CS 7639 HW#5
CPS Variations around the FFT

Question 1:
Both FFT computation and Display blocks activate every 250 ms once system reaches steady state with full buffer

N = 16 samples needed for FFT computation
Each sample acquired every 250 ms
Processing task is activated every 250 ms
16 * 250 ms = 4000 ms
Buffer completely fills after 4000 ms from cold start. Once the buffer is full, the FFT computation runs every processing period (250 ms). Since the Display block activates right after the FFT computation is completed, the period of activation for the Display block will be 250 ms as well.

Question 2:
Conditions necessary for correct sync between each block:

1. Buffer that holds samples needs to be secured when being accessed by multiple blocks to prevent race conditions (11.2.3 in the book). There needs to be a mechanism to prevent the data being read while it's being written and vice versa. Mutexes or message queues can help where mutex provide exclusive access to sample buffer and msg queue to pass data between tasks with built in synchronization

2. The FFT computation block needs to know when there are sufficient samples available (adding condition variables for notification could help).If the sampling speed is faster than processing speed, the system should prevent the buffer overflow. After FFT computation is completed, the display block should then activate. Semaphores can be used to signal when specific numbers of samples are ready.

3. Memory for storing samples and results have to be properly allocated and accessed, and the computation resources have to be scheduled to meet timing requirements. Event flags can be used to indicate when the processing task is complete and when the display task can begin.

In the code currently, there is a message box mechanism msg_box which provides some synchronization, but may needs tweaking fore more robustness

Question 3:

- acq_time = acquisition time or time it takes to sample the input signal
- buffer_time = buffering delay or time that's spent waiting in the buffer period
- process_time = processing time of FFT computation
- display_time = time it takes to render the results and display the outputs

latency = acq_time + buffer_time + process_time + display_time

best-case scenario:
For example, when a sample arrives just before processing task executes, there is minimal buffer delay:
buffer_time = 0
latency = acq_time + 0 + process_time + display_time

worst_case scenario:
If sample arrives just after processing time starts, it has to wait for next processing cycle:
processing_period = 250 ms
buffer_time = processing_period
latency = acq_time + processing_period + process_time + display_time

If sample is first in a new window, then it waits for N-1 more samples to be acquired:
N =16
acquire_period = 250 ms
buffer_time = (N-1) * acquire_period // 15*250 = 3750 ms
latency = acq_time + ((N-1) * acquire_period) + process_time + display_time

Question 4:
fADC = 30 MHz, 15 cycles equivalent to 0.5 us per sample
Under ideal conditions with maximum ADC performance, the minimum time to build 16 samples is 8 us (16 samples * 0.5 us/sample).

Question 5:
Looking at fft() function in the complex.c, the algorithm uses O(N log N) time complexity (times goes up linearly while N goes up exponentially)

For N = 16:
Initial level is 16 samples (N=16)
2 recursive level calls (N=8)
4 recursive level calls (N=4)
8 recursive level calls (N=2)

N = 2^4 = 16

We can count the division and recursive splitting: $\log_2(16) = 4$ levels
At each level, these are the key computations:
Array copying operations: n operations
Sin/Cosine calculations: n / 2 operations
Complex arithmetic (multiplication and addition): 4 * n/2 operations

From the key computations, sin/cos, complex arithmetic and memory operations are the most time-consuming. Sin/cos calculations require several CPU cycles. Complex arithmetic involves 4 floating point multiplications and 2 additions. Memory operations require moving data between arrays.

Since there are 16 samples for the FFT, computation time would be less than the 250 ms sampling period, somewhere in microseconds.

Question 6:
To ensure no samples are lost, the buffer should be sized to handle the worst-case scenario where the processing task is delayed. Minimum size = Processing_Period / Acquired_Period = 250/250 = 1. Since both periods are 250 ms, only one new sample is generated before the processing task runs again. A slightly larger buffer would be safer, somewhere between 2 and 4.

With the msg_box_receive() function, to ensure no messages are lost, the third parameter should be set to True to ensure the processing task waits if no data is available. This helps prevent the processing task from running ahead of available data. If there are multiple messages waiting, process all available messages before returning. The sampling task should use non-blocking sends to avoid it being delayed. Even if the processing task is temporarily delayed, samples will be buffered and processed when the task becomes available.

Question 7:
Tested both methods by commenting out one method.

With the incremental method, time is incremented by fixed value at 250 ms per activation. It provides evenly spaced samples but it doesn't account for processing delays or system load which can lead to a drift from real-world time. Samples appear exactly at ACQUIRE_PERIOD intervals in simulated time.

With the time provided by the OS, it's more accurate to real world wall-clock and doesn't drift over time but there can be system jitter, scheduling variation, and clock resolution limits. Samples may have slightly uneven spacing due to system factors.

In regards to clock accuracy requirements, even sample spacing is important for FFT analysis. Any jitter or uneven spacing creates artifacts in the frequency domain. Maximum acceptable jitter should be smaller than the period of the highest frequency component you want to analyze. With the incremental method, it provides better sample spacing but there is potential long-term drift.

As to the impacts on CPS controllability, there can be timing errors where uneven sampling causes phase errors in the frequency domain and in certain control applications, timing inconsistencies can cause incorrect frequency identification, phase margin errors, and unstable feedback. Clock inaccuracies reduces the system's ability to accurately observe fast changing signals.

Question 8:
FFT results are only displayed after 16 samples are collected, not after every sample
Each fast fourier transform is performed on another 16 new samples

Using this approach, the latency is higher:
In the sliding window, each sample's effect appears in output within one processing period (250 ms)
In the new window, worst case latency for a sample is:
Adding wait time for a full block with the processing time,
((N-1)*ACQUIRE_PERIOD) + PROCESSING_PERIOD
N = 16, ACQUIRE_PERIOD = 250 ms, PROCESSING_PERIOD = 250 ms
15 * 250 ms + 250 ms = 4000 ms
This longer latency is the trade-off for more efficient processing.

Question 9:
Architectural changes:
- Create new task for UART transmission in order to separate processing from communication
- Add data buffering, store the FFT results temporarily until they can be transmitted
- Configure UART hardware, limit the speed since its slower than internal processing, and convert the floating-point values to suitable format for transmission
Scheduling priorities would be from highest to lowest priority where the sampling task is highest and then FFT processing and UART transmission in the end. To manage the resources, use double buffering to allow the processing task to continue while transmitting and create flow control to prevent a buffer overflow.

Question 10:

Justifications for using hardware timers and interrupts for sampling in FFT:
- timing precision: Question 7, timing methods have limitations where internal counters drift from real-world time and OS time has jitter due to scheduling and other factors), looking back at the datasheet of the STM microcontroller, there is hardware timer which provides an accurate clock cycle timing. They continue counting even when CPU is busy and generate interrupts at precise intervals
- managing high sampling rates: Question 4, at maximum ADC speed, sampling happens quickly. 16 samples can be collected in 8 us, microseconds. Hardware timing ensures consistent sampling at high rates (interrupts have predictable low latency and priority based interrupt handling ensures critical operation happen first)
- real time system requirement: in the book section 10.2.1, interrupts provide a clean mechanism to pause normal execution, predictable response times, and automatic context saving and restoring. All these are important systems that have to meet reliable timing deadlines
- resource efficiency: interrupt driver sampling allows CPU to focus on other tasks: CPU doesn't need to check constantly to see if it's time to sample, it can perform FFT between samples, DMA can transfer samples without CPU intervention (10.1.5). This is useful for systems with limited processing power.