ECE 561 - Project 4

Milestone 2 – Final Report
"Two-Dimensional Brick Breaker"

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, refilling the sound buffer, managing sound, reading the accelerometer, reading the touch screen, and updating the game state.

Refilling the Sound Buffer

The refilling the sound buffer task's role is to ensure the DMA has access to audio samples for audio playback.

Triggering

The task to refill the sound buffer is triggered by the setting of the event EV_REFILL_SOUND. This event is set in the ISR for the Direct Memory Access (DMA), which is triggered when the buffer feeding the DMA is empty. In other words, this task is run when the DMA needs new samples to be sent to the DAC driving the speaker.

Top Level Design

A sound is stored in the program's ROM as an array of approximately 3500 unsigned 8-bit audio samples. When this sound is playing, the role of this task is simply to sequentially store each sample in the buffer used by the DMA. When the sound is not being played, the task fills the buffer in with silence. The drawing below represents this process.
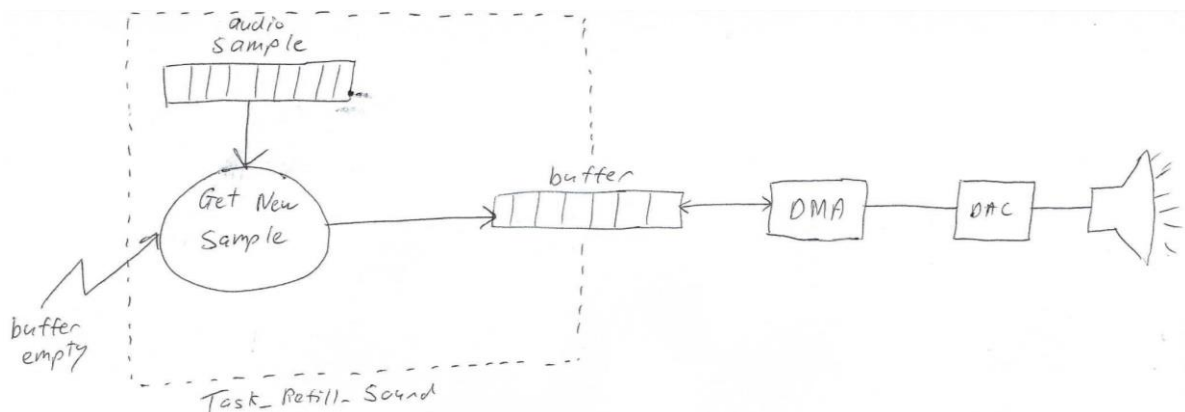
Complex/Critical Processing

This thread has no complex processing as it simply copies data from one array to another. Of course, care must be taken to ensure that a buffer overflow does not occur.
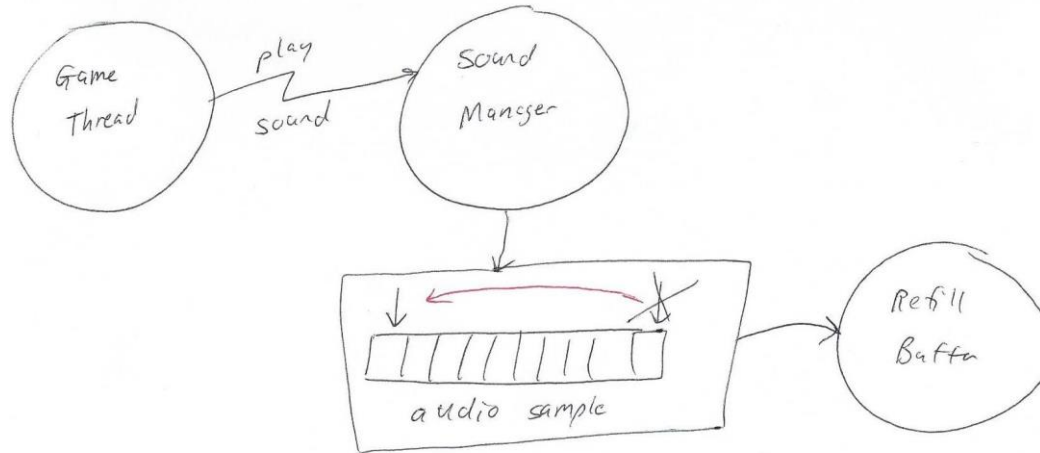
Sound Manager

The sound manager task's role is to determine whether the sound refilling task should be providing the DMA with a sound effect or silence.

