SOLUTIONS: ANALYZING AND "OPTIMIZING" RESPONSIVENESS (V1.1)

CONTENTS

OVERVIEW

In this lab you will evaluate the timing characteristics of an RTOS-based system and then improve the quality of audio generation through scheduling and other changes. Please make sure there is no µSD card in the shield's socket, because this code uses the SPI pins as debug signals. Pick up a jumper wire (socket-to-socket) from the instructor or a TA to connect the DAC output to the AD2 1+ input.

You will need to submit your work and responses as follows:

- Google Forms (link to be announced):
  - Responses to numbered questions in this document.
- Moodle:
  - PDF lab report (based on the provided report template)
  - Build log file (Objects/*.build_log.htm)

Set up the code as follows:

- Download the project code for the lab from the archive.
- Confirm that you can build the program without errors. Note that we are not using the profiler in this work, so you only need to build the program once after a code change, not three times.

METHODS FOR ANALYZING THREAD TIMING

We will use the logic analyzer and debug signals to see when threads and handlers execute over time. We would like to understand the timing of the software. When does a thread run? When does an ISR run? How much time overhead does the RTOS take to switch between threads?

The software in this exercise supports two types of debug signals: user-defined signals and automatic thread visualization signals. You have used user-defined debug signals before, but they do not directly show RTOS activity. The code in this exercise makes the RTOS automatically generate thread visualization (TV) debug signals to indicate when threads resume running, are blocked, or are preempted. However, these TV signals are not generated for interrupt and exception handlers.

USER-DEFINED DEBUG SIGNALS

We can monitor some software timing with user-defined debug signals. This involves picking a specific GPIO output signal (e.g. DBG_0) for the activity and then adding macro calls (e.g. DEBUG_START(DBG_0), DEBUG_STOP(DBG_0)) to our code to control that output signal. Relevant macros and code are in debug.c and debug.h. In general, when the event starts, use DEBUG_START() to set the bit to 1. When the event stops, use DEBUG_STOP() to clear the debug signal to 0. To toggle the output, use DEBUG_TOGGLE().

Exception handlers and interrupt service routines are not threads, so they start running without RTOS involvement, so they are not supported by the thread visualizer. For this type of routine, you need to select a user debug signal, set it at the beginning of the handler/ISR (DEBUG_START) and clear it at the end (DEBUG_STOP).

- Add code to DMA0_IRQHandler (in DMA.c) to control DBG_1 showing when it is executing.

One special case is that we want to know when the RTOS idle thread is running, not just when it resumes or is preempted. The idle thread (RTX_Config.c: osRtxIdleThread()) is an infinite loop which runs if there are no other

threads ready to run. Making that loop toggle a debug signal will make idle processor time easier to see on the logic analyzer. Note that the signal is changing at a high frequency, so the logic analyzer's sampling may cause aliasing, so the signal appears to be constantly 1, constantly 0, or changing at a much lower frequency than reality. You may need to adjust the analyzer's time base (zoom in or out) to see the real signal without aliasing.

➢ Add code to osRtxIdleThread() (in RTX_Config.c) to toggle (invert) output signal DBG_0 each time the loop executes. If the signal is changing, that means the idle thread is running.

THREAD VISUALIZER DEBUG SIGNALS

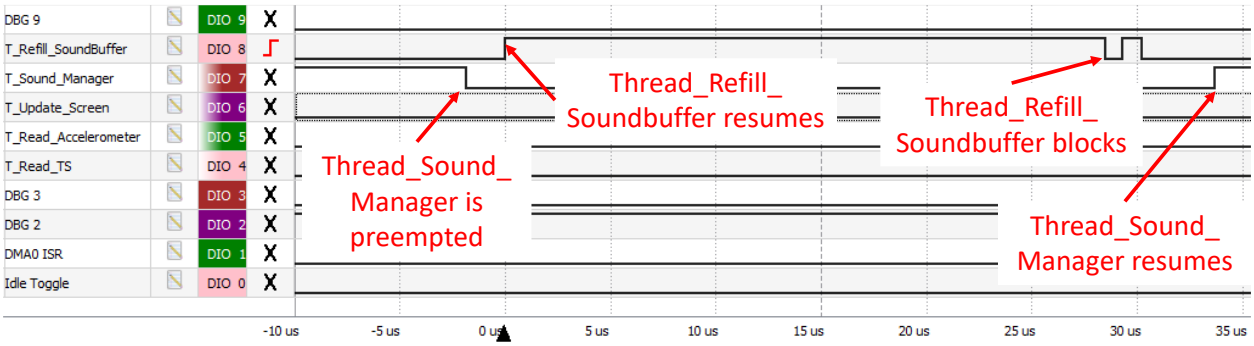


Figure 1. Example of thread visualizer signals. At -2 µs, Thread_Sound_Manager is preempted. At 0 µs, Thread_Refill_SoundBuffer resumes running (triggering the logic analyzer). At 28 µs, Thread_Refill_SoundBuffer blocks. At 34 µs, Thread_Sound_Manager resumes running.

The thread visualizer code in new_events.c/h lets you use a logic analyzer to see when different threads are executing in a system built on the RTX5 kernel. Each thread's activity is represented by a debug output signal. The kernel uses instrumentation code to automatically:

- Set the bit to one when the thread **starts** or **resumes** executing.
- Clear the bit to zero when the thread is **preempted** and stops running.
- Toggle the bit twice and then clear it to zero when the thread **blocks** and stops running. Based on the sampling rate and display time scale of your logic analyzer, you many need to zoom in to distinguish between blocking and preemption. Note that this toggling takes about 1.7 µs, delaying other activities slightly.

Signal Name in Code	osThreadNew Call Order	Use in this Code	AD2 DIO Signal	MCU Port Bit
DBG_0	n/a, user defined	Toggled by idle thread loop	DIO 0	D0
DBG_1	n/a, user defined	DMA0_IRQ Handler	DIO 1	D2
DBG_2	n/a, user defined	↑: Refill SoundBuffer requested ↓: First sample written to SoundBuffer	DIO 2	D3
DBG_3	n/a, user defined	unused	DIO 3	D4
DBG_4	1st	TV: Thread_Read_TS (touchscreen)	DIO 4	B8
DBG_5	2nd	TV: Thread_Read_Accelerometer	DIO 5	B9
DBG_6	3rd	TV: Thread_Update_Screen	DIO 6	B10
DBG_7	4th	TV: Thread_Sound_Manager	DIO 7	B11
DBG_8	5th	TV: Thread_Refill_SoundBuffer	DIO 8	E2
DBG_9	6th	TV: osRtxIdleThread	DIO 9	E3
DBG_10	7th	TV: osRtxTimerThread	DIO 10	E1
DBG_11	8th	unused	DIO 11	E4

As configured for this project, the debug code supports up to twelve debug signals. Debug signals are allocated to threads in the order that osThreadNew is called, up until all are used. The default signal allocation is shown in the table. After all user threads have been created, the kernel calls osThreadNew to create the idle thread and the timer thread, which may result in up to two more TV signals being allocated.

VIEWING DEBUG SIGNALS WITH LOGIC ANALYZER

- Connect the logic analyzer to the shield. Start the logic analyzer program (Waveforms).
- Build and download the program. Reset the Freedom board to start the program running. Run the logic analyzer without setting up any triggering. Adjust the sampling time base ("base") to about 50 ms/division. You should see something like

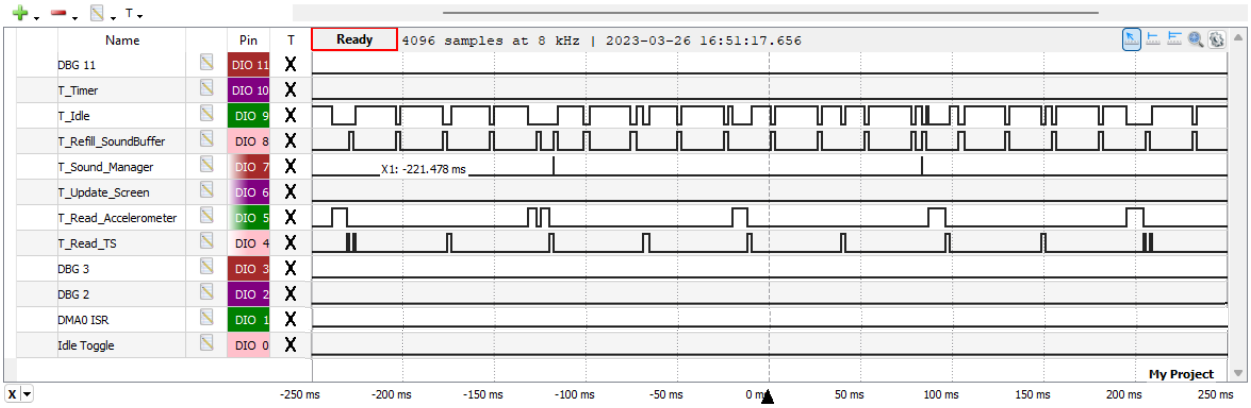


Figure 2:

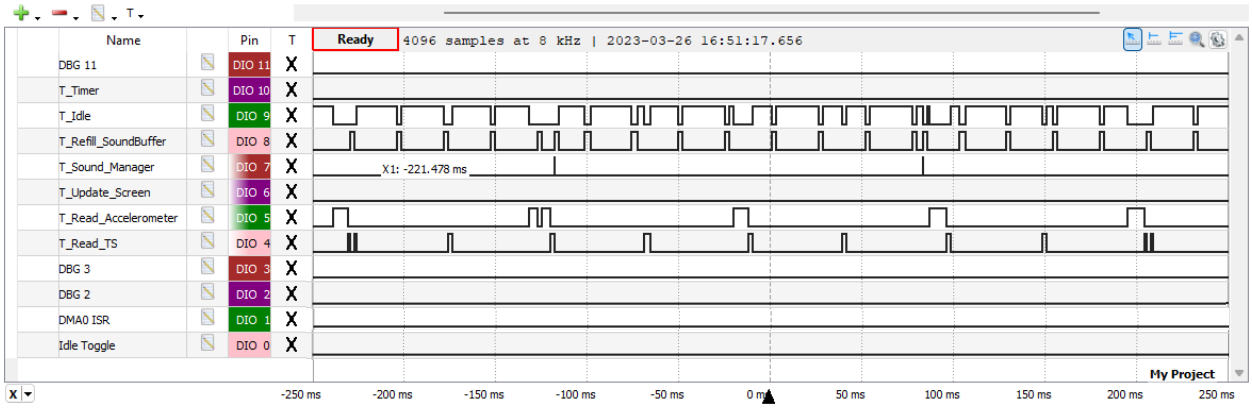


Figure 2. Example of logic analyzer display.

