REPORT:

SIMULATION ANALYSIS OF INTERRUPT HANDLING PERFORMANCE

PART II - DESIGN AND IMPLEMENTATION OF AN INTERRUPTS SIMULATOR

Akshavi Baskaran	101302106
Liam Addie	101315124

Repository Link: *https://github.com/akshavib/SYSC4001_A1.git*

Lab Section: L4

Group #: 15

Due: October 6th, 2025

OBJECTIVE

This report presents an analysis of interrupt handling performance under varying simulation conditions. Key features such as variations in context saving and restoring times, ISR execution time, CPU speed, and other process characteristics are analyzed to understand their impact on the overall efficiency of the interrupt handling process.

IMPLEMENTATION

The simulation was developed by including the `interrupts.hpp` header file and significantly modifying the `interrupts.cpp` file to implement the main logic. Multiple trace files (alike trace.txt), were formulated to model CPU bursts, system calls, and I/O completions. Key parameters were varied and are examined:

- Context save/restore time: 10 ms, 20 ms, 30 ms
- ISR activity duration: 40 ms to 200 ms
- Vector table addresses: 2 bytes and 4 bytes

For each run, the final execution time along with logging statements were recorded in output files (alike execution.txt). To distinguish the program's actual workload from overhead, a Python script was used to calculate the total CPU time separately from the overhead duration. This is attached in the input files called `CPU_efficiency.py`.

1) ANALYSIS FOR VARYING CONTEXT SAVE/RESTORE TIME

Input Files:

- *single_device_trace.txt, multiple_device_trace.txt, mixed_command_trace.txt*

Output Files:

- *context_test10_single.txt, context_test20_single.txt, context_test30_single.txt*
- *context_test10_multiple.txt, context_test20_multiple.txt, context_test30_multiple.txt*
- *context_test10_mixed.txt, context_test20_mixed.txt, context_test30_mixed.txt*

Table 1.0: *Context Save/Restore Time Varying on Different Tracing Files*

	10	**20**	**30**
Single Device	1029 ms	28432 ms	2959 ms
CPU Efficiency (%)	8.7%	8.6%	8.4%
Multiple Devices	28432 ms	29092 ms	29752 ms
CPU Efficiency (%)	12.3%	11.9%	11.7%
Mixed Commands	2959 ms	3019 ms	3079 ms

CPU Efficiency (%)	3.3%	3.2%	3.2%

In the single device trace, the overall runtime increases slightly (by about 20ms) as the context save/restore time rises from 10 to 30 ms. This indicates that few interrupts result in minimal overhead. The multiple device trace shows a steeper rise in runtime due to the high frequency of interrupts. Each additional millisecond of context time accumulates quickly, leading to an increase of approximately 660 ms in total runtime for every 10 ms increment in context time. The mixed command trace lies between the two extremes, demonstrating an increase of runtime by 60 ms.

When comparing CPU efficiencies, the results remain relatively constant across the different context times, suggesting that the proportion of overhead to useful processing time does not change significantly. The CPU efficiency averages around 8% for the single device trace, 12% for the multiple device trace, and 3% for the mixed commands.

Overall, these findings show that while increasing context save/restore time slows overall performance, especially in systems with frequent interrupts, whilst the relative ratio between CPU work and overhead remains nearly constant.

