Task 2

Frame Transmitter description

FrameTransmitter reads and transmits frames from a binary file according to simulated time. FrameTransmitter is responsible for reading frames from FRAMES.bin, timing frame transmission using the system clock, and yielding flash frames for processing.

It stores a path to the binary file containing the frames, and the system clock object used to coordinate timing.

FrameTransmitter implements the following public functions:

- start_frame_transmission – used to read and transmit frames from the binary file, yielding one frame at a time.

See docstrings for a more detailed description of each function of the module.

Writing Pattern Detector description

WritingPatternDetector detects and logs failures in writing patterns. It monitors the sequence of transmitted frames one by one and checks if the configured failure condition is met. If so, logs the failure to a log file and raises an error.

WritingPatternDetector checks the address of each incoming frame for a failure condition. If any of the frames in a memory write cause a failure, the whole memory write is considered faulty, and none of the frames of this memory write are written to FLASH.

It stores a system clock object to track simulated time, threshold - the maximum allowed number of writes in the pattern, delta - the allowed time window for failure (in seconds), a callback function to log a pattern failure, and a path to the file where a pattern failure is logged.

WritingPatternDetector implements the following public functions:

- init_failure_logger – used to initialize the detector logger to log failures to the log file.

- close_failure_logger – used to close all file handlers of the detector logger and clear the handlers.
- process_incoming_frame – used to process an incoming frame and check if the failure condition is met. If the condition is met, it logs the failure to a file.
- notify_mw_tx_end – used to notify the detector of the memory write transmission end.
- notify_pattern_tx_end – used to notify the detector of the pattern transmission end.
- print_statistics – used by the memory system to log a simulation run's statistics

See docstrings for a more detailed description of each function of the module.

Additional questions

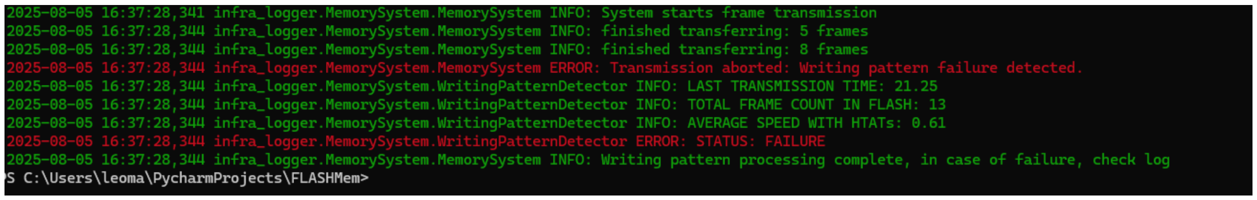
How to test the system?

- The memory system calls self.__detector.print_statistics() at the end of each run. The output can then be compared with the expected results. I added an expected statistics comment to each of the test/example input YAML files:

```
# FAST SEQUENTIAL WRITE ABOVE THRESHOLD
# THRESHOLD: 15
# DELTA: 50

#EXPECTED STATISTICS:
# TOTAL FRAME COUNT IN FLASH: 13 (ONLY MW1 AND MW2 ARE FLUSHED TO FLASH)
# LAST TRANSMISSION ATTEMPT TIME: 21.25 (attempting to transfer frame
number 15)
# AVERAGE SPEED WITH HTATs: 0.61 FPS
# STATUS: FAILURE
```

  Here is the output that I got:



```
2025-08-05 16:37:28,341 infra_logger.MemorySystem.MemorySystem INFO: System starts frame transmission
2025-08-05 16:37:28,344 infra_logger.MemorySystem.MemorySystem INFO: finished transferring: 5 frames
2025-08-05 16:37:28,344 infra_logger.MemorySystem.MemorySystem INFO: finished transferring: 8 frames
2025-08-05 16:37:28,344 infra_logger.MemorySystem.MemorySystem ERROR: Transmission aborted: Writing pattern failure detected.
2025-08-05 16:37:28,344 infra_logger.MemorySystem.WritingPatternDetector INFO: LAST TRANSMISSION TIME: 21.25
2025-08-05 16:37:28,344 infra_logger.MemorySystem.WritingPatternDetector INFO: TOTAL FRAME COUNT IN FLASH: 13
2025-08-05 16:37:28,344 infra_logger.MemorySystem.WritingPatternDetector INFO: AVERAGE SPEED WITH HTATs: 0.61
2025-08-05 16:37:28,344 infra_logger.MemorySystem.WritingPatternDetector ERROR: STATUS: FAILURE
2025-08-05 16:37:28,344 infra_logger.MemorySystem.MemorySystem INFO: Writing pattern processing complete, in case of failure, check log
'S C:\Users\leoma\PycharmProjects\FLASHMem>
```

- By writing unit tests for each function of the FrameTransmitter and WritingPatternDetector.

How would your implementation change in the case of multiple CPUs?

- I would not change my implementation. This implementation is synchronous and deterministic. Increasing the number of simulations running in parallel will not affect the results of the simulations, as they will remain synchronous and deterministic by not having any shared state, not relying on OS time, and not having any asynchronous events.