SYSC4001 A1 Report
Repository Link: https://github.com/Xelatrz/SYSC4001_A1.git

Table 1. Simulation results from interrupt simulation tests.

Test #	Input	Context Time (ms)	ISR Activity Time (ms)	Total Time (ms)	Overhead Time (ms)	User Time (ms)	Output
1	trace_1.txt	10	40	1148	360	788	execution_1_40_10.txt
2	trace_1.txt	20	40	1298	510	788	execution_1_40_20.txt
3	trace_1.txt	30	40	1448	660	788	execution_1_40_30.txt
4	trace_1.txt	10	80	1348	560	788	execution_1_80_10.txt
5	trace_1.txt	20	80	1498	710	788	execution_1_80_20.txt
6	trace_1.txt	30	80	1648	860	788	execution_1_80_30.txt
7	trace_1.txt	10	150	1698	910	788	execution_1_150_10.txt
8	trace_1.txt	20	150	1848	1060	788	execution_1_150_20.txt
9	trace_1.txt	30	150	1998	1210	788	execution_1_150_30.txt
10	trace_1.txt	10	200	1948	1160	788	execution_1_200_10.txt
11	trace_1.txt	20	200	2098	1310	788	execution_1_200_20.txt
12	trace_1.txt	30	200	2248	1460	788	execution_1_200_30.txt
13	trace_2.txt	10	40	630	216	414	execution_2_40_10.txt
14	trace_2.txt	20	40	720	306	414	execution_2_40_20.txt
15	trace_2.txt	30	40	810	396	414	execution_2_40_30.txt
16	trace_2.txt	10	80	750	336	414	execution_2_80_10.txt
17	trace_2.txt	20	80	840	426	414	execution_2_80_20.txt
18	trace_2.txt	30	80	930	516	414	execution_2_80_30.txt
19	trace_2.txt	10	150	960	546	414	execution_2_150_10.txt
20	trace_2.txt	20	150	1050	636	414	execution_2_150_20.txt
21	trace_2.txt	30	150	1140	726	414	execution_2_150_30.txt
22	trace_2.txt	10	200	1110	696	414	execution_2_200_10.txt

23	trace_2.txt	20	200	1200	786	414	execution_2_200_20.txt
24	trace_2.txt	30	200	1290	876	414	execution_2_200_30.txt
25	trace_3.txt	10	40	5795	2304	3491	execution_3_40_10.txt
26	trace_4.txt	10	40	6099	2232	3867	execution_4_40_10.txt
27	trace_5.txt	10	40	6406	2448	3958	execution_5_40_10.txt
28	student_trace_1.txt	10	40	943	228	655	execution_s1_40_10.txt
29	student_trace_2.txt	10	40	1393	576	817	execution_s2_40_10.txt

As demonstrated in Table 1. varying the save/restore context time doesn't affect the user time, however it does affect the overhead time which represents all the time doing anything other than executing user application code through the CPU. This means saving/restoring context happens outside of the application code, among other operations such as setup/teardown for the application code. Table 1. demonstrates this connection in, for example, test 1, 2, and 3, (among many other tests proving consistency). Tests 1 through 3 all have the same input and ISR activity time, with the only difference in the tests being the change in save/restore context time. It can be observed that with this being the only difference, the user time remains the same, and the overhead time increases (which in turn also increases total time). A similar trait can be observed for the ISR activity times in tests 1, 4, 7, and 10. In these tests the only change is the ISR activity time, and the same trait occurs by increasing overhead time but not user time. The significance of these two realizations is that these are run outside of the user application code, and that increasing these times significantly increases the overall runtime. This is important to know, because optimizing these times could save some time and money in the long run. If the ISR activity time is too long it will be waiting an unnecessary amount of time before noticing an interrupt signal, meaning if the program doesn't require that much time to execute, you're simply wasting time and power.

Now that the effect of the context and ISR activity times have been discussed, it's important to take a look at other aspects, such as CPU times. If a faster CPU is used, it will surely lower the overall program time, but how? It will do this by executing the user application code faster, speeding up the user application time, and in turn affecting the overall runtime.

If a device was called that doesn't exist, what would happen? Normally, the attempt to access a device outside the index range of the device table would throw an error and terminate the program. That's why, to avoid the termination of a program, if a device is out of index a statement can be added to print a warning and skip that device, resulting in no abrupt terminations and a clear, understandable warning message. The same can be done preventatively for the vector table, adding a statement in interrupts.cpp before calling the vector table to assure the index is not out of range.

Another curiosity to touch on is what would happen if addresses of 4 bytes are used instead of 2? Since each vector address is now at 4 bytes instead of 2 bytes, the address will be shifted to different memory locations that are further apart from one another, effectively doubling the size of the vector table. This would make it take up more memory storage, and potentially increase runtime. This is why it's ideal to keep the address length as short as possible, and only increase it if absolutely necessary.