- In order to simplify data analysis, you may want to rename each channel in use based on the table above.

EXPERIMENTAL TIMING ANALYSIS

MEASURING TASK AND HANDLER COMPUTATION TIME

We need task and handler computation times to build a periodic task model of the system's software.

- To simplify the timing analysis, set all threads to have the same priority. This way, only ISRs can interfere with our timing measurements. Make sure that each **THREAD_..._PRIO** declaration in threads.h is **osPriorityNormal**.

1. Use the logic analyzer to complete the following table with the range of observed timing information for each thread or handler, except the Idle thread. Trigger the analyzer on the rising edge of the given thread's TV debug signal when possible. You may need to adjust the sampling rate to accommodate short or long functions. Move the board and press the screen to execute as much code as possible.

Thread or IRQ Handler	Maximum Measured Execution Duration C_i	Execution Period T_i
DMA_Handler	22.36 µs	25.6 ms
Read Touchscreen (with screen touched)	7.07 ms (Wrong: 2.32 ms without touch)	~57 ms (scope). OK: 50 ms (period in threads.h)
Update Screen	890 µs: tilted moving board. (Wrong: 28.5 µs without movement)	~57 ms (scope). OK: 50 ms (period in threads.h)
Refill SoundBuffer	~2.9 to 5.5 ms (may have measured full and partial (with variable amounts of work))	25.6 ms
Sound Manager	145 µs	200.1 ms. OK: 200 ms, defined in sound.h. OK: 250 ms (there is dead code in threads.h)
Read Accelerometer	~ 8.7 ms	~108.7 ms. OK: 100 ms (period in threads.h)

Points are allocated per cell based on how closely it matches the expected value(s).

2. You should see only one thread being preempted. Which thread is preempted, and is this acceptable?
Only the idle thread is preempted, and this is fine.

ANALYZING THE SOUNDBUFFER REFILL LATENCY

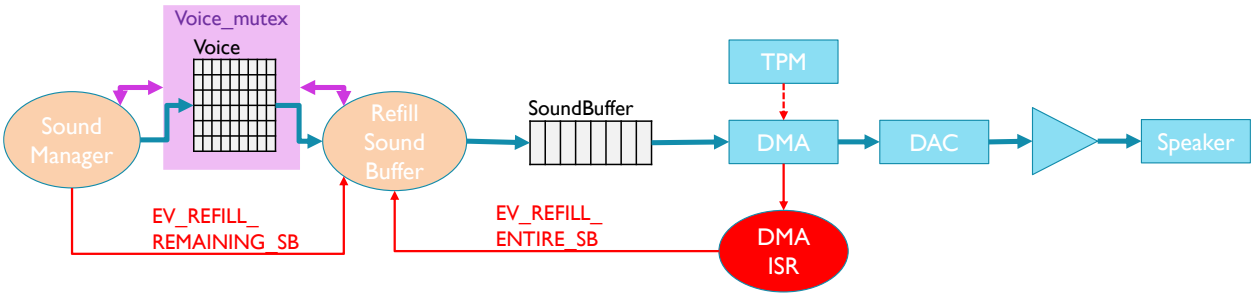


Figure 3. Sound generation architecture with default single buffer.

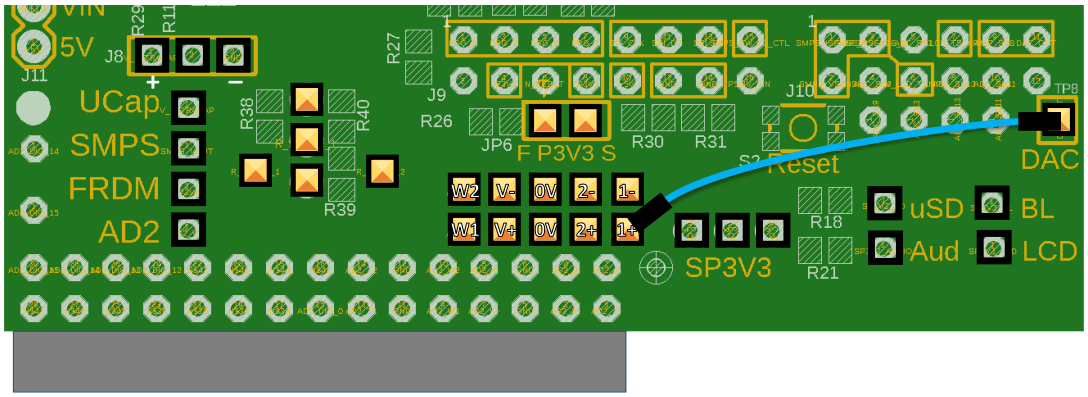
One time-critical operation in the shield code is refilling the sound buffer with audio samples before the DMA reads the next sample. By default, a single buffer is used. The sound buffer is refilled using this sequence of events:

- DMA transfers last sample, triggering IRQ.
- DMA IRQ handler runs, setting an event flag for Thread_Refill_Sound_Buffer.
- RTOS sees the set event flag and unblocks Thread_Refill_Sound_Buffer, allowing it to run eventually (once it is the highest priority ready thread).

Because the playback rate is one sample per 50 microseconds, the first sample needs to be in the buffer within 50 microseconds of the previous sample being used.

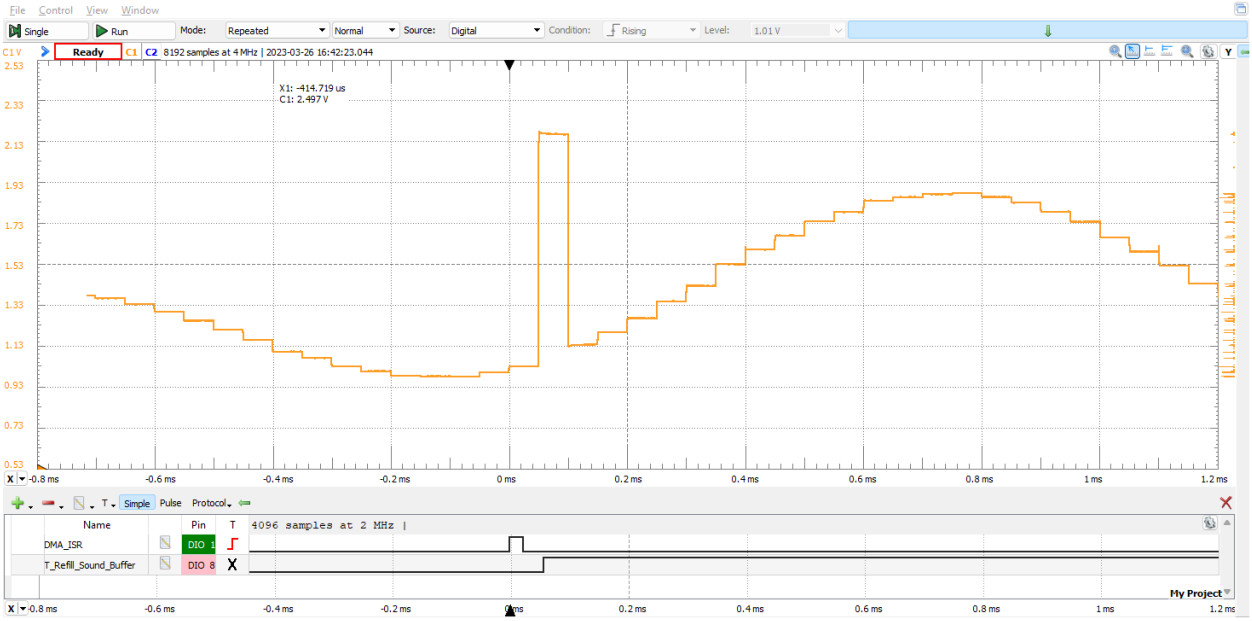
If you press **Unmute** on the LCD you'll hear the flawed audio output, with lots of buzzing and clicks which come from the sound buffer not being refilled in time. Let's look at the analog output of the DAC and directly see the impact of the delayed sound buffer refill. Use the Analog Discovery 2 to monitor the analog output (as well as the digital outputs).

- Connect Channel 1+ to the DAC output. Use a socket-socket jumper wire from as shown:



- Configure the Waveforms program to provide a mixed-signal display. On the Welcome tab, select **Scope**. In the View menu, select Digital (at the bottom of the list).
- In the newly-created logic analyzer window, press the green **+** to add the debug signals for the DMA IRQ Handler (DIO 1) and the TV signal for Thread_Refill_Sound_Buffer (DIO 8). Trigger using the logic analyzer, on the rising edge of the DMA IRQ Handler. Start the program running on the MCU board and press Run in Waveforms.

You should see a display similar to the one shown here, though the time scale and amplitude will vary.



Notice the error in the oscilloscope trace – the bad data output sticks up like a sore thumb. The DAC output changes before Thread_Refill_Sound_Buffer has a chance to change the first sample in the buffer. The timebase is set to 200 us/division, so each division represents four output samples. This new DAC output is wrong - it is an old value which hasn't been updated yet. The DAC output should change again after 50 us to correct value, and subsequent values should also be correct.