Triggering

The sound manager task is triggered by the setting of the event EV_PLAYSOUND. This event is set by the game thread whenever the ball hits a paddle or a brick, which is when the generation of the bounce sound should occur.

Complex/Critical Processing

The sound manager is only called when the sound needs to begin to play. The sound should only play once. Therefore, a variable is used to communicate between this thread and the refill thread. The sound manager resets this variable to zero and this variable is then used as an indexer to access a sample from the sound array. When this variable reaches outside the bounds of the array, it indicates that silence should be sent. Since there is only one writer and one reader on this variable and access of it is atomic, it does not need to be protected by a mutex.

Reading Accelerometer

The reading accelerometer task will read the acceleration data from the accelerometer and convert it to roll and pitch for use by the game thread.

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
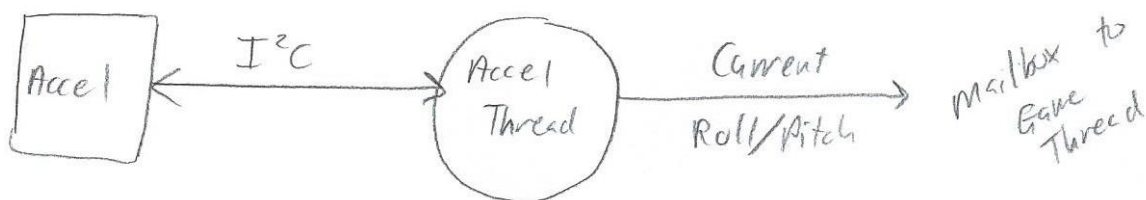
Complex/Critical Processing

The roll and pitch calculated by this thread are used by the game thread to change the positions of the paddles on the screen. Since the accelerometer thread can interrupt the game thread and since these calculations are not atomic, the values of roll and pitch need to be thread safe. To ensure this, a mailbox is used. In the accelerometer thread, once calculated, the roll and pitch are stored in a mailbox where the game thread can then safely read and work on them when ready. The mailbox also allows the

queuing of roll and pitch information, but since the game thread is running much more frequently than this thread, this situation will never practically arise.

Reading Touch Screen

The reading touch screen task will poll the touch screen for new inputs and then send the touch position to the game thread.

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
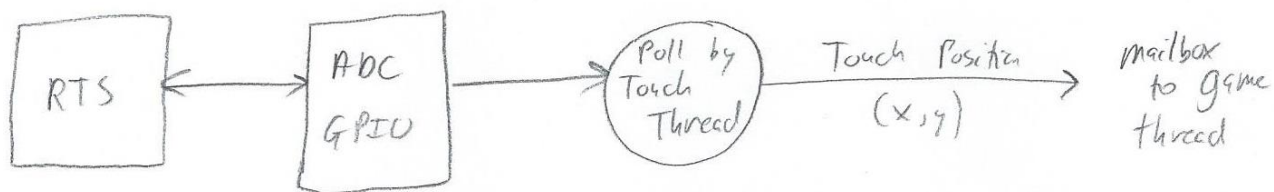
Complex/Critical Processing

The game thread must constantly monitor touch inputs to perform actions such as launching the ball or opening up the ECE 561 required system status menu. Since the touch screen thread can interrupt the game thread, the value of the touch position must be thread safe. To ensure this, a mailbox is used to send the touch position information to the game thread. In addition, since the mailbox is essentially a queue, the game thread will be able to handle multiple quick touches on the screen, even if it cannot handle and process each one immediately, which is a huge advantage.

