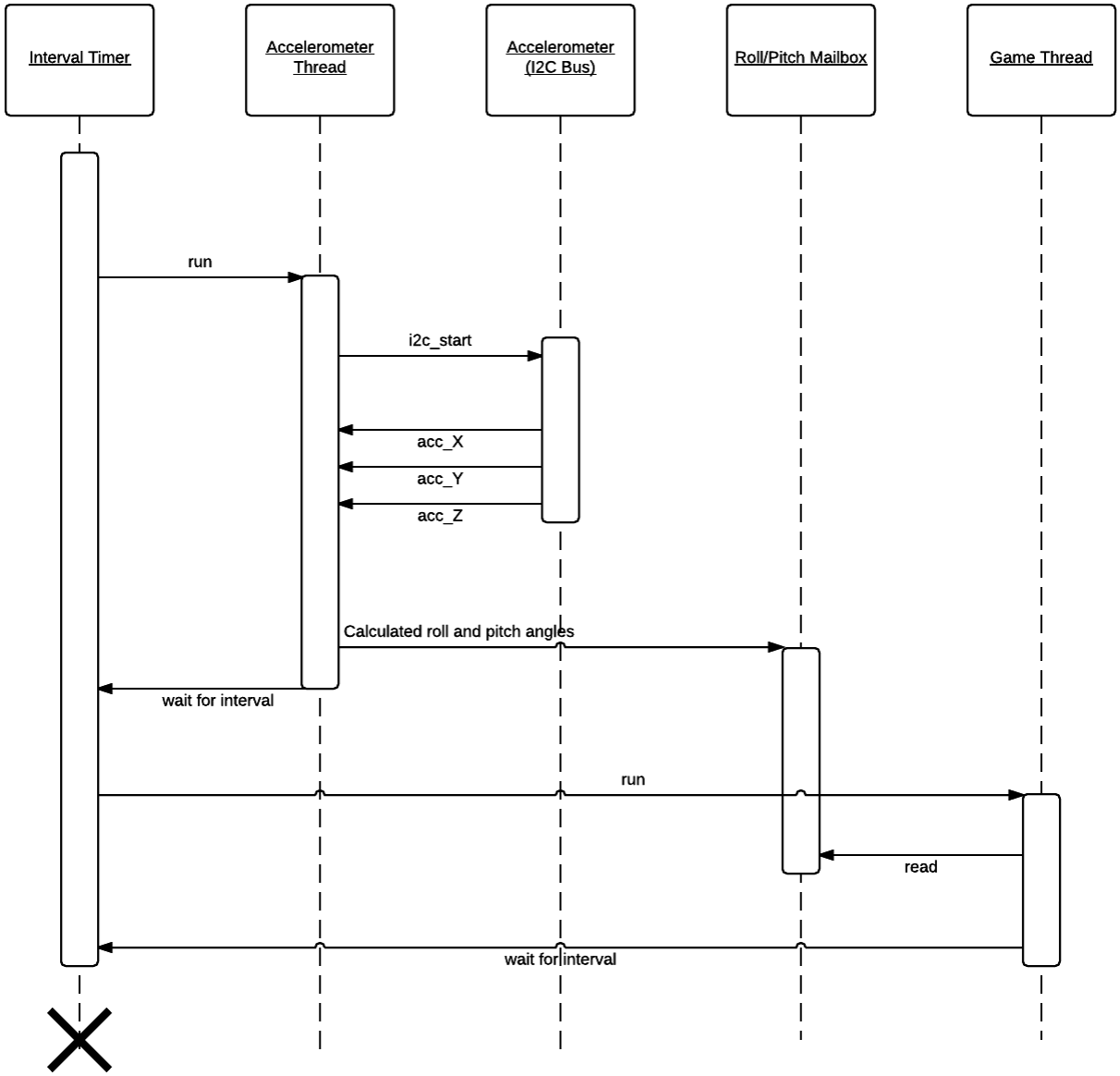
- The accelerometer thread will send roll/pitch data to the game thread via mailbox.
- The touch screen thread will send touch position information to the game thread via mailbox.

Sequence Diagrams

Speaker/Audio



Accelerometer and Roll/Pitch Use



Touch Screen