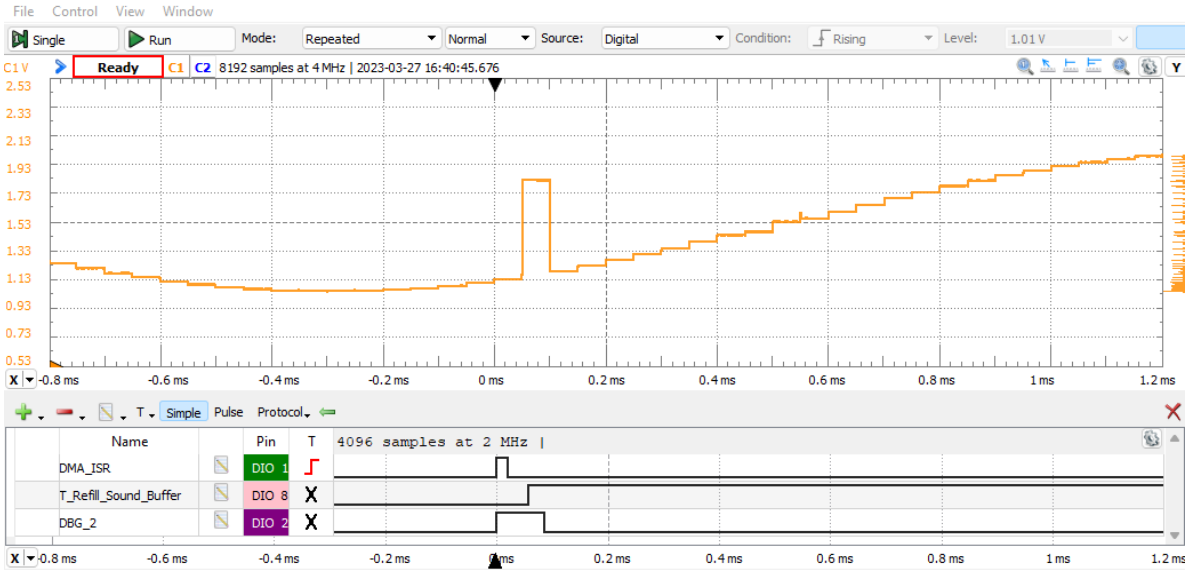
- Press Unmute on the LCD and listen to the audio. There should be occasional clicking, indicating the bad data outputs.

MAKING THE REFILL LATENCY VISIBLE

Let's simplify the timing delay analysis by making signal DBG_2 explicitly show the time from the refill request to when the first sample is written.

- Change the DMA IRQ Handler to set debug signal DBG_2.
- Modify Thread_Refill_Sound_Buffer to clear DBG_2 after it has written the first sample to the buffer.
- In the logic analyzer window, press the green + to add DBG_2 (DIO 2). Trigger the scope on the rising edge of DBG_2 and start the scope running. You should see something similar to this, with the C1 (orange) and

T_Refill_SoundBuffer and DBG_2 traces varying as the scope is triggered:

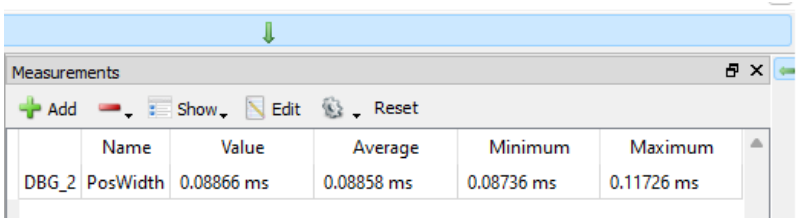


3. Let the scope run for a few seconds. What is the shortest duration DBG_2 pulse you see? (Format: 1.23 us)
   87.34 µs

AUTOMATIC REFILL LATENCY MEASUREMENT

The Waveforms scope tool can measure the input channel characteristics, such as the width of the DBG_2 signal.

- In the Scope 1 View menu items, select Digital Measurements. This will create a Measurements window.
- Click the + Add button to get the Add measurement dialog box. Select the digital signal to measure (DIO 2 or DBG_2 if renamed) and select PosWidth (width of positive pulse). Click Add and then click Close.
- Click the Show button and check the Average, Minimum and Maximum entries.
- Click the gear icon and check the Multiple Acquisitions box.
- Start the scope running. You should see something like this:



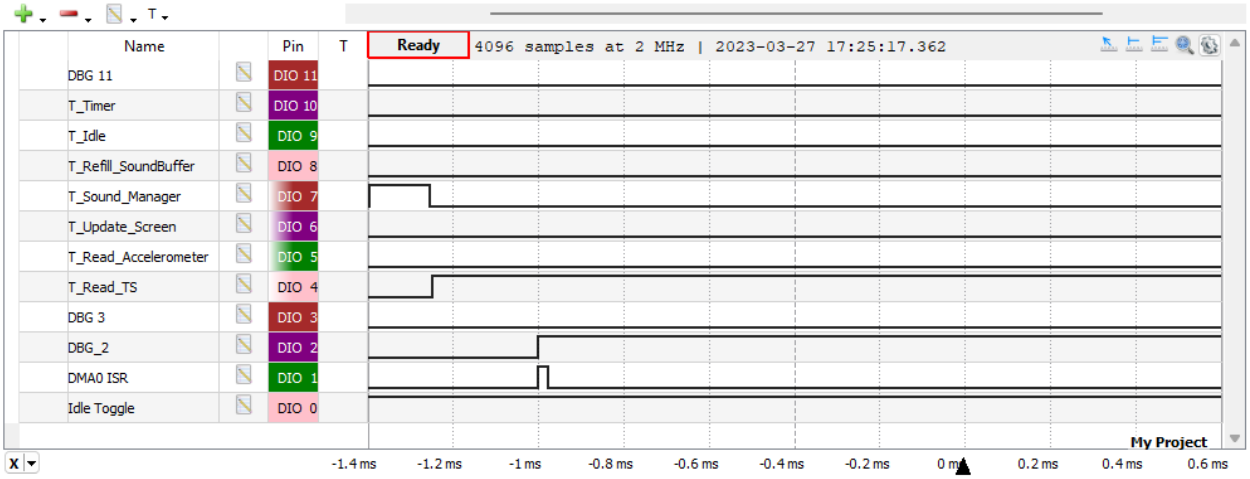
- Note that the Maximum value is limited to the maximum time range shown on the horizontal axis (1.2 ms in this case). Increase the time per division as the scope is running until the maximum stops increasing.

4. What is the maximum positive pulse width of DIO 2 (i.e. delay to update the first SoundBuffer sample)? It should be several ms. (format: 1.23 ms)
   8.5125 ms

WHAT MAKES THE REFILL LATENCY SO LONG?

Let's switch to the logic analyzer to see what the other threads are doing when we get multi-millisecond latencies.

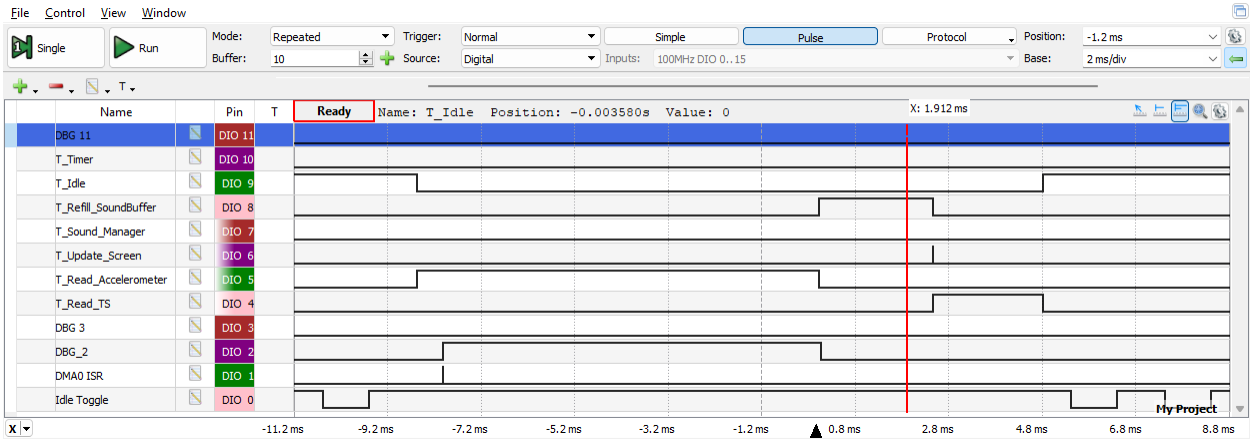
- Stop the scope and switch to the Logic 1 tab.
- Click the Pulse trigger button, then select the Timeout tab, setting Source to DIO 2, Polarity to Positive, and More than: to 1 ms. Click OK.
- Run the logic analyzer. It will trigger (t =0 ms) when the DIO 2 signal has been asserted for 1 ms. The rising edge of DIO 2 should be at t = -1 ms; adjust the time base to see that if needed. Here is an example which shows that Thread_Read_TS started running at about t = -1.25 ms. Although the refill was requested at t = -1 ms, it still hadn't been serviced as of t = 0.6 ms because Thread_Read_TS kept running.



5. Increase the timeout value to just below the maximum delay you determined in the previous question. Note that you can type in a specific numerical value (e.g. 1.23 ms) instead of just selecting one of the listed timeouts. Capture the logic analyzer traces in a screenshot and include it in your report.

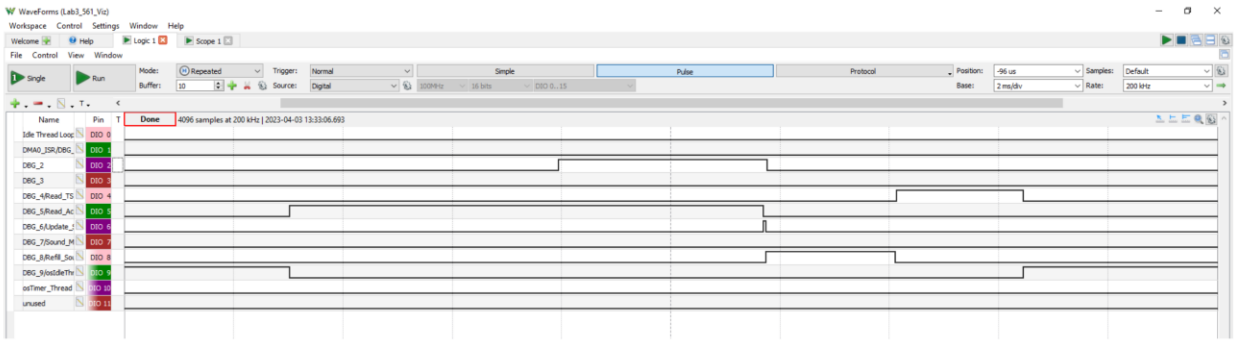
Solution Screenshot:

Has labeled signal traces. Includes timing information (e.g. X axis labels, time base ("2 ms/div").
Shows entire DBG_2 pulse (8.5 ms), starting with DMA0_ISR and ending right after T_Refill_SB starts running, and with intervening thread (e.g. Read Accelerometer).