Updating the Game State and Display

The thread to update the game state and display will handle the actual execution and logic of the game. In addition, if the player has selected to show system status, this thread will also handle pausing the game and displaying such utilization stats on the screen.

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 two pages for the game state flow.

```
                    ┌─────────────────────┐
                    │  Status Menu State? │
                    └─────────────────────┘
                  Open ↙                  ↘ Closed
        ┌─────────────────────────┐
        │  Exit button not pressed │
        └─────────────────────────┘
         True ↙              ↘ False
 ┌──────────────────────┐  ┌──────────────────────┐
 │ Display CPU utilization│  │ Status menu = CLOSED │
 └──────────────────────┘  └──────────────────────┘
           │                          │
 ┌──────────────────────────┐        │
 │ Display current stack depth│       ↓
 └──────────────────────────┘  ┌──────────────────────┐
           │                    │ Erase ball and paddles│
 ┌──────────────────────┐       └──────────────────────┘
 │ Display max stack depth│                │
 └──────────────────────┘          ┌──────────────┐
           │                        │  Brick hit?  │
 ┌──────────────────────┐           └──────────────┘
 │ Check touch mailbox   │        True ↙         ↘ False
 └──────────────────────┘
                    ┌──────────────────────────────┐
                    │ Adjust Vx/Vy to bounce ball back│
                    └──────────────────────────────┘
                                 │
                    ┌──────────────────────┐
                    │ Reduce health of brick│
                    └──────────────────────┘
                                 │
                    ┌──────────────────────┐
                    │   Brick health == 0?  │
                    └──────────────────────┘
                      True ↙       ↘ False
                    ┌──────────────┐
                    │ Remove brick  │
                    └──────────────┘
                                 │
                    ┌──────────────┐
                    │  Last brick?  │
                    └──────────────┘
                 False ↙      ↘ True
 ┌──────────────────────────┐   ┌──────────┐
 │ Redraw brick with new color│   │ VICTORY! │
 └──────────────────────────┘   └──────────┘
                    ┌──────────────────────┐
                    │   Ball hits paddle?   │
                    └──────────────────────┘
       Vertical paddle ↙  Horizontal paddle ↓   ↘ False
         ┌──────────────┐   ┌──────────────┐
         │ Flip ball Vx  │   │ Flip ball Vy  │
         └──────────────┘   └──────────────┘
                    ┌──────────────┐
                    │ Ball hit wall?│
                    └──────────────┘
```

```
                                            True
                              ┌──────────────────────┐
                              ▼                       │
                      ◇ Lives remaining? ◇            │
                 False  │          │ True             │
          ┌─────────────┘          │                  │ False
          ▼                        ▼                  │
     ( GAME OVER )      ┌──────────────────────┐      │
                        │ Flip ball Vx/Vy to   │      │
                        │ bounce it back       │      │
                        └──────────────────────┘      │
                                   │                  │
                                   ▼                  │
                        ┌──────────────────────┐      │
                        │ Reduce number of     │      │
                        │ lives by 1           │      │
                        └──────────────────────┘      │
                                   │                  │
                                   ▼                  ▼
                        ┌──────────────────────────────┐
                        │ Adjust ball position based on │
                        │ Vx and Vy                     │
                        └──────────────────────────────┘
                                        │
                                        ▼
                               ┌──────────────┐
                               │ Redraw ball  │
                               └──────────────┘
                                        │
                                        ▼
                        ◇ New roll and pitch in mailbox? ◇
                             True │          │ False
                                  ▼          │
                        ┌──────────────────┐ │
                        │ Adjust paddle    │ │
                        │ velocities       │ │
                        └──────────────────┘ │
                                  │          ▼
                        ┌──────────────────────────────┐
                        │ Adjust paddle position based  │
                        │ on velocities                 │
                        └──────────────────────────────┘
                                        │
                                        ▼
                               ┌──────────────┐
                               │ Redraw       │
                               │ paddles      │
                               └──────────────┘
                                        │
                                        ▼
                        ◇ New touch position in mailbox? ◇
                             True │                │ False
                                  ▼                │
                        ◇ Touched status button? ◇ │
                     False │          │ True       │
                           │          ▼            │
                           │  ┌──────────────────┐ │
                           │  │ Status menu =    │ │
                           │  │ OPEN             │ │
                           │  └──────────────────┘ │
                           │          │            │
                           ▼          ▼            ▼
                              ( DONE )
```

