ECE 224 – LAB 1

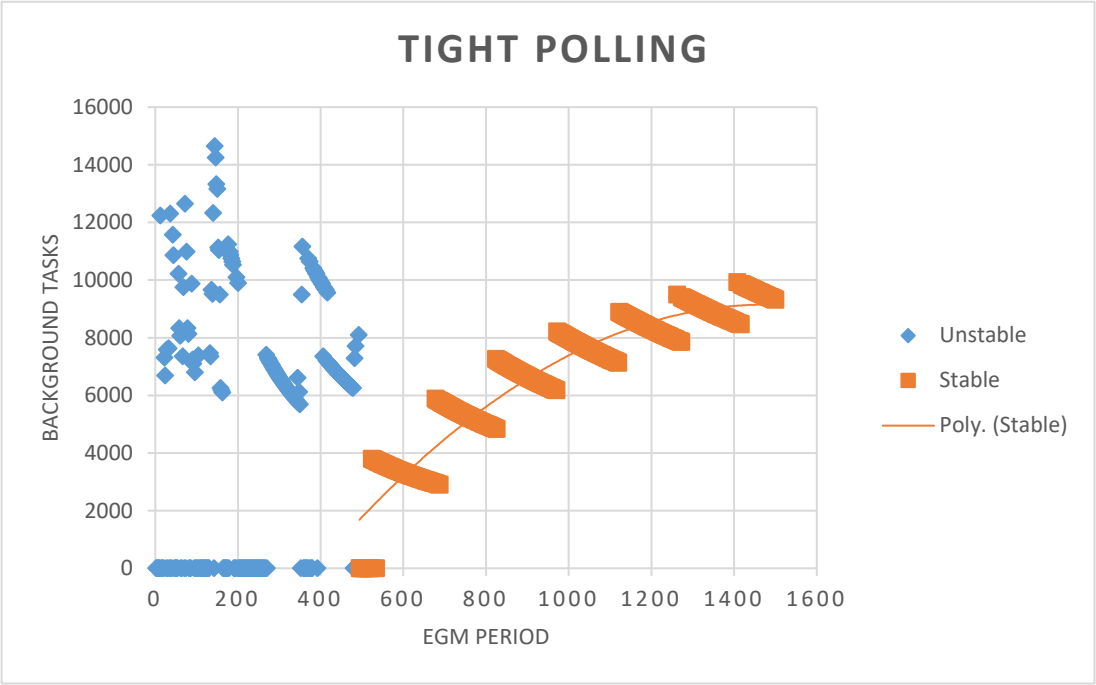
Daivik Goel and Aiden Bishop Wood

OCTOBER 18, 2018

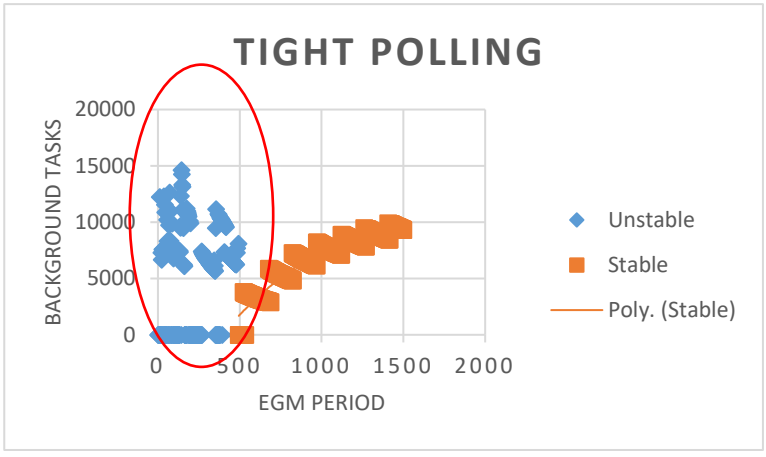
GROUP 11

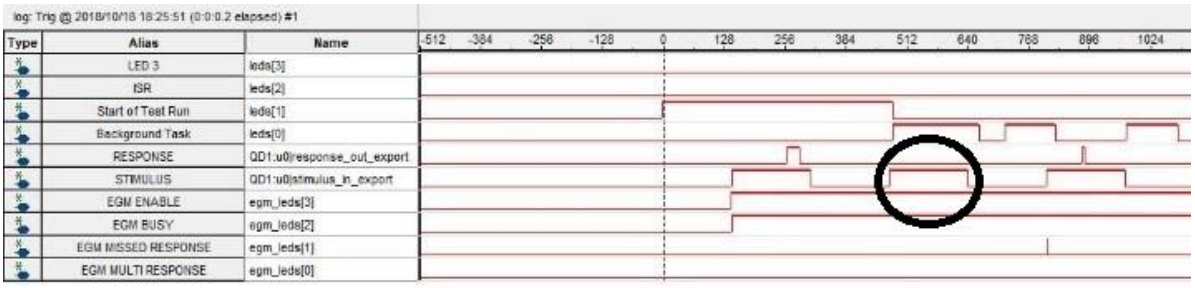
ECE 224

Tight Polling Graph Analysis



This is the tight polling graph relating the EGM Period to the Background Tasks between 2-1500 clock cycles. As we can see when the EGM Period is between 0 and approximately 500, the background tasks experience a high level of variation between =<6000 and 0.

