Music Synthesizer Report

Group Cyberpsychosis

Yuting Xu (yx8918)

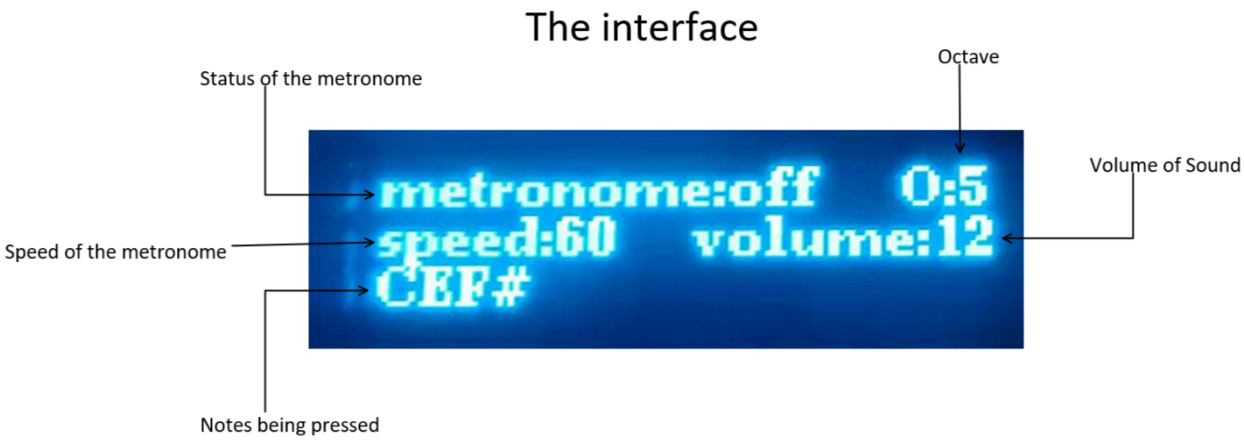
Keran Zheng (kz5218)

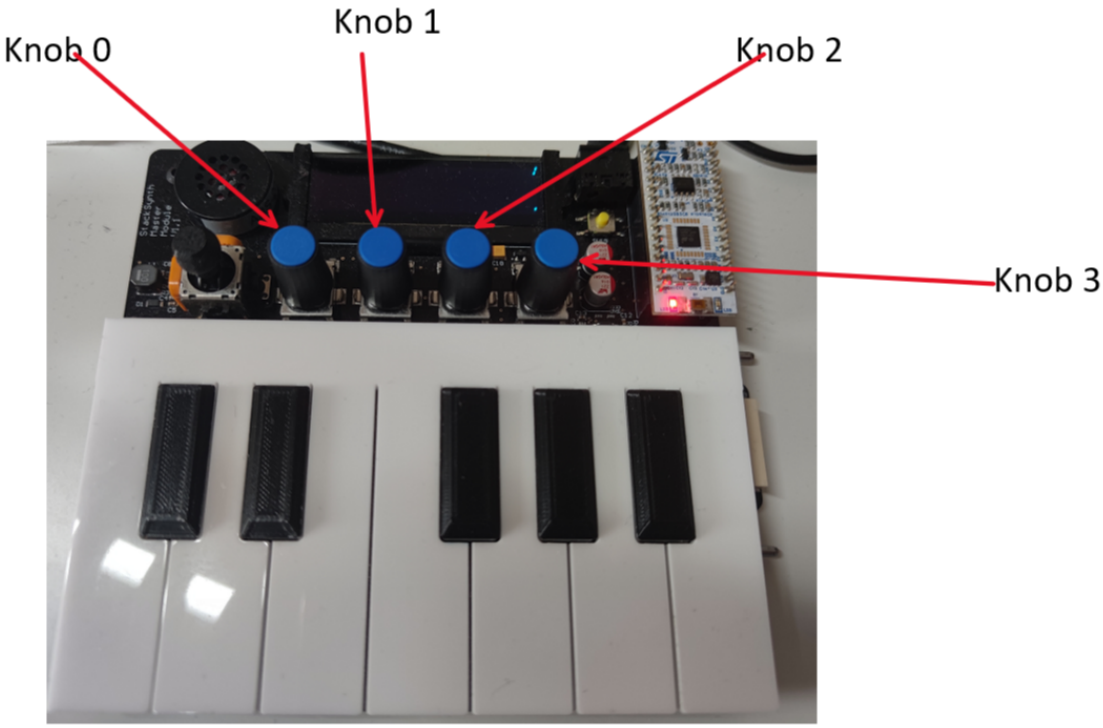
Haolong Song (hs5518)

