1. Worst-case execution time (WCET) of Hough transform is the maximum length of time the ATmega32U4 could take to execute Hough transform code. It’s usually performed to ensure meeting deadlines.

To determine this time, we are making use of the global time keeping variable “time\_ms”. The time before the execution has been passed on to the variable name “\_time\_start” and the control is passed to Hough transform task. After the completion of the task the current time at the global time keeping task has been passed on to “\_time\_end”. Since now we have the time before the execution, “\_time\_start” and the time after the execution “\_time\_end”, we can easily calculate the amount of time the code has taken to execute. This will give us the approximate WCET.

Average amount of time Hough transform took to execute is 51ms and which varies very cycle with some difference and I am confident that this is true with 10% margin since this method involve approximations.

Factors influencing WCET:

1. Interrupts- This time will change in case an interruptions occurs in between the execution of the task. This will lead to intermittent large increases in the end-to-end execution time of a program, and this effect needs to be identified and compensated
2. Complexity of the hardware
3. Since all the three tasks – Executing Hough transform, RED LED, JITTER LED and YELLOW LED tasks are executed in main, it is not possible to complete the tasks as per the deadline.

Since the Hough transform takes around 200ms to, the RED and JITTER LED tasks are delayed. Specifically, after introducing Hough transform task, RED LED and the JITTER LED tasks are delayed significantly. Due to this almost every deadline of these tasks are missed.

But due to the fact that YELLOW LED tasks are executed inside the ISR, it executes independently without any interference of other task, it meets the required deadline.

1. Experiment #1
   1. With the frequency of 2Hz, there was a slight delay in the bilking of the RED LED after 10-15 secs compared to the GREEN LED which showed no change in the behavior. This drift may be due to the fact the RED LED is part of the main which is executing high intensity task such as Hough Transform which might cause a small delay in the execution of RED LED task.
   2. However, in the beginning, all the LED’s were blinking in a sync. This is due to the fact that the combined period of all the events were less than the execution time of the code so it made sure that the all the LEDs med their deadline and were blink as per the specification.

Following results are obtained

Worst case Execution time of Hough Transform in ms: 51

Number of Green LED toggles: 64

Number of Yellow LED toggles: 63

Number of missed deadlines - GREEN LED: 1

Number of missed deadlines - YELLOW LED: 0

Number of missed deadlines - JITTER LED: 1

1. Experiment #2

Initially, without any busy wait delay, RED, YELLOW and GREEN GPIO LEDs were blinking almost in a sink and also JITTER and RED LED’s missed less number of deadlines.

Worst case Execution time of Hough Transform in ms: 51

Number of Green LED toggles: 64

Number of Yellow LED toggles: 63

Number of missed deadlines - GREEN LED: 1

Number of missed deadlines - YELLOW LED: 0

Number of missed deadlines - JITTER LED: 13

Please note that, for a given situation it is very difficult to calculate the number of missed deadlines of RED LED since Current time is not tracked correctly due to delay in other ISR. So in the report, I have commented only on the behavior of RED LED instead of the exact number of missed deadlines.

After introducing a 20ms busy wait in the ISR of the GREEN LED counting task, GREEN LED starts first and after a small duration of time GREEN and RED LED’s start. So they are not in sink compared to the earlier case and this time, JITTER missed more number of deadlines compared to the earlier case where there was no busy wait delay.

Worst case Execution time of Hough Transform in ms: 51

Number of Green LED toggles: 54

Number of Yellow LED toggles: 53

Number of missed deadlines - GREEN LED: 11

Number of missed deadlines - YELLOW LED: 10

Number of missed deadlines - JITTER LED: 17

After introducing a 20ms busy wait in the ISR of the YELLOW LED counting task, JITTER LED missed many more deadlines compared to other results and so as the GREEN and YELLOW LEDs. Also RED LED started blinking slowly compared to the previous result.

Worst case Execution time of Hough Transform in ms: 60

Number of Green LED toggles: 95

Number of Yellow LED toggles: 94

Number of missed deadlines - GREEN LED: 73

Number of missed deadlines - YELLOW LED: 72

Number of missed deadlines - JITTER LED: 60

Similarly with 30ms busy wait in the ISR of the GREEN LED counting task, GREEN, YELLOW and JITTER LED’s missed more number of tasks.

Worst case Execution time of Hough Transform in ms :52

Number of Green LED toggles: 68

Number of Yellow LED toggles: 50

Number of missed deadlines - GREEN LED: 21

Number of missed deadlines - YELLOW LED: 3

Number of missed deadlines - JITTER LED: 15

Similarly with 30ms busy wait in the ISR of the YELLOW LED counting task, JITTER and RED LED’s whose statuses are changed in the main, as per the flag set in the ISR, almost stopped blinking.

And finally, with 105ms busy wait in the ISR of the GREEN LED counting task, only GREEN LED connected to GPIO continued to blink but the other LEDs stopped blinking.

With 105ms busy wait in the ISR of the YELLOW LED counting task, GREEN LED continued to blink and also YELLOW LED but slowly compared to the GREEN LEDs.

The both the above cases, since the main function, does not get the chance to execute, the JITTER and RED LED status is not changed and hence they stopped blinking. Additionally, the menu task, which is placed in the main function, could not be executed.

In summary, due to the busy wait, main function does not get a chance to execute and hence the blinking action performed in the main is not been executed and in turn these LEDs (RED and JITTER) stops blinking.

1. With 105ms busy wait in YELLOW LED task and by placing sei() at the top of ISR before the toggle and delay, we observe following behavior of the system.

In the previous experiment, where while not placing sei() at the top of ISR before the toggle and the delay, we observe that the YELLOW led missed far many deadlines and not many but few tasks executed in main. In particular JITTER LED GLOW once in the beginning of the execution. This is due to the fact that after executing the interrupt, control will always return to the main program and execute one more instruction before any pending interrupt is served.

But by setting sei() at the top of ISR before toggle and delay, we observe that YELLOW LED task picked up and missed very few deadline compared to the earlier case. Whereas no tasks executed in the in the main function. Since we are setting sei() after entering the interrupt, it services the other interrupt before returning to the main function. And hence both YELLOW and GREEN LED’s meet more deadlines compared to earlier case.

1. From the experiment we can answer following questions:
   1. Time driven execution inside the ISR

It is one of the reliable method to keep track of the timing and not to miss the deadline. Since the code in the main program stops in order to service an ISR, compared to the other tasks inside the main, system will be more responsive and system handles them with higher priority compared to the tasks inside the ISR.

* 1. Time driven release from within an ISR

Even though time driven release from within an ISR, gives a control at the appropriate time, again the execution within the correct deadline depends on the intensity (load) with which the main task is running.

* 1. External Interrupt with Execution Inside ISE

As we have observed in the experiment # 6 and 7 with the delay of 105ms inside the ISR, the main task did not get executed. So in this scenario, it becomes impossible for the user to operate the system. In such scenario, to execute the menu task, it is useful to program them as an external interrupt using ISR and so that the user command is executed.

* 1. Periodic polling for an event from within an ISR, then release the task

From the experiment we can conclude that, this method likely to result in more missed deadlines than time driven execution inside an ISR. Even though the even might have occurred while executing the main part of the code, this is realized by the program only when it polls for a value. Also again, since the task is to be executed is a part of main function, the deadline and its execution depends on the loading of the task.

Note: Please read “read me.txt file” on how to operate the system and the connections.