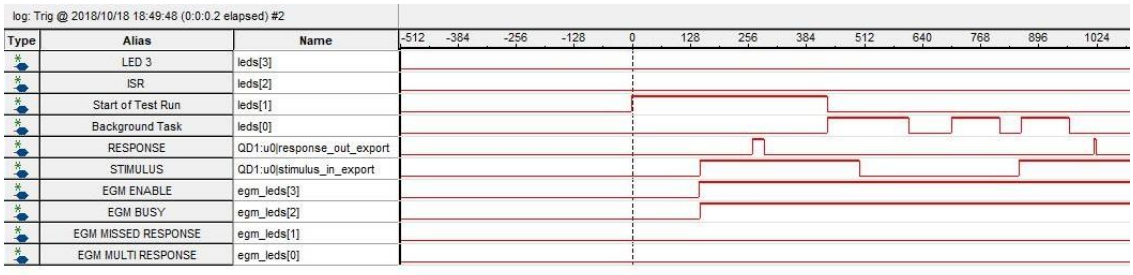


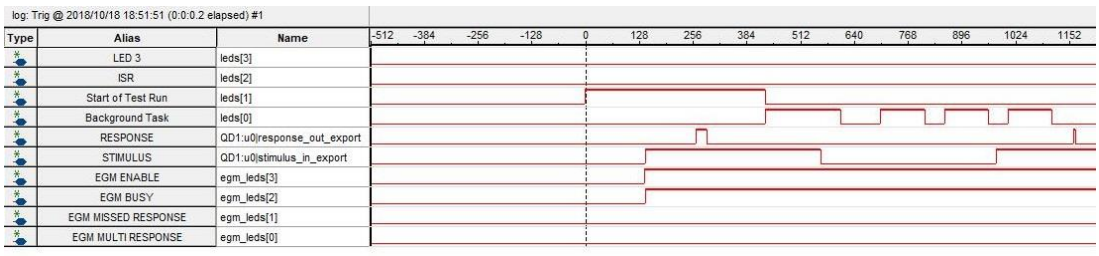


The unstable portion of the plot is due to the fact that one background task can be larger than a single period of the EGM stimulus. After one background task has run when the stimulus is checked, the data is low once again (after multiple stimulus periods have occurred) which causes the logic to assume we are still within the bounds of one period and a background task is run once again. This can occur multiple times until when stimulus is checked the data is high. This causes an invalid number of backgrounds to be recorded and the characterization fails. Above can be seen a stimulus high being generated by the EGM that is entirely missed due to a background task being run (due to a failed characterization). After the period is larger than the amount of time elapsed during one background cycle (and the overhead involved with the running the background cycle) the characterization can correctly occur and the stable portion of the graph begins.

The overall trend line on the stable portion of the graph increases in an asymptotic fashion due to the following. The fraction of time spent running background tasks over total time spent in each test run approaches the asymptote of 1 (all time spent running background tasks) as the total number of stimulus' to respond to decreases (and therefore the total time spent responding goes down).

The discontinuities occur due to there being a specific value upon which 1 more background task can be run. Any lower than this value and all the time between the last background task running and the end of the stimulus period is wasted. Therefore upon the period surpassing that specific value one more background task can be run resulting in a discontinuity or "jump" in the graph. The negative sloped portions between discontinuities is due to period of the stimulus increasing but another background task being unable to run in the time between the last background task running and the end of the stimulus.