Complex/Critical Processing

The additional requirement for ECE 561 is to implement a way to see system statistics. In order to determine CPU utilization, the PIT was configured to interrupt every millisecond. A counter for each task was recorded, and when the ISR was raised, the function _isr_get_task() was called to determine which task was interrupted and increment the appropriate counter. To determine stack depth, at each point within the program where stack size would change, the value of the current stack pointer for each task was recorded into an array. This was then subtracted from the location of the bottom of the stack to determine stack depth. Maximum stack depth was simply tracked by finding the largest difference and saving that. All of this information is shown on the system information screen that can be opened by pressing the [STATUS] button in the top left corner of the screen during the game. This functionality was originally going to be built into a separate thread, but it was determined that this process is much simpler as the game thread and system utilization thread would never be running concurrently but needed to share information and communicate.

Inter-Thread Communication

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

Events

- o The game thread shall raise an event to trigger the sound manager when a sound needs to begin to play.
- o The DMA ISR will raise an event to trigger the refill sound buffer task to refill the sound buffer.

Mutexes

- o The LCD needs to be protected by a mutex so multiple threads do not attempt writing to it at the same time.
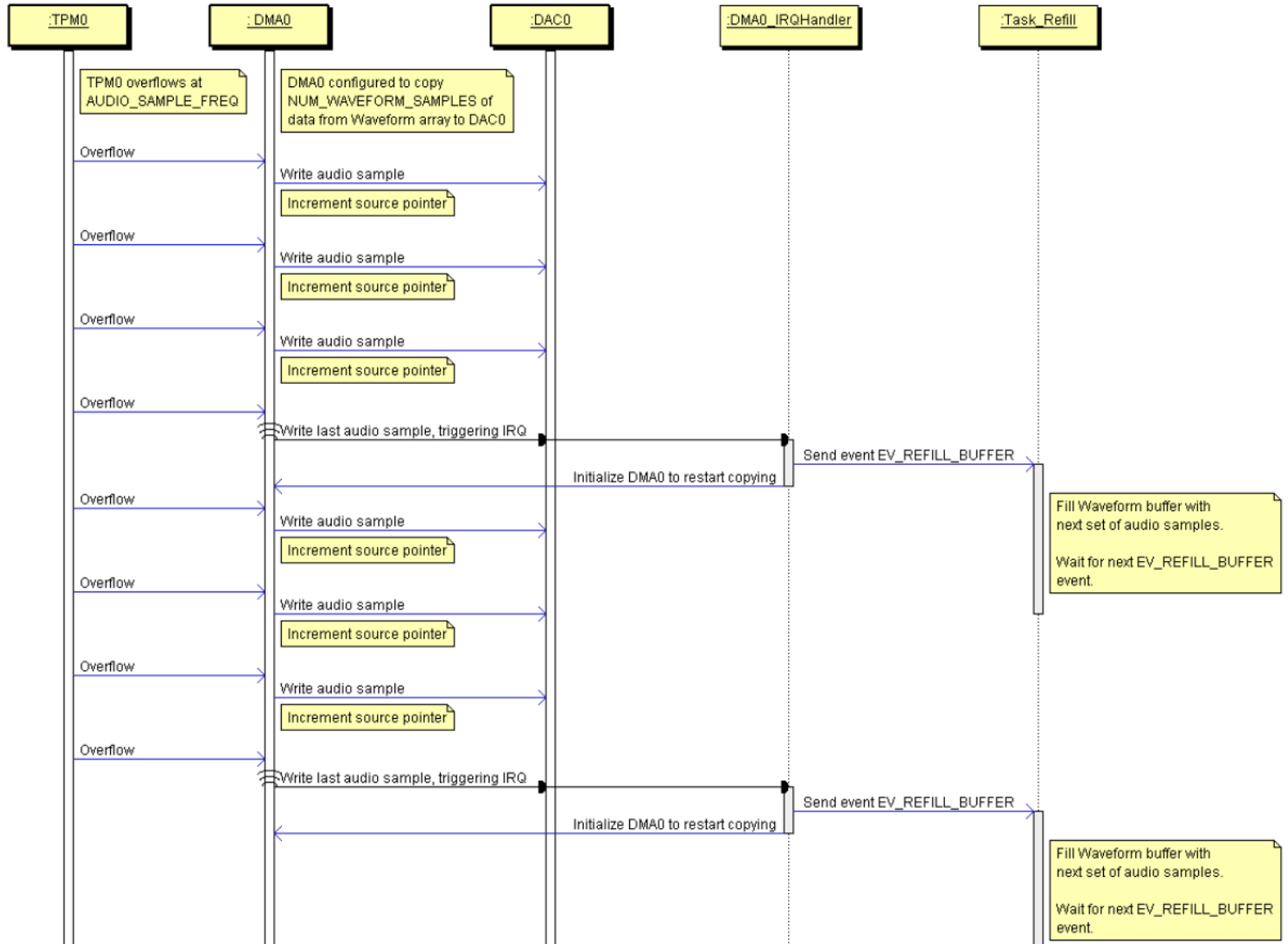
Mailboxes

