UNIVERSITY OF WATERLOO

ECE750 Real-time Embedded System

Lab #5

Name: Shaocong Ren

Student number: 20478300

E-mail: s7ren@uwaterloo.ca

Q1. Describe the behavior of the interrupt schedulers under a normal interrupt load (i.e. non-interrupt storm). Do interrupt schedulers have any effect on the overall program under normal interrupt loads?

Under a normal interrupt load, there is no distinguish between schedulers. Since all the interrupts are responded. And there is no effect on the overall program.

For the strict scheduler, after each ISR there should be a 50ms idle time during which all the interrupts are ignored. While the interval of normal interrupts is greater than 50ms, so we can say zero interrupt is disabled.

For the bursty scheduler, there should be less than or equal to 5 interrupts in each 250ms are responded, the other interrupts that in the same duration arrive later than the 5^{th} should be ignored. While for the normal interrupt load, there are no more than 5 interrupts per 250ms from beginning. So that bursty scheduler does not affect the normal interrupt load.

Q2. Describe the behavior of the strict and bursty schedulers during a continuous interrupt storm. What is different between the two different schedulers?

For the strict scheduler, it enforces a minimum interarrival time between interrupts. After each ISR there should be a 50ms idle time during which all the interrupts are disabled. The result of this scheduler is program responses interrupts every 50ms roughly under the continuous interrupt storm.

For the bursty scheduler, there should be less than or equal to 5 (burst size) interrupts in each 250ms (maximum arrival rate) are responded, the other interrupts that in the same duration arrive later than the 5^{th} should be ignored. The result of bursty scheduler during a continuous interrupt storm is that during first short time of each 250ms (maximum arrival rate) there are 5 (burst size) interrupts responded, then it holds a long idle time until the end of the 250ms (maximum arrival rate).

The phenomena of these two schedulers are different. Bursty scheduler is lazier, disabling an interrupt only after a burst of interrupt requests has been observed. And bursty scheduler has lower overhead than the strict scheduler but provides weaker isolation.

Q3. In what situations is a strict scheduler preferable? In what situations is a bursty scheduler preferable? For each scheduler, give one example of a practical application that is well-suited to the scheduler.

When the interrupt arrive at a high but stable rate and there is no strict constraint on the cost of the program strict scheduler is preferable. And the program request avoid a long idle time. Since in this way the program can monitor most of the emergency, rather than handling first few interrupts and holding a long idle time. For example, strict scheduler can be used on the real-time temperature monitoring system to take temperature samples in a permanent frequency. This can avoid losing some samples in a long period.

When a program has strict constraints on the overhead, or the feature of interrupts is bursty then bursty scheduler is preferable since it can handle more interrupts than other scheduler on a lower cost. For example, the network interfaces, are inherently bursty. It may be desirable to attempt to handle an entire burst, rather than handling only the first interrupt in a burst, or the first few, and dropping the rest, as the strict scheduler would do.

Q4. Explain how you test your project for compliance to the posted MISRA-C rules. Highlight, how you identified non-compliant code that you used from the STM32 Peripheral Library.

For most of the MISRA-C rules can be detected by the warnings when compiling the project if we set parameters "-*Wall*" and "-*pedantic*". For example, "The character sequence /* and // shall not be used within a Comment" and "There shall be no unused parameters in definitions." Some of the rules tested manually, such as "Line-splicing shall not be used in // comments" and "A "u" or "U" suffix shall be applied to all integer constants that are represented in an unsigned type."

As far as the non-compliant code I use the following method to detect. For a certain *.c file in the project modifying its filename and using these three commands ***"make clean", "make", "make program",*** if the program runs well then the whole file is not used in the project. Then for the file used in the project going into the file commenting a certain function or a variable and using the three commands again, if it runs well the function or variable is not used in the project. If it pops up error then I roll back the code. From the two steps, I detect most of the dead code. Then filter the project again in the same way, it can be sure almost all the dead code is cleared.