IN4343 – Lab 2

Casper van Wezel (4209192) & Erwin de Haan (4222814) – 2017-03-11

# Part 1

## Overhead of timer handler

Hypothesis: The time will increase linearly as a function of NUMTASKS, due

1. This data was produced using a script the takes the average for every period of a time of 5 seconds. This is more than five thousand periods.

|  |  |
| --- | --- |
| **NUMTASKS** | **Time (us)** |
| 1 | 41,3 |
| 2 | 120 |
| 3 | 157 |
| 4 | 196 |
| 5 | 232 |
| 6 | 271 |
| 7 | 308 |
| 8 | 345 |
| 9 | 381 |
| 10 | 420 |
| 15 | 607 |
| 20 | 793 |
| 24 | 942 |
| 25 | 980 |

2. The period of the timer is 976,5 us. So 25 tasks is the threshold value (see table above).

3. The compiler will optimize the loop out of the final machine code. This means a lot of code can be dropped for NUMTASKS = 1

4. Make the NUMTASKS equal to the number of registered tasks. In the case of Tst1 3. Change the clock ticks per second to 2 and divide the periods of the tasks by 512.

## Event latency

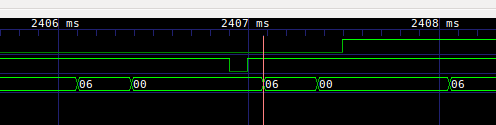
5.

This data was produced using a script that measured over 11 events the delay between the intr\_num rising edge and the “green” rising edge. The CountDelay was removed and only the Green task was registered.

|  |  |  |
| --- | --- | --- |
| **PRIORITY** | **Time (us)** | **Time (us) Lower** |
| 0 | 261 | 51 |
| 1 | 224 | 51 |
| 2 | 187 | 47 |
| 3 | 149 | 51 |
| 4 | 112 | 47 |

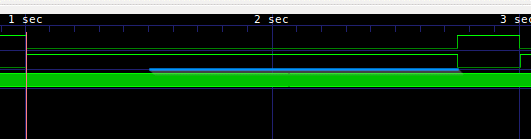
An “empty” interrupt handler takes 250 us, this was measured in the same way as for question one, but one task was registered. The ISR with one Task activation was measured to take 297 or 301 us.

6. There now is a Program loop, while in the previous Test there was none. The LED are turned on much later, than the actually interrupt fires, in order or priority. There are some artifacts: (This is due to CountDelay and the task getting re-executed right after.



7ab. Base ISR time are actually longer, although much more consistent, and the ISR’s that trigger a task are also longer, but relatively not quite as bad. But the best thing, is that the interrupts keep running.

8. The delay that CountDelay(60000) represents is about 1.738-0.5=1.238 seconds.



9a. The calculated length would be 1219ms

640ms represents 655.36 ISR executions, that adds 304 ms of execution time, this adds 311 ISR executions. We did this iteratively until sufficient precision was achieved. Table used is below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Duration (ms)** | **ISR time (ms)** | **Intr Period (ms)** | **# Intr's** |
| 640 | 0,464 | 0,9765625 | 655,36 |
| 304,08704 |  |  | 311,3851 |
| 144,4826998 |  |  | 147,9503 |
| 68,64893207 |  |  | 70,29651 |
| 32,61757899 |  |  | 33,4004 |
| 15,49778601 |  |  | 15,86973 |
| 7,363556054 |  |  | 7,540281 |
| 3,498690569 |  |  | 3,582659 |
| 1,662353842 |  |  | 1,70225 |
| 0,789844155 |  |  | 0,8088 |
| 0,375283393 |  |  | 0,38429 |
| 0,17831065 |  |  | 0,18259 |
| 0,084721809 |  |  | 0,086755 |

The final time is the sum of all values in the first columns.

9b. (Sum of all elements in last column) 1248 interrupts are interfering with the execution of CountDelay.

(Answer of 9a minus 640ms) 579 ms

9c. The measured time is greater, so the calculated value is not a valid upped bound.

10. The measured time for the ISR is 465 us. These values do not differ significantly. But the small increase over many executions can be the reason for the 1238-1219 = 19 ms difference. This could be done due to clock inaccuracies and system calls (like in lab 1)

11. The assignment of t->Invoked = t->Activated does not really have to happen everytime the loop hit that condition. The other bigger improvement would be to switch the two if statements. That would drop the need for the second if’s else statement (The assignment mentions before). Those values can be set when a task is registered.

12. The BlinkYellow executes last, this is because even though it has a higher priority the HandleTask does not look for other tasks when it is processing all the queued up Tasks of one type. It would need to start it’s loop again to start at the top of the priority queue.

13. Add a flag when a task is activated and put the flag in the while loop negated. Reset it before the inner while loop.