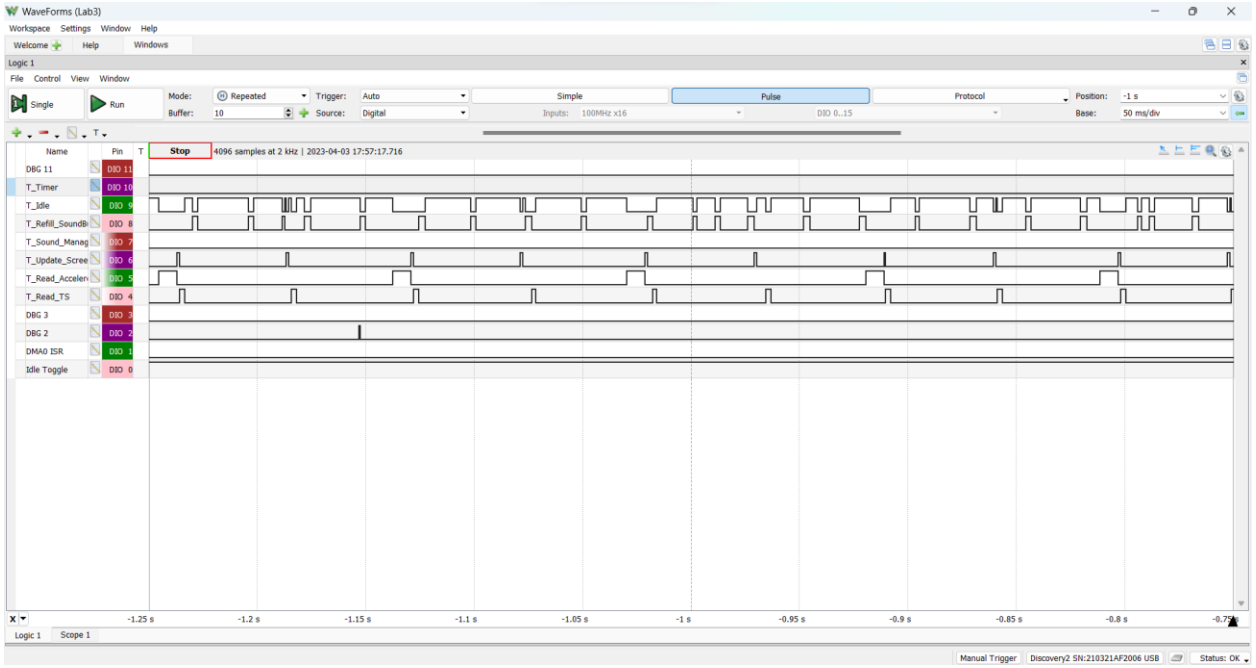


Examples of screenshots which lost points:

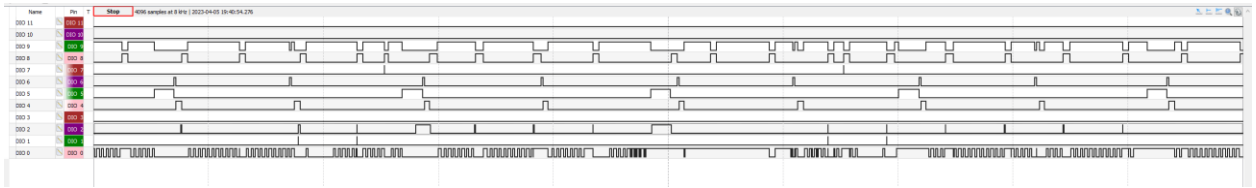
DMA0 ISR signal missing, hard to read text (labels, timing information)



Sampling rate (time base) too slow, so can't measure width of DBG2.



Timing information cropped from image, text is very hard to read, signals not labeled, sampling rate too slow to measure DBG2 pulse width.



6. Which threads and/or ISRs are delaying the refill request in your specific screenshot case?
   Thread_Read_Accelerometer here.
7. How do their execution times compare with the corresponding maximum execution times (C_i) you provided in Question 1?
   Thread_Read_Accelerometer takes about 8.6 ms, which is close to the observed maximum above.

IMPROVING THE SOUNDBUFFER REFILL LATENCY

CHANGING THREAD PRIORITIES

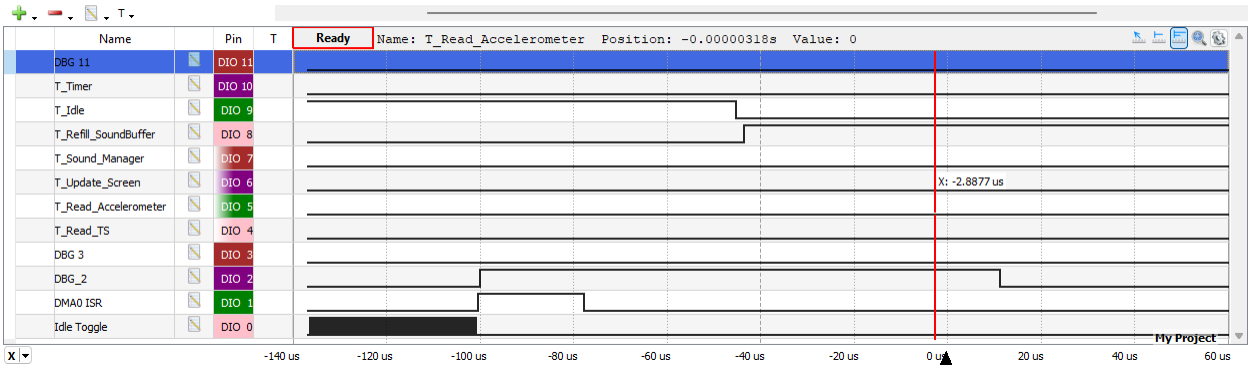
One problem with this system is that Thread_Refill_Sound_Buffer is not given priority over other threads, making its response time longer than necessary. Let's raise the priority of Thread_Refill_Sound_Buffer above all other threads.

- Examine the structure Refill_Sound_Buffer_attr in threads.c to see that its .priority field is set to THREAD_RSB_PRIO, which is defined in threads.h. Change the definition from osPriorityNormal to osPriorityAboveNormal. Valid priority levels are define in cmsis_os2.h (search for osPriority) and online here. Save the files, rebuild the project and download it.
- Switch back to the scope window. Click Reset in the Measurements window to reset the statistics. Start the scope running again.

8. What is the maximum positive pulse width of DIO 2 (i.e. delay to update the first SoundBuffer sample)? It should be well under one ms, but still larger than the deadline of 50 µs. (format: 1.23 ms)
   116 us (or about 90 us if DIO_2 is set at end of ISR)

- Switch to the logic analyzer and set the timeout to a little less than the maximum delay you just measured.

9. Capture the logic analyzer traces in a screenshot and include it in your report.
   Solution Screenshot:
   Has labeled signal traces. Includes timing information (e.g. X axis labels, time base ("2 ms/div").
   Shows DBG_2 pulse (116 us), starting with DMA0_ISR and ending right after T_Refill_SB starts running, with no threads running during that time (other than the previously running thread, which hasn't had its TV signal updated yet).



10. Which threads and/or ISRs are delaying the refill request in your specific screenshot case? Describe the sequence of events.
11. How do their execution times compare with the corresponding maximum execution times (C_i) you provided in Question 1?

USING DOUBLE-BUFFERING FOR SOUND BUFFER

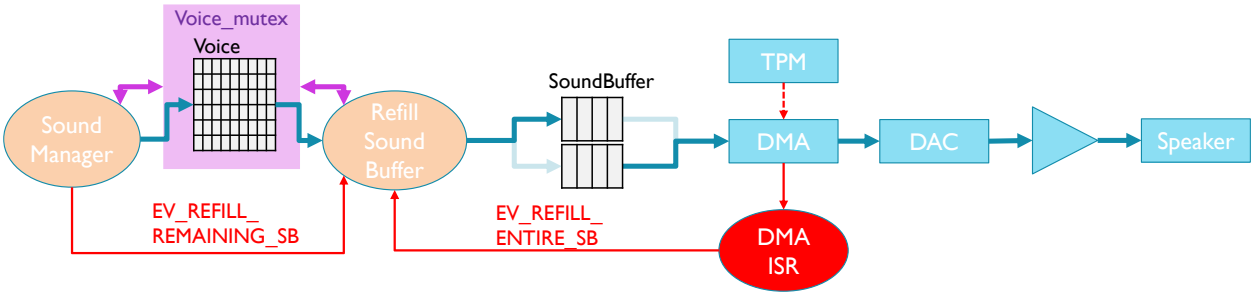


Figure 4. Sound generation architecture with double buffer.

The code has support for double-buffering the sound output. This raises the deadline for refilling the first SoundBuffer entry from $1*T_{Sample}$ = 50 µs to NUM_SOUNDBUFFER_SAMPLES$*T_{Sample}$ = 256*50 µs = 12.8 ms.

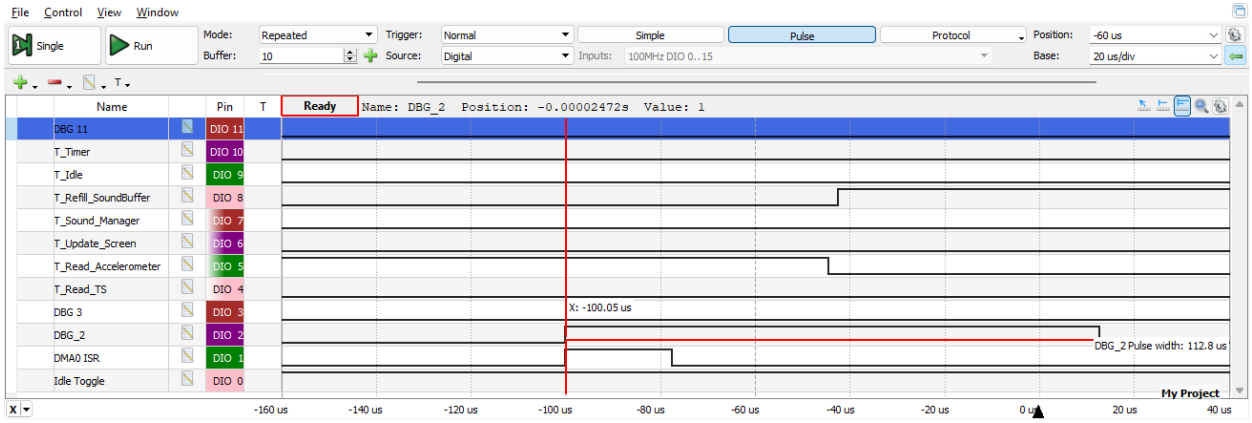
- To enable it, change the definition of USE_DOUBLE_BUFFER (in sound.h) from 0 to 1. To keep the total buffer memory size the same, each buffer is half the size of the original buffer. This reduction will double the release frequency of DMA controller interrupt, its handler, and Thread_Refill_Sound_Buffer.
- Save the files, rebuild the code and download it.
- Switch back to the scope window. Click Reset in the Measurements window to reset the statistics. Start the scope running again.