Weiqian Ni (wn1918)

Table of contents

Introduction and User Interface





Our interface shows the following key information:

- Metronome status and the beats per minute (bpm).
- Current octave that the keyboard is on.
- Volume of the keyboard, 0 being muted, 16 being maximum.
- A display of all the notes currently pressed.
- Pressing knob 0: Envelope activated.

- Turning knob 1: Change the octave.
- Pressing knob 1: Reset the octave to 4.
- Pressing knob 2: Activate the metronome.
- Turning knob 2: Alter speed of metronome.
- Turning knob 3: Change the volume.
- Upshift/downshift the joystick: Increase/decrease the pitch.

Implemented Features

Here we only listed the features, for the complete documentation on interface and implementation, please check out the markdown file in our repo (insert link here).

Core functionalities

1. The synthesizer plays the note in an octave as sawtooth wave with no perceptible delay.
2. There is a volume control with at least 8 increments, which is changed by turning knob three.
3. A user interface displaying current volume.
4. The synthesizer can send a message over the serial port whenever a key is pressed or released.
5. The synthesizer can play a note or stop playing a note when it receives an appropriate message on the serial port.
6. Serial port can take a message of the specific format which controls the synthesizer to play or stop playing a note.

Advanced features

1. Implemented an envelope to simulate the decaying sound of a real piano, can be switched on and off by pressing knob zero.
2. Can use the joystick to change the tone to achieve additional sound effects
3. Implemented a metronome that sounds independently from the keys, speed of the metronome is changed by turning knob two and switched on and off by pressing knob two.
4. Can switch to other octaves by turning knob one.

5. 14 channel polyphony is implemented with the 12 keys having separate channels. The additional two channels are used for the metronome and the input from serial port, respectively.
6. The system is implemented using interrupts and threads to achieve concurrent execution of tasks. Interrupt is used to make sound over the speaker and the
7. All data and other resources that are accessed by multiple tasks are protected against errors caused by simultaneous access. Atomic operations have been used wherever it is considered useful. Special care has been taken to prevent concurrent changes on shared variables.
8. The code is well-structured and maintainable. Helper functions and LUTs are placed in separate header files. All functions are well commented and have detailed specifications.

System Analysis

Implemented Tasks

Here we list the tasks implemented in descending order of task priority.

- sampleISR () – Interrupt Service Routine

  Worst case execution time: 23.16us

  The sampleISR task implements a check on all the activated features and finally performs processing on each channel as specified by the keys pressed. The worst case is measured when all implemented advanced features are activated, and all keys are pressed.

- msgInTask () – Thread priority: 4

  The msgInTask decodes the 3-character message sent in through the Serial port. For a single line command, the whole process takes about 20us in all cases. Assuming the serial port receives 12 note commands at once the total delay for the task is approximately 240us.

- msgOutTask () – Thread priority: 3

  msgOutTask is initiated when the queue has elements, as it sends the elements in the queue over the serial port. Hence the worst-case scenario is when the queue is full. The worst-case execution time is measured to be 28 µs.

- scanKeysTask () – Thread priority: 3

  This is the foundation for all other tasks. Hence its performance must be closely monitored for it not to affect other threads' performance. The test was performed under the worst-case scenario with all keys pressed down and update disabled (the update function is still being ran, but it writes to a variable that is of no use). The msgOutTask had to be disabled for the serial to show the performance stats. Hence, queue was updated to be 12 * 32 = 384 bytes long for this test.

  Over 32 cycles, the total execution time was recorded to be 8703 µs. This averages out to be 272 µs per cycle.