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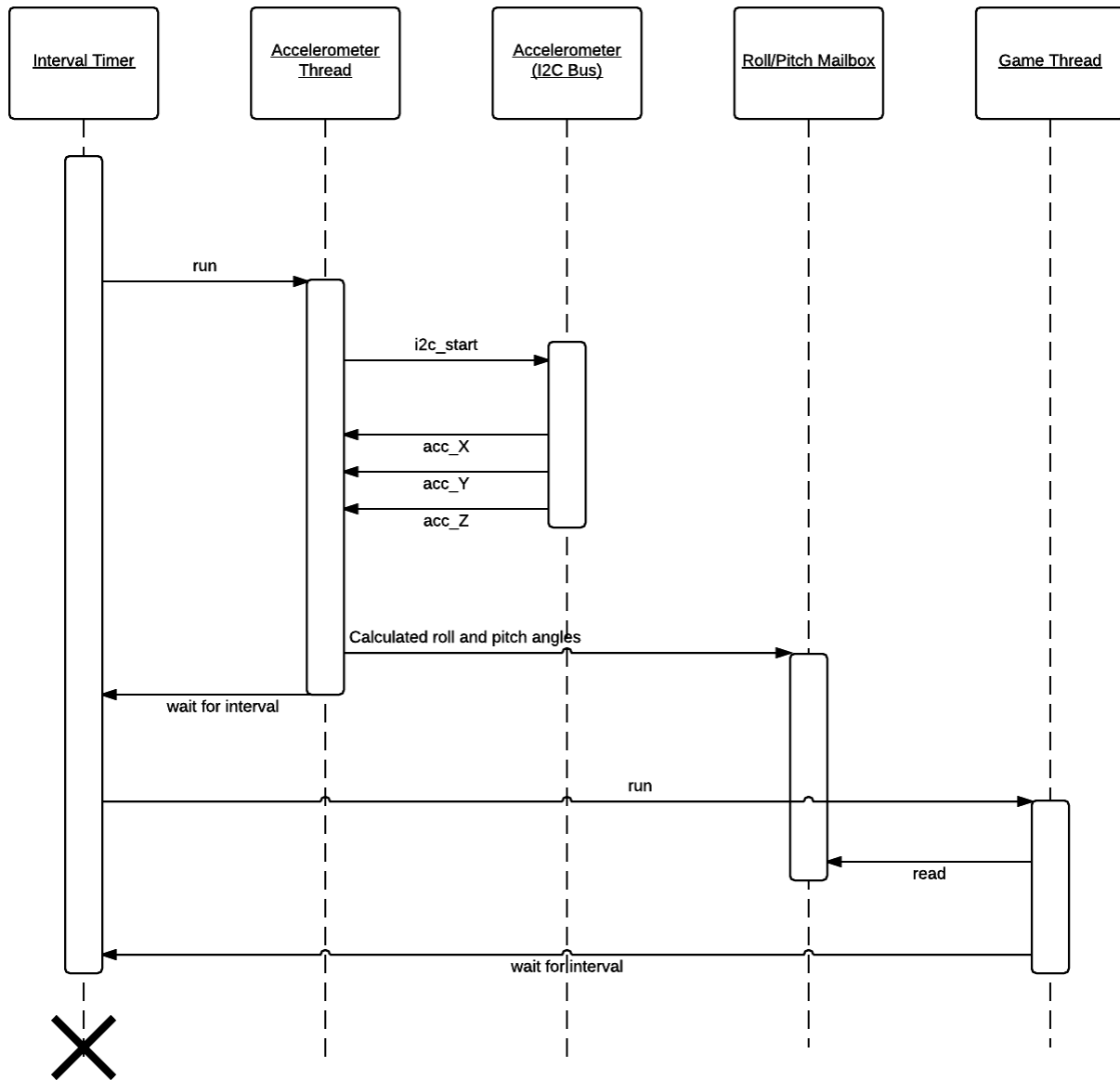
Global Data