12. What is the maximum positive pulse width of DIO 2 (i.e. delay to update the first SoundBuffer sample)? It should be well under the deadline of 12.8 ms. (format: 1.23 ms)
    0.1135 ms, 0.133 ms
13. Watch the DAC output on the scope for at least 20 seconds. Do you ever see a corrupted sample?
14. Press Unmute on the LCD and listen to the audio. Is the sound quality better? Most of the clicking should be gone, but there are other bugs remaining in the system.

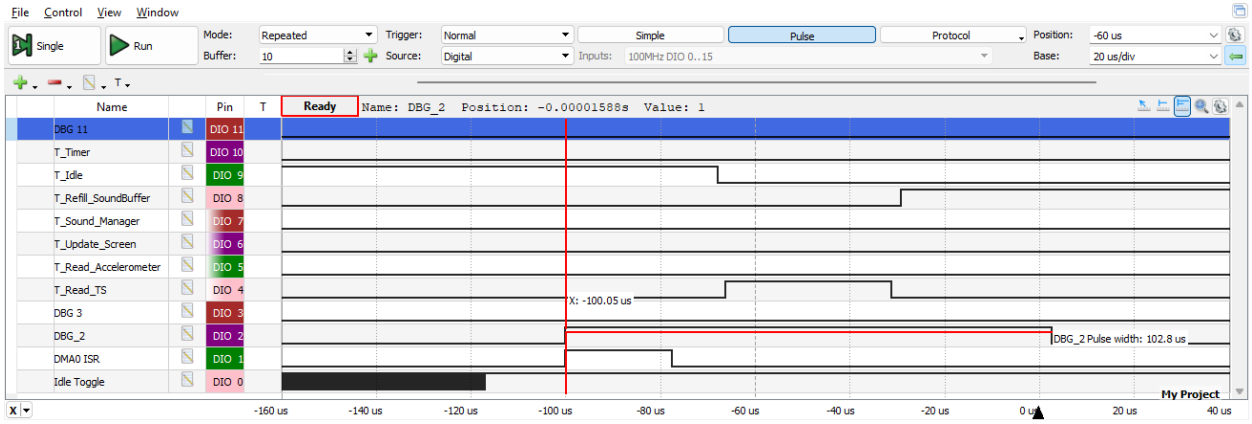
- Switch to the logic analyzer and set the timeout to a little less than the maximum delay you just measured.

15. Capture the logic analyzer traces in a screenshot and include it in your report.
    Has labeled signal traces. Includes timing information (e.g. X axis labels, time base (“2 ms/div”).
    Timing reference information present, appropriate zoom, DBG_2 pulse width around 130 us. See examples below.

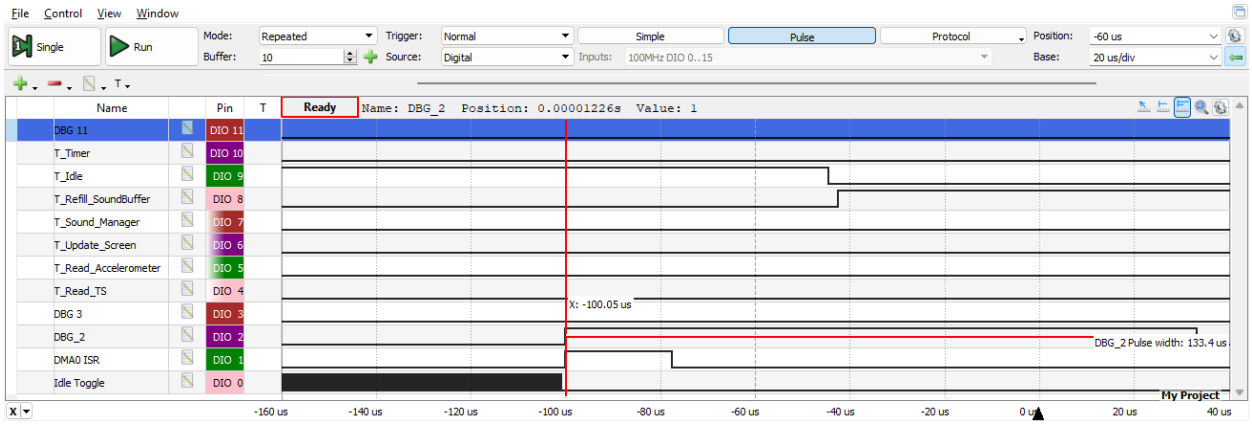
16. Which threads and/or ISRs are delaying the refill request in your specific screenshot case? Describe the sequence of events.
    Thread_Read_Accelerometer was running: 112.8 us:



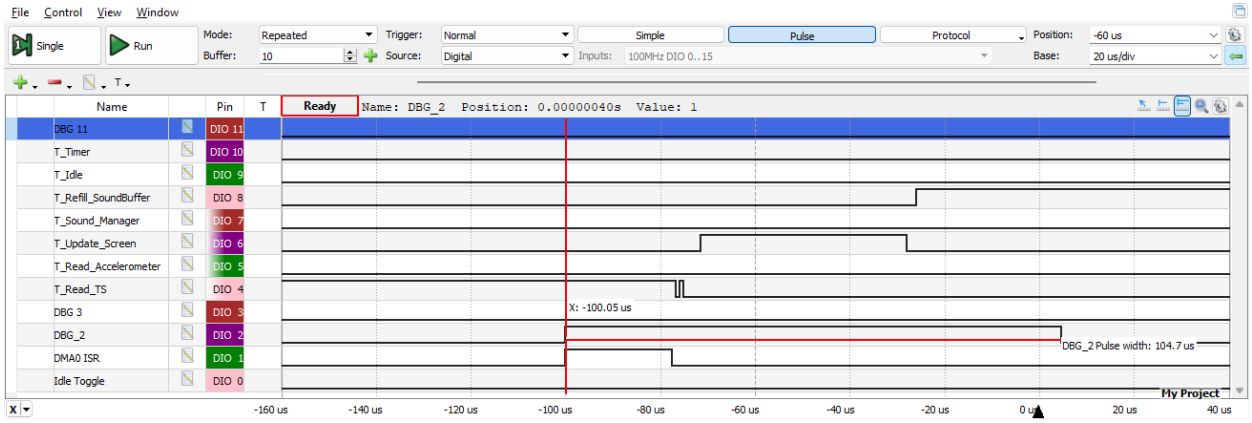
Why does T_Read_TS run before T_Refill_Soundbuffer?:



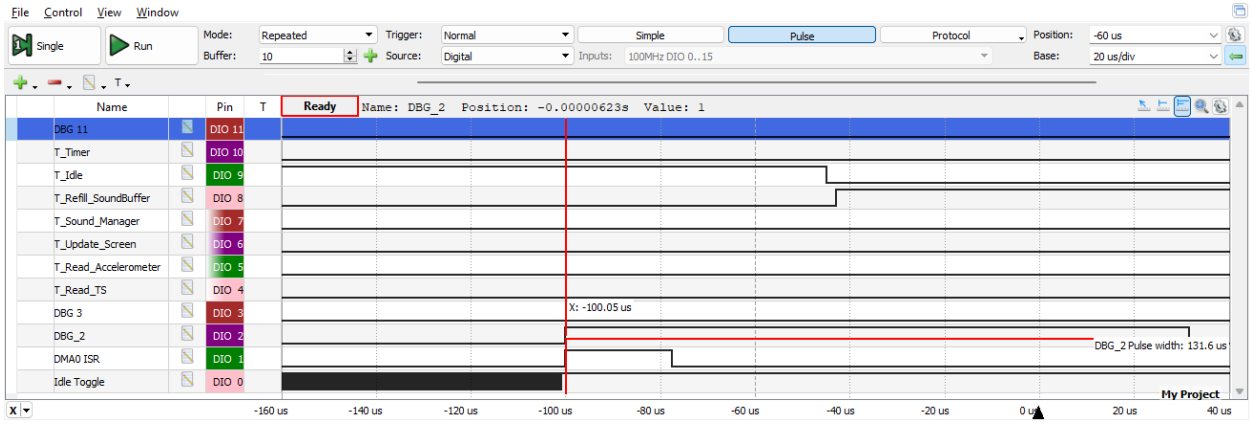
Screen touched, and idle thread was running: 133.4 us:



Thread_Read_TS was preempted by Thread_Update_Screen (104.7 us):



Screen untouched, and idle thread was running: 131.6 us:



17. How do their execution times compare with the corresponding maximum execution times (C_i) you provided in Question 1?
    Times should be no larger than C_i above.

ECE 561 ONLY: MAXIMIZING THE UPDATE RATE

The deadline with the double buffer is so long that we now have large timing margin. Let's see how high of a sampling frequency this system can support.

➢ Change the definition of AUDIO_SAMPLE_FREQ from its initial frequency of 20,000 Hz (20 kHz). Note that this reduces the deadline to NUM_SOUNDBUFFER_SAMPLES/AUDIO_SAMPLE_FREQ. At 100 kHz, the deadline will be 2.56 ms. Raise AUDIO_SAMPLE_FREQ to the closest multiple of 10 kHz which works.

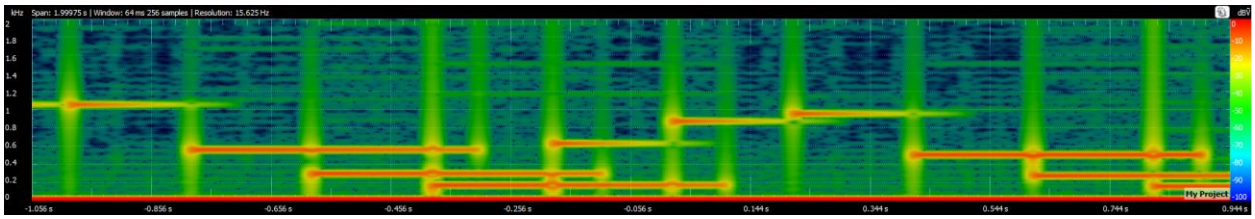
18. What is your maximum value of AUDIO_SAMPLE_FREQ (a 10 kHz multiple) which works?
    ~150 kHz? Depends on definition of "works"

    Triggering Frequency for T_Refill_SoundBuffer
    Both DMA_ISR and T_Sound_Manager trigger T_Refill_SoundBuffer. Examine their triggering frequencies:
    DMA_ISR:
    $f_{release\_DMA}$ = AUDIO_SAMPLE_FREQ/NUM_SOUNDBUFFER_SAMPLES = 100 kHz/256 = 390.6 Hz.
    T_SoundManager:
    $f_{release\_SM}$ = 4 Hz. C_{SM} = 115 us.