- metronomeTask () – Thread priority: 1

  This task implements a variable beat generator as specified by a variable controlled by knob 2 in a range 0-255. The task controls the timing of the metronome by varying its execution interval. There is no worst-case scenario for this function, its execution time is independent of any state. It always updates the fixed length buffer the same number of times. There are no branching statements that would cause any performance costs. Worst case execution time average over 32 cycles is 115.5 µs.

- updateDisplayTask () – Thread priority: 2

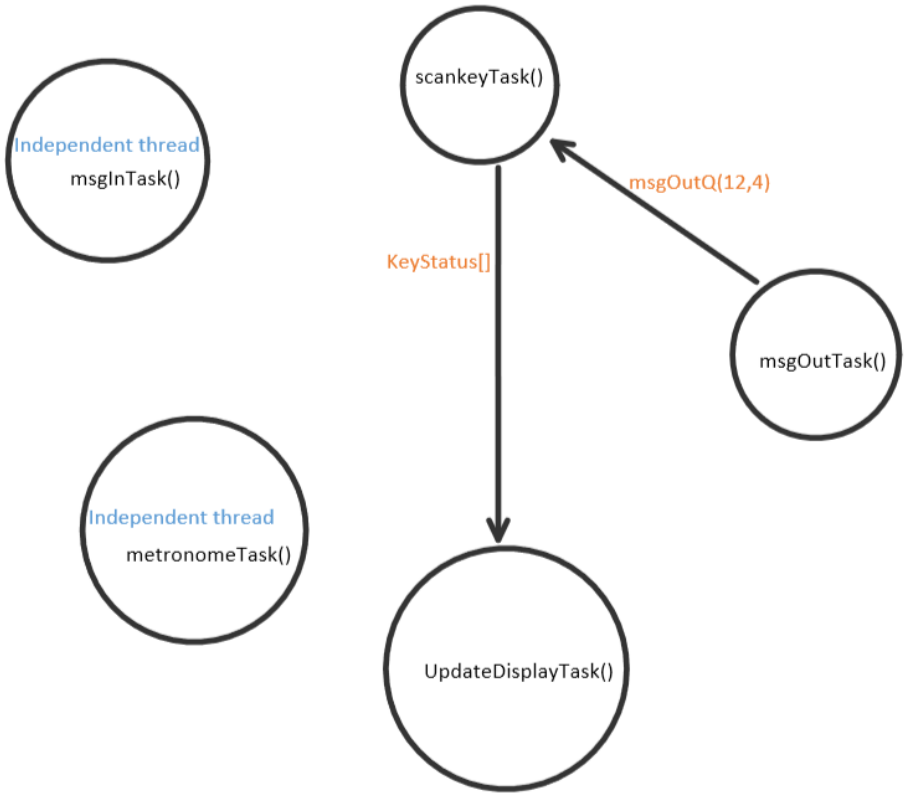
  The worst-case scenario for updateDisplayTask() is when all the keys are pressed down. The for loop must iterate through the char array and print out all its elements.

  Worst case execution time: 18,337 µs

Data structures

Global Variables	Accessed by tasks (Read, Write)	Description	Protection
uint32_t currentStepsizes[13]	SampleISR(R), msgInTask(R,W), scanKeysTask(W)	LUT used in changing keyboard octave, contains shifted step size	Atomic operations
bool keyStatus[13]	SampleISR(R) , msgInTask(W), scanKeysTask(W)	A boolean array to store which keys are pressed	Mutex
int16_t Joy_y	SampleISR(R), scanKeysTask(W)	A signed integer that holds the joystick reading	Mutex
Bool envelopeEnabled	sampleISR(R), scanKeysTask(W), updateDisplayTask(R)	A boolean variable to store the activation status of envelope feature	Atomic operations
msgOutQ(12, 4)	msgOutTask(R), scanKeysTask(W)	A FreeRTOS managed queue of size 12, with each item of 4 bytes, which stores message to be sent over the serial port	Queue
Uint8_t keyArray[7]	scanKeysTask(W), updateDisplayTask(R)	An array which contains the key array combinations for each scanKeysTask cycle	Mutex
Bool metronomeEnabled	scanKeysTask(W), metronomeTask(R), updateDisplayTask(R)	A boolean variable indicating if the metronome feature is enabled	Atomic operations
Uint8_t metronome	scanKeysTask(W), metronomeTask(R), updateDisplayTask(R)	Variable containing the metronome beat per second as specified by knob 2 (third one from left)	Mutex
Uint32_t metronomeBuffer[512]	metronomeTask(W), SampleISR(R)	Variable containing ‘tick’ and ‘tock’ sounds for sampleISR to produce metronome sounds.	State machine
Uint8_t volume	scanKeysTask(W), SampleISR(R)	Variable containing current volume.	Atomic operations
Uint32_t baseTime	scanKeysTask(W), sampleISR(R)	Variable to hold the time at which a key is pressed. Used by sampleISR to calculate decay envelope	Mutex
Uint16_t metronomeBufferLength	metronomeTask(W), sampleISR(R)	This is used as an indicator for sampleISR to read the metronome buffer or not.	Atomic operations