- o The profiling system information variables are global as this allows for access to the counters required for profiling from any thread. These operations occur mainly with an ISR which will not be interrupted by any other task, so risk of data integrity violations are essentially non-existant.
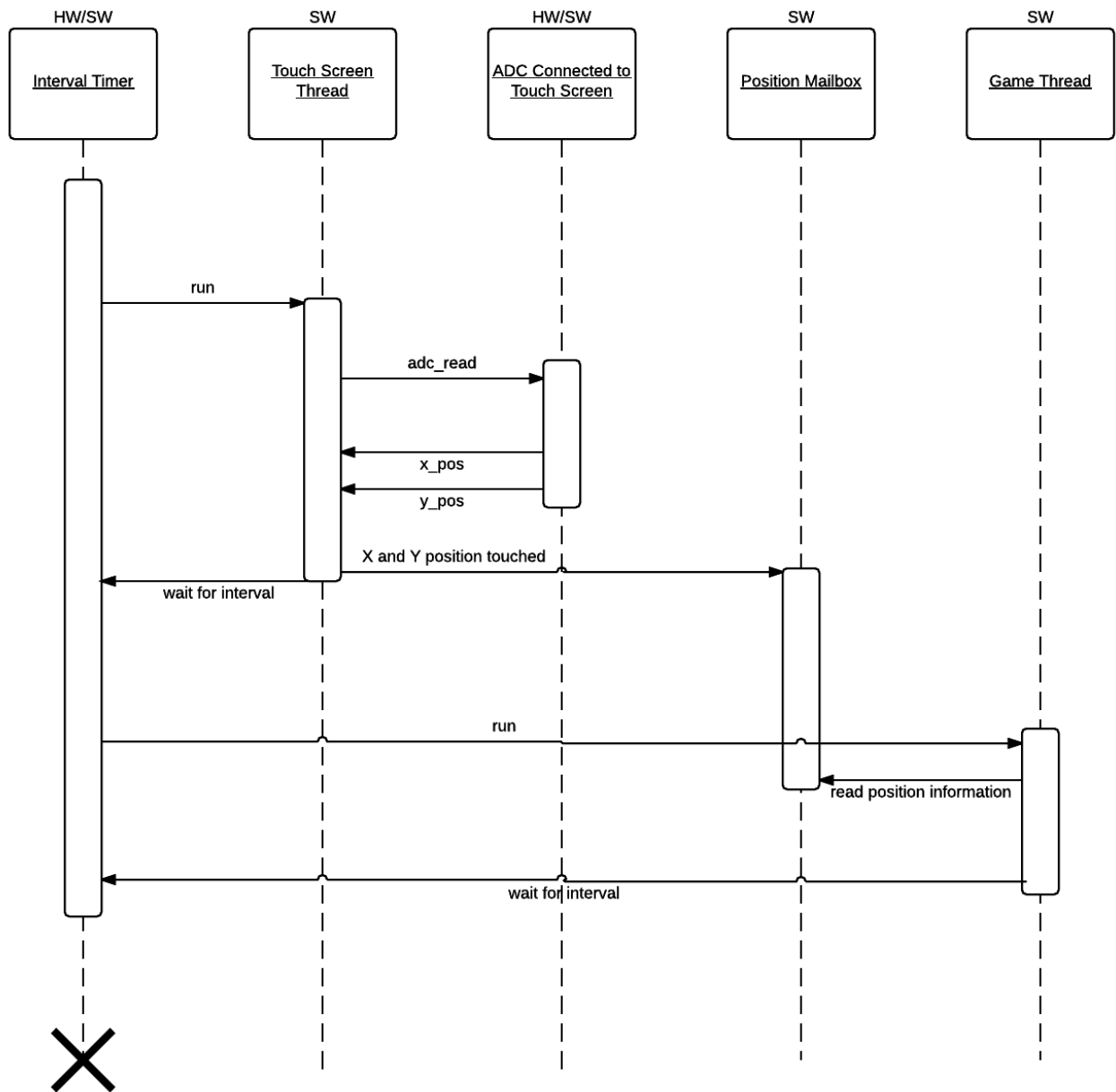
Sequence Diagrams