2) ANALYSIS FOR VARYING ISR ACTIVITY TIME

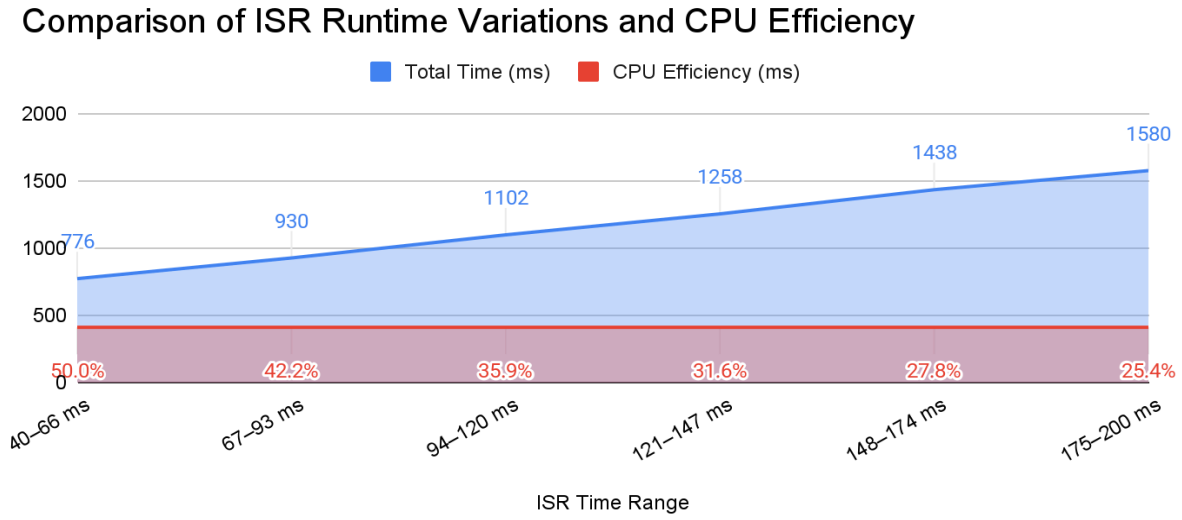
Input Files:

- *device_table_v2.txt, low_isr_trace.txt, low_medium_isr_trace.txt, medium_isr_trace.txt, medium_high_isr_trace.txt, high_isr_trace.txt, very_high_isr_trace.txt*

Output Files:

- *low.txt, low_medium.txt, medium.txt, medium_high.txt, high.txt, very_high.txt*

Table 2.0: *Comparison Graph of ISR Runtime Variations and CPU Efficiency*



As the ISR execution time increases, the system's total runtime rises steadily by approximately 10–20%. This was analyzed by changing the device delay times to range from 40 ms - 200 ms. Longer ISR activity periods delay the CPU from returning to normal processing. In the 40–66 ms range, CPU efficiency is around 50%, meaning half of the total runtime is productive CPU work. However, as ISR activity time continues to increase toward 200 ms, efficiency gradually declines to just 25%. At this point, only one-quarter of the total runtime is dedicated to actual processing, while the remaining 75% is spent handling interrupts. This indicates that when an ISR activity takes too long, there is a significant loss in system efficiency.

3) ANALYSIS FOR VARYING VECTOR ENTRY SIZE

Input Files:

- *single_device_tracev2.txt, multiple_device_tracev2.txt, mixed_command_tracev2.txt*

Output Files:

- *vector_2byte_single.txt, vector_4byte_single.txt*
- *vector_2byte_multiple.txt, vector_4byte_multiple.txt*
- *vector_2byte_mixed.txt, vector_4byte_mixed.txt*

Table 3.0: *Comparing Memory Addresses for Vector Tables in Simulation*

	VECTOR_SIZE = 2	**VECTOR_SIZE = 4**
Single Device	0x000A	0x0014
Multiple Devices	0x0004, 0x0010, 0x0018	0x0008, 0x0020, 0x0030
Mixed Commands	0x0006, 0x0012, 0x001E	0x00C, 0x0024, 0x003C

Referring to the table above, altering the vector size from 2 to 4 bytes only changes the memory address spacing for the interrupt vectors. Each vector entry now occupies twice the space, which means the memory offset between consecutive vectors doubles. This change affects how the system locates the appropriate ISR address in memory but does not impact the actual execution flow or performance, as the ISR logic and timing remain unchanged (seen in the execution files).

CONCLUSION

The simulation shows that longer context save times and ISR durations increase total runtime. The ISR length has a great impact on CPU efficiency as well. Even more, vector entry sizes affect the memory layout, however not the performance. All in all, minimizing ISR time and context switching overhead is key to efficient interrupt handling.