Compute Times

T_Refill_SoundBuffer:

Due to full buffer refill: C_{RSB_Refill} = 1215 us or 1548 us. Depends on how many voices are active (2 to 3). A new note is generated every 200 ms. Each voice's note duration is hardcoded to AUDIO_SAMPLE_FREQ/2, so it will last 500 ms. So for the DMA-triggered full buffer refills, either 3 notes need to be generated (5/6 of the time), or 2 notes (1/6 of the time).



Due to partial buffer update (due to new note): C_{RSB_Update} = depends on how much of buffer must be updated, from nothing up to the maximum (max(C_{RSB_Refill}). Remember that new note generation (sound manager) and full buffer refill (DMA IRQ) are not synchronized.

DMA_ISR:

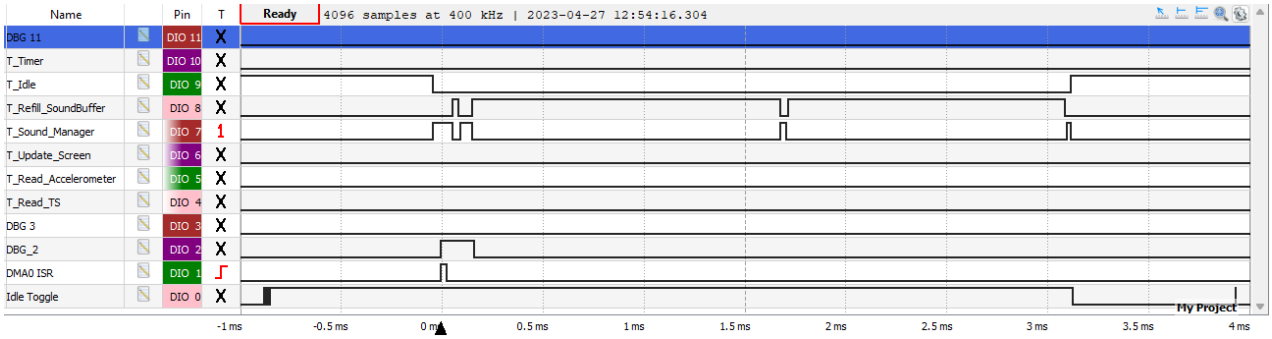
C_{DMA_ISR} = 22.8 us.

T_SoundManager:

C_{SM} = 115 us (200 us with OS overhead).

$U = U_{DMA\_ISR} + U_{T\_SM} + U_{T\_RSB}$ = (390.6 Hz * 22.8 us) +

Extreme Example



Sequence:

- -40 us: T_SM starts running and aquires Voice_mutex just before DMA empties buffer.
- 0 us: DMA ISR starts running, sets event flag (EV_REFILL_ENTIRE_SB).
- 53 us: Scheduler preempts T_SM since event unblocks T_RSB.
- 83 us: T_RSB blocks on Voice_mutex, T_SM resumes running
- 148 us: T_SM releases Voice_mutex, T_RSB aquires it and resumes running
- 157 us: T_RSB updates first sample in waveform buffer
- 1675 us: T_RSB finishes filling sound buffer and blocks waiting on event flag. T_SM resumes running and sets event flag (EV_REFILL_REMAINING_SB).
- 1700 us: T_SM stops, T_RSB resumes
- 3090 us: T_RSB finishes updating sound buffer, T_SM resumes
- 3113 us: T_SM finishes. Elapsed time = 3153 us.

Notes:

C_{SM} = 200 us with OS overhead, up from 115 us.

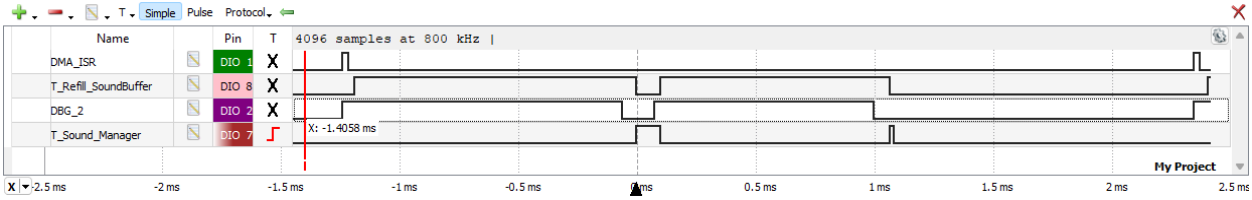
C_{RSB} = 27.5 us + 1525 us + 1377 us = entire refill (1552.5 us) + update (1377 us)

So minimum period = 3153 us.

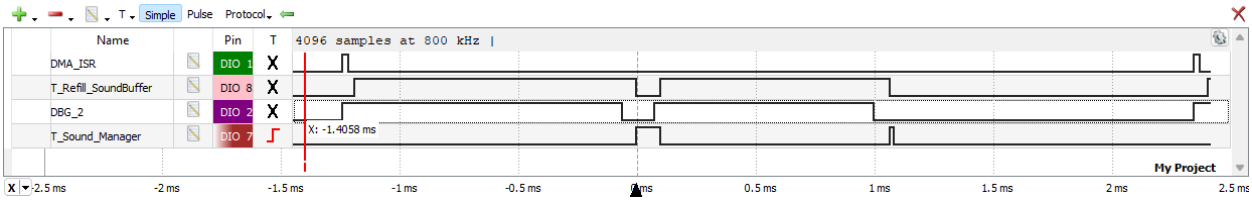
Experiments:

Measure margin between last sample written by T_Refill_SoundBuffer (**indicated by DBG_2 with new code**) and next DMA ISR starting.

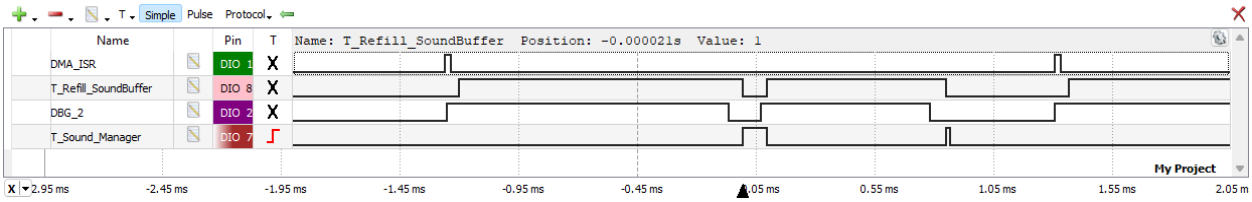
70 kHz: 1.346 ms margin



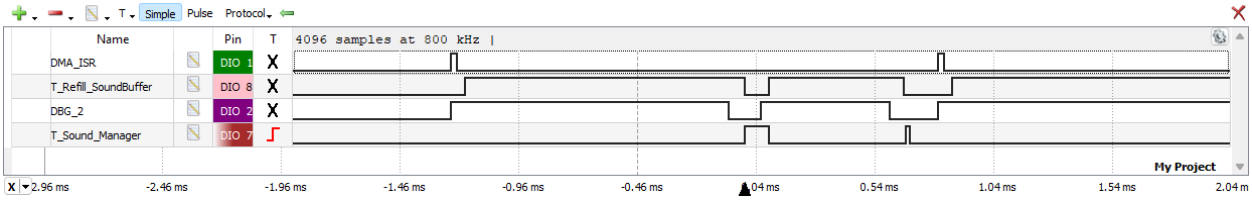
80 kHz: 855 us margin



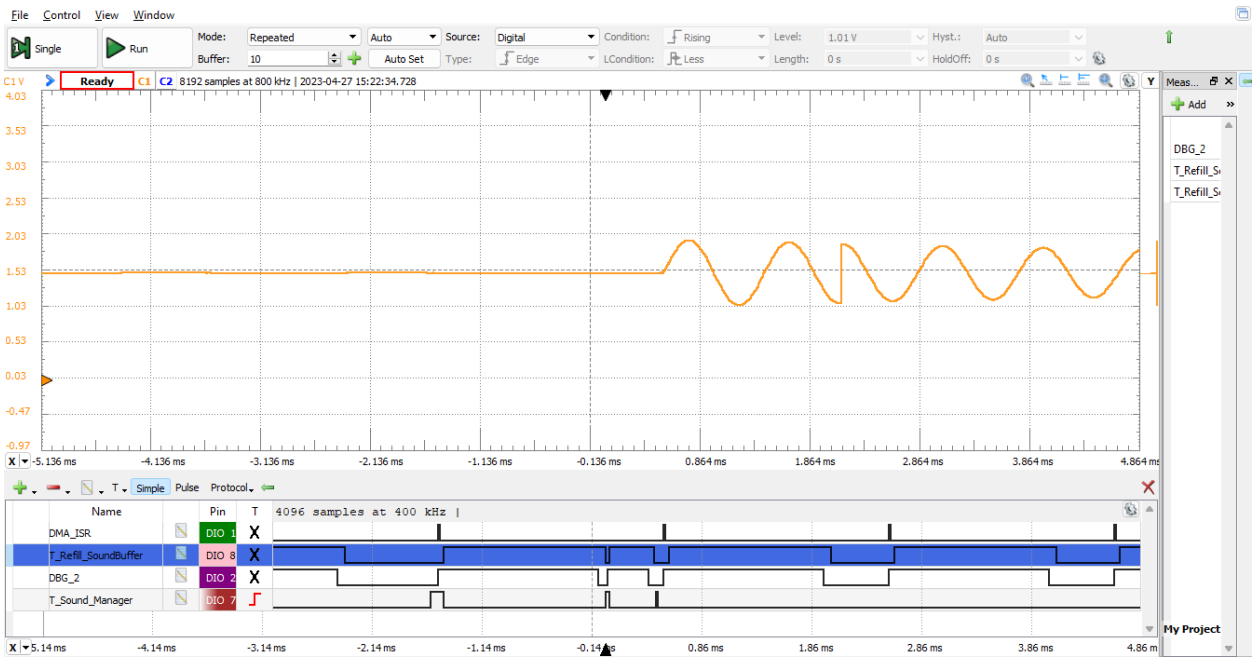
100 kHz: DMA_ISR runs every 2.565 ms. 460 us margin.