Speaker/Audio

:TPM0	: DMA0	:DAC0	:DMA0_IRQHandler	:Task_Refill

TPM0 overflows at AUDIO_SAMPLE_FREQ

DMA0 configured to copy NUM_WAVEFORM_SAMPLES of data from Waveform array to DAC0

Overflow

Write audio sample

Increment source pointer

Overflow

Write audio sample

Increment source pointer

Overflow

Write audio sample

Increment source pointer

Overflow

Write last audio sample, triggering IRQ

Send event EV_REFILL_BUFFER

Initialize DMA0 to restart copying

Overflow

Write audio sample

Increment source pointer

Fill Waveform buffer with next set of audio samples.

Wait for next EV_REFILL_BUFFER event.

Overflow

Write audio sample

Increment source pointer

Overflow

Write audio sample

Increment source pointer

Overflow

Write last audio sample, triggering IRQ

Send event EV_REFILL_BUFFER

Initialize DMA0 to restart copying

Fill Waveform buffer with next set of audio samples.

Wait for next EV_REFILL_BUFFER event.

Accelerometer and Roll/Pitch Use

Interval Timer	Accelerometer Thread	Accelerometer (I2C Bus)	Roll/Pitch Mailbox	Game Thread

run

i2c_start

acc_X

acc_Y

acc_Z

Calculated roll and pitch angles

wait for interval

run

read

wait for interval

Touch Screen

HW/SW	SW	HW/SW	SW	SW
Interval Timer	Touch Screen Thread	ADC Connected to Touch Screen	Position Mailbox	Game Thread

run

adc_read

x_pos

y_pos

X and Y position touched

wait for interval

run

read position information

wait for interval

CPU Utilization Tracking

HW/	SW	SW	SW
Periodic Interrupt Timer (PIT)	2D Brick Breaker Game	PIT ISR	Tick Counts