Dependency analysis



Through analysis, we find that dependencies exist between scanKeysTask and msgOutTask; updateDisplayTask and scanKeysTask.

In scanKeysTask, the thread blocks if the msgOutQ is filled. In our implementation this situation seldom happens as it is assumed that, in most cases, at most 8 keys will be pressed simultaneously. By setting the msgOutQ with size 12 we eliminated the possibility of filling the queue.

Another possibility of blocking exists where updateDisplayTask requires the access to the keyStatus array, which is protected by a mutex which could potentially be blocked by scankeysArray's dependency on msgOutTask. However, since scanKeysTask's dependency is largely eliminated by introducing a large Queue, the above possibility is mostly eliminated as well.

Critical instant analysis

Task	*Worst-case execution time(μs)*	*Minimum Initiation interval (μs)*	*RMS Priority*	τ_n/τ_i *(%)*
scanKeysTask	272	50,000	3	0.544
msgInTask	240	5,000	4	4.8
msgOutTask	28	50,000	3	0.14
metronomeTask	115.5	236,000*	1	0.0489
updateDisplayTask	18,337	100,000	2	18.337
sampleISR	23	45	N/A	51.1
Total	N/A	N/A	N/A	74.9

*MetronomeTask is a variable initiation interval task, in this case we consider the minimum initiation interval it can take.

t1: T_1=272, τ_1=50,000

t2: T_2=240, τ_2=5,000

t3: T_3=28, τ_3=50,000

t4: T_4=115.5, τ_4=236,000

t5: T_5=18337, τ_5=100,000

t6: T_6=23, τ_6=45

Within the timing constraint of 236,000 μs, we consider:

$$5 * t1, \quad 48 * t2, \quad 5 * t3, \quad 1 * t4, \quad 3 * t5, \quad 245 * t6$$

Hence, the total delay overhead for metronomeTask is:

$$1{,}360 + 11{,}520 + 140 + 116 + 55{,}111 + 120{,}635 = 188{,}882 < 236{,}000$$

Which is less than its minimum deadline, which proves that the scheduling meets the critical instant deadline.

Finally, the CPU Utilisation = 188,882 / 236,000 = 80.0%

Future improvements

Due to the lack of time or performance, some compromises were made in our design.

1. There is no management of sound channels. Although the current implementation works fine on the 12-key keyboard, this is not scalable when multiple keyboards are connected. A full-scale keyboard would require 84 channels! In a real-life scenario, we should expect less than four-hands-worth of keys (20) pressed at the same time. Assigning an individual channel to every key is a huge waste of resources.

   A possible improvement would be a 'queue' for channels, where the first 20 keys pressed each secure a channel to produce sounds. Its channel is released only when the key is released.

2. Global envelope that affects all the sound channels at the same time. Currently the envelope refreshes on any key press, regardless how long the previous key presses have decayed. This is counterintuitive, since keys should fade independently on a piano. However, we would have to calculate 12 envelopes in a single interrupt and keep track of all the press & release times. This will most likely cause the execution time of the interrupt to exceed its initiation interval.

3. More features could be added. These could include non-linear filters, echo effects.