120 kHz: DMA_ISR runs every 2.054 ms. 146 us margin



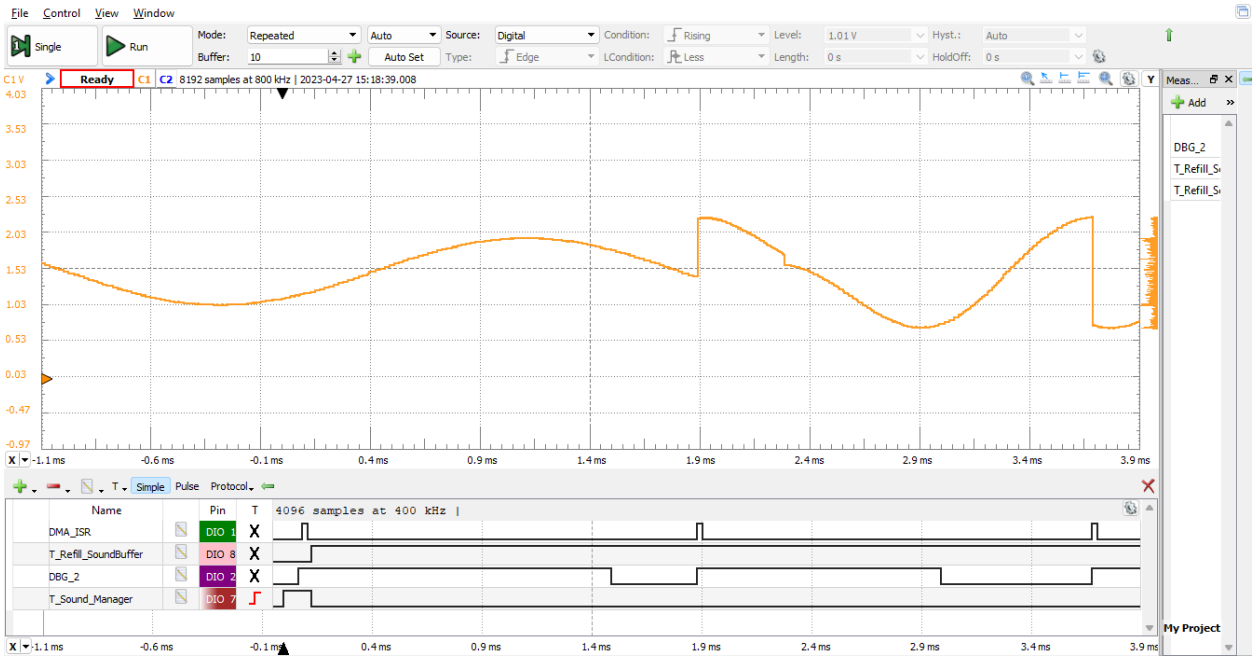
Glitch: Partial Refill writes to wrong buffer?



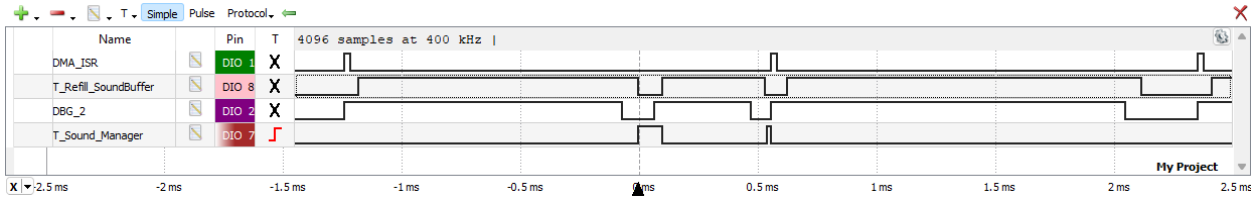
Is partial refill starting at 0 ms related to glitch in buffer starting at 2.144 ms?

130 kHz: 94 us margin

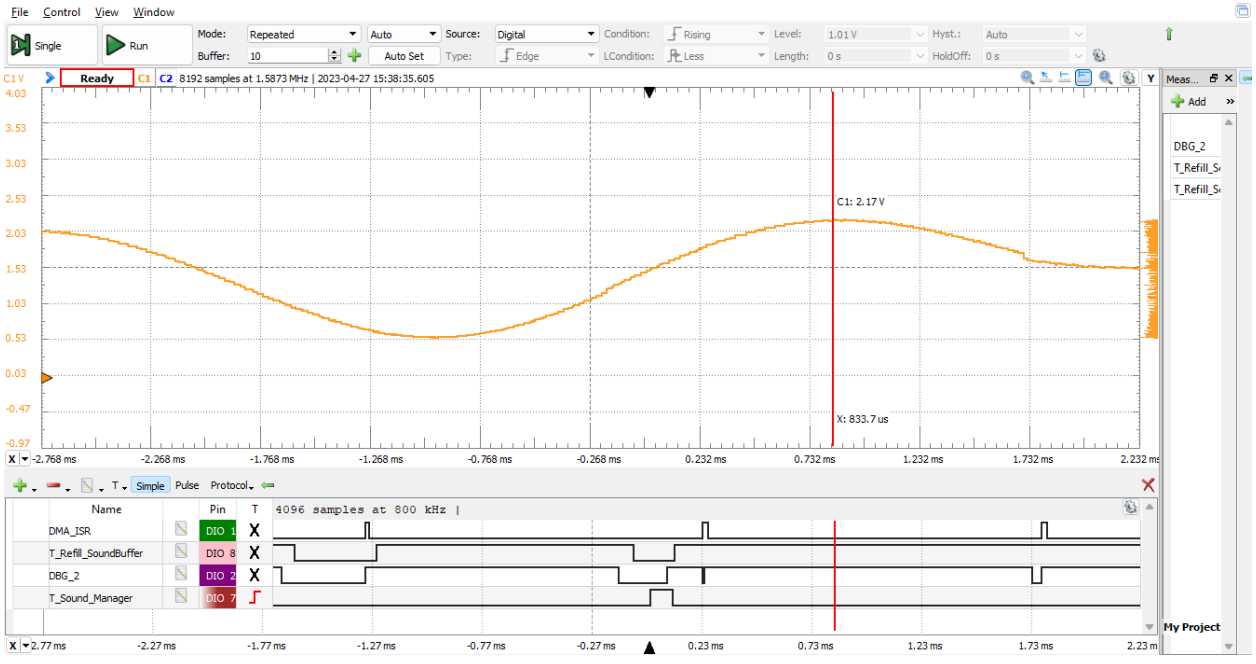
Not enough time for Refill Soundbuffer to complete.



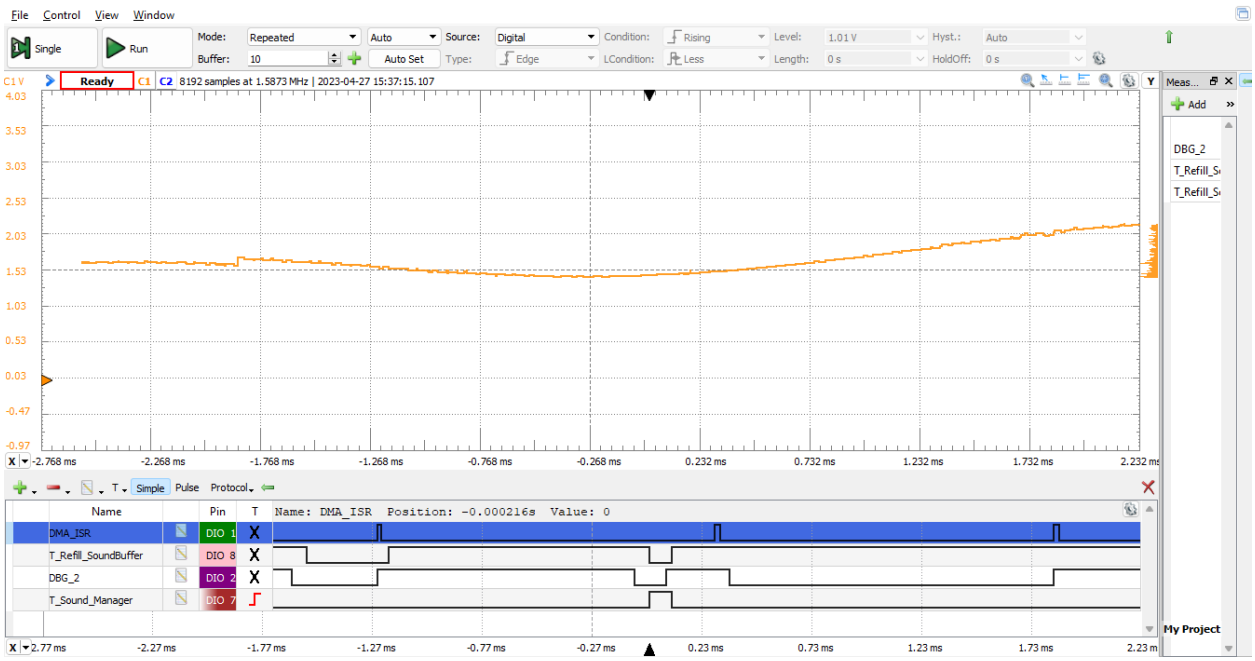
140 kHz: 1.798 ms period, 88 us margin



1.541 ms period, 6 us margin



160 kHz: 1.541 ms period, no margin



- Raise the AUDIO_SAMPLE_FREQ to 10 kHz above your working frequency in order to break the system.
- Switch back to the scope window. Click Reset in the Measurements window to reset the statistics. Start the scope running again.

19. What is the maximum positive pulse width of DIO 2 (i.e. delay to update the first SoundBuffer sample)? (format: 1.23 ms)
    Depends on screenshot.

- Switch to the logic analyzer and set the timeout to a little less than the maximum delay you just measured.

20. Capture the logic analyzer traces in a screenshot and include it in your report.
    SB Refills should take more of CPU's time.
    Expect to see more frequent SB Refills. 20 kHz -> 12.8 ms, 100 kHz -> 2.56 ms, 150 kHz -> 1.7 ms, 200 kHz -> 1.28 ms
21. Which threads and/or ISRs are delaying the refill request in your specific screenshot case? Describe the sequence of events.
    Depends on screenshot.
22. How do their execution times compare with the corresponding maximum execution times (C_i) you provided in Question 1?
    Times should be no larger than C_i above.