Timer Expires Every 1 ms

Increment Counter for Current Task

Timer Expires Every 1 ms

Increment Counter for Current Task

Timer Expires Every 1 ms

Increment Counter for Current Task
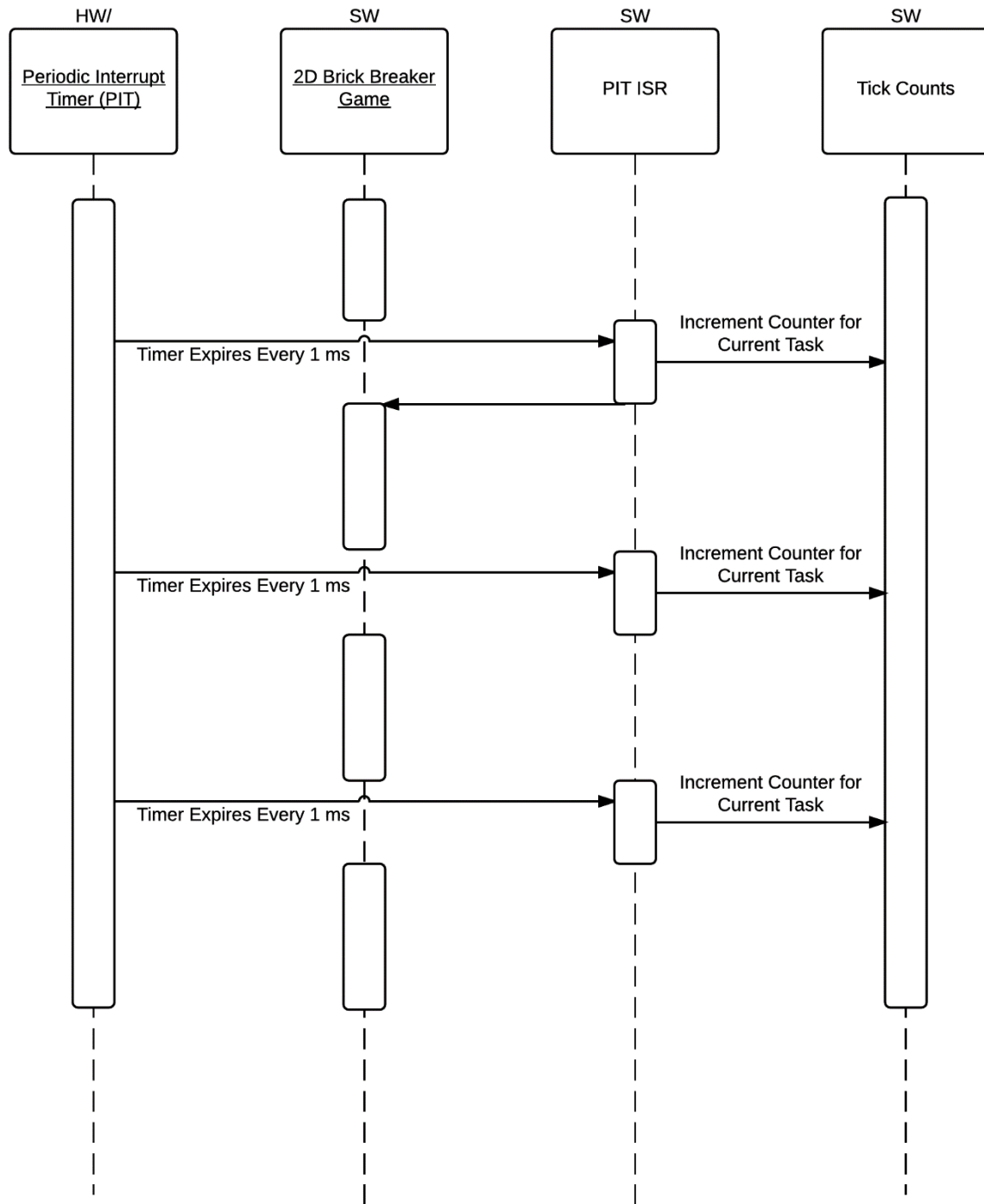
Development Effort Tracking

Estimated development time:

30 hours

Actual development time:

38 hours

The reason for the overage was because I decided to change the game that I was developing partially through development. Fortunately, I was able to copy most of the code base over to the new game, so this did not cause a significant problem.

Demo Video

A demo of the gameplay can be seen at the following link on YouTube:

https://www.youtube.com/watch?v=xD-wBMo0qSI

Apologizes for the low quality but my cellphone camera does not work and I only had access to an archaic camcorder.