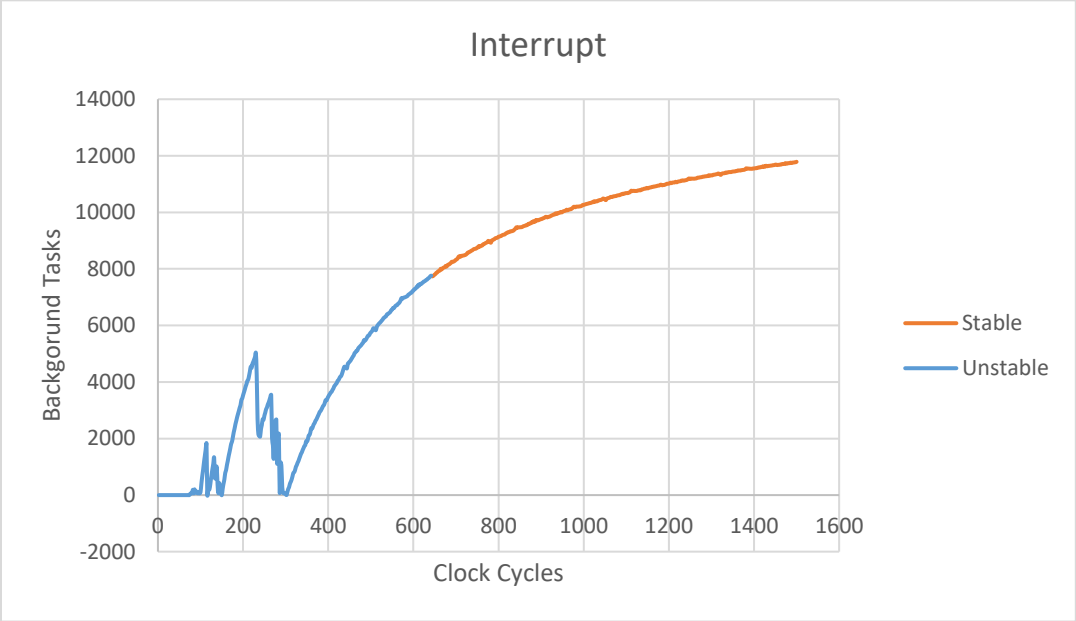




Therefore in each negative slope the same number of background tasks is being run however the length of the period increases (and therefore the overall number of stimulus periods per test run decreases), causing

a negative slope. Below is a signal with 3 background, and one with 4 backgrounds elapsed in the stimulus period.

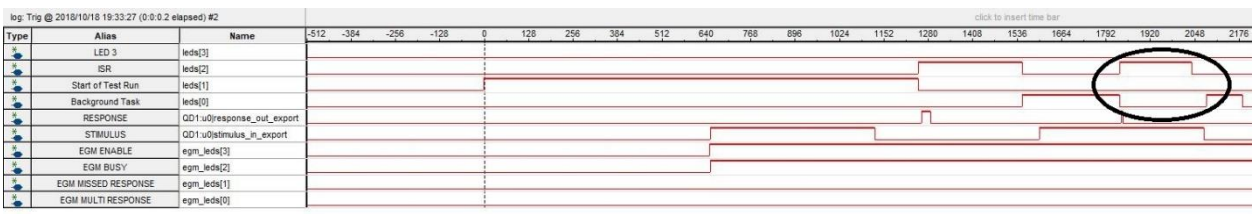
Interrupt



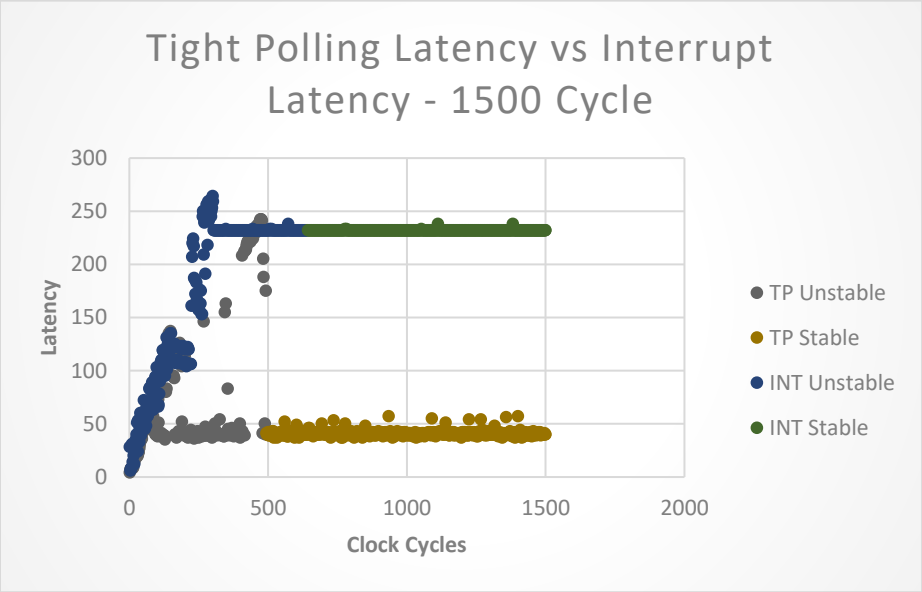
The missed pulses for the interrupt part of the graph are due the large latency caused by the interrupt. When the period of the stimulus is smaller than the latency of the interrupt then a pulse can be missed because multiple periods can have elapsed before the ISR has run and responded to a stimulus.

The overall trend line on the stable portion of the graph increases in an asymptotic fashion due to the following. The fraction of time spent running background tasks over total time spent in each test run approaches the asymptote of 1 (all time spent running background tasks) as the total number of calls to the interrupt decreases (and therefore total time spent interrupting decreases).

The background task efficiency is very high due to the background running 100% of the time in which it is not being interrupted and unlike the tight poll there is no waiting time caused by a gap between the last background task running and the end of the stimulus period. Below in the circled portion is an example of a background task being interrupted and then resumed.