OPTIONAL: EVALUATING THREAD VISUALIZATION TIMING ACCURACY

How accurate is the timing of the debug signals? Let's try to determine the timing differences between TV signals and changes in the actual thread execution. Everything from here to the end of this lab is optional and can be done for extra credit.

- Change the definition of AUDIO_SAMPLE_FREQ back to 20,000 Hz (20 kHz) to reduce processor loading and simplify the timing analysis.

DELAY BETWEEN THREAD ACTIVITY AND TV SIGNAL

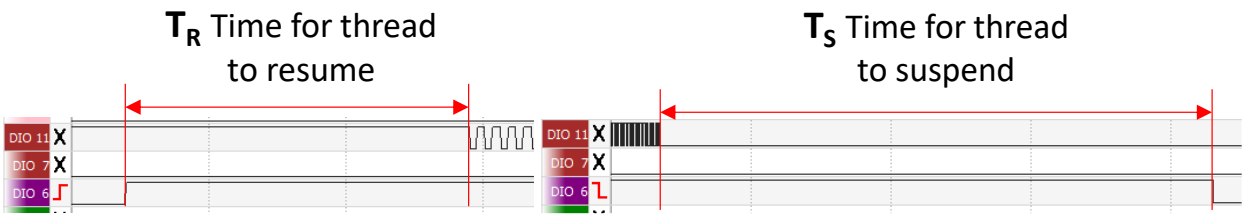


Figure 5. Time delays between TV signal changing and thread resuming, and thread suspending and and TV signal changing.

You'll determine the times shown in Figure 5. First you will measure the delay T_R between a thread resume event (indicated on a TV debug signal) and when the thread actually starts running. We will use the idle thread because when it is running, it is toggling DBG_0 quickly.

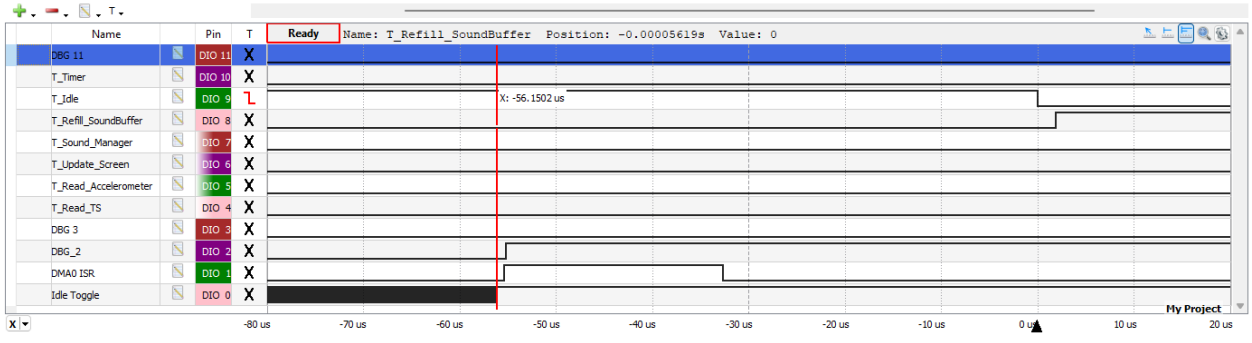
- Use Normal trigger mode, and set the trigger the condition to be the rising edge of the idle thread's TV debug signal (DIO 9).

23. What is the range of delays from triggering (rising edge of DIO 9) until the user debug signal (Idle Toggle, DBG_0/DIO 0) resumes toggling? (Format: 1.23 us – 2.34 us)
    2.4 us - 11.45 us
    There are only two times measured: 2.4 us (when T_Refill_SoundBuffer or T_Sound_Manager blocks) and 11.45 us (when T_Read_TS or T_Update_Screen blocks).

Second, measure the delay T_S between when a thread stops executing and when the TV debug signal indicates it has been preempted.

- Set the trigger condition to be the falling edge of the idle thread's TV debug signal (DIO 9).

24. What is the range of delays from the idle thread user debug signal stopping toggling until the logic analyzer triggers? (Format: 1.23 us – 2.34 us)
    25 us - 56 us
    There are only two times measured: 25 us (normally) and 56 us (when DMA0 ISR executes, and then sets the event flag to run Thread_Refill_SoundBuffer). The RTOS doesn't update the TV signal for the idle thread until just before resuming Thread_Refill_SoundBuffer.



MEASURING CPU AND RTOS TIMING OVERHEADS

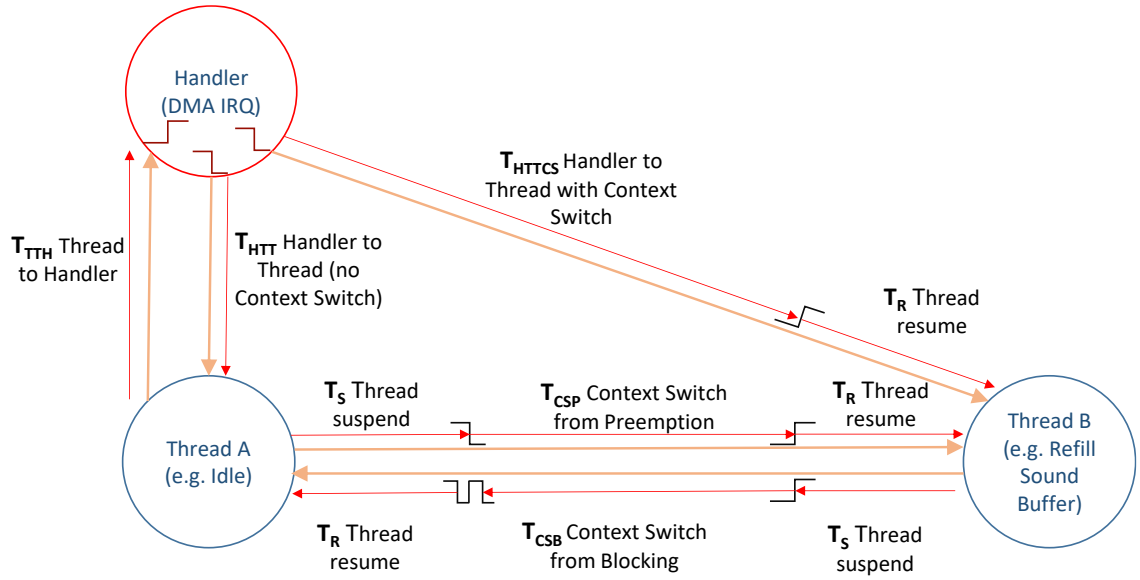


Figure 6. RTOS and CPU time delays to measure.

Next measure how much time the CPU and RTOS use to switch between threads and handlers, and between threads, as shown in Figure 6. **Note that these are measurements which likely underestimate the worst-case times, making any analysis likely to be unsafe. You must add a large margin of safety (ignorance) when applying these values in actual practice.** The two times T_{TTH} and T_{HTT} involve the CPU's interrupt response behavior and are accurate, assuming interrupts have not been disabled.

HANDLER/THREAD INTERACTIONS

- First measure the delay **T_{TTH}** between when a thread stops running and when the body of the ISR handler begins. Use two simultaneous trigger conditions for the logic analyzer: the rising edge of the user debug signal for the DMA IRQ Handler, and the idle thread TV bit being one.

25. What is the range of delays? (Format: 1.23 us – 2.34 us)
    0.72 us – 0.77 us

- Second, measure the delay **T_{HTTCS}** between the body of the DMA IRQ handler ending and the TV debug signal of a new thread rising. Note that the handler will set a thread event flag to request the refilling of the sound buffer. When the handler completes, the scheduler will switch contexts to a different thread.

26. What is the range of delays? (Format: 1.23 us – 2.34 us)
    34 – 38 us

- Third, consider the DMA IRQ handler interrupting the **idle** thread and then letting the idle thread resume. Temporarily comment out the call to osThreadFlagsSet in DMA0_IRQHandler (in DMA.c) to let the idle thread run immediately after the handler. Measure the delay **T_{HTT}** between the body of the DMA IRQ handler ending and the body of the interrupted idle thread resuming. Rebuild and download the project and run the code.

27. What is the range of delays? (Format: 1.23 us – 2.34 us)
    0.38 us – 0.38 us

- Uncomment the call to osThreadFlagsSet in DMA0_IRQHandler to re-enable the code.

THREAD/THREAD INTERACTIONS

- First, measure the delay $\mathbf{T_{CSP}}$ when switching from a thread to any other thread due to **preemption**. Trigger the logic analyzer on the falling single edge of the idle thread's TV debug signal and measure the range of times until the first thread resumes (a TV debug signal rises).

28. What is the range of delays? (Format: 1.23 us – 2.34 us)
    1.85 us – 1.98 us

- Second, measure the delay $\mathbf{T_{CSB}}$ when switching from a thread to any other thread due to **blocking**. Blocking is indicated by the signal falling, rising, and then falling again within a few microseconds. Trigger the logic analyzer on the falling edge of the Read_Accelerometer thread's TV debug signal and measure the range of times from the first falling edge until the first other TV debug signal rises.

29. What is the range of delays? (Format: 1.23 us – 2.34 us)
    4.97 us – 5.17 us
30. How much extra time is taken by the **blocking** case's extra two toggles of the output signal? I.e., what is the time $\mathbf{T_{Toggle}}$ from the first falling edge to the second falling edge? (Format: 1.23 us)
    1.67 us
31. What is the corrected range $\mathbf{T_{CSB}}$ after **s**ubtracting the time $\mathbf{T_{Toggle}}$**?**
    3.3 us – 3.5 us

MEASURING SYSTICK HANDLER COMPUTATION TIME

We haven't instrumented the SysTick timer. We expect it to run at a frequency set in RTX_Config.h (System Configuration). We'll evaluate its behavior by looking at times when the idle thread should be toggling its user debug signal but is not. We'll use Pulse Trigger (instead of Simple) with Timeout to trigger when the idle thread signal has a positive pulse longer than 1 µs.

- Specify the signal source (the idle user debug signal), polarity (positive), and time More than 1 us. The analyzer will trigger when it sees a pulse of the given polarity from the source lasting long enough.

32. For how long does the idle signal stop toggling? Be sure to exclude cases where there is other CPU activity shown on the DMA0 ISR and TV threads. What is the range of times?
    (Format: 1.23 us – 2.34 us)
    4.36 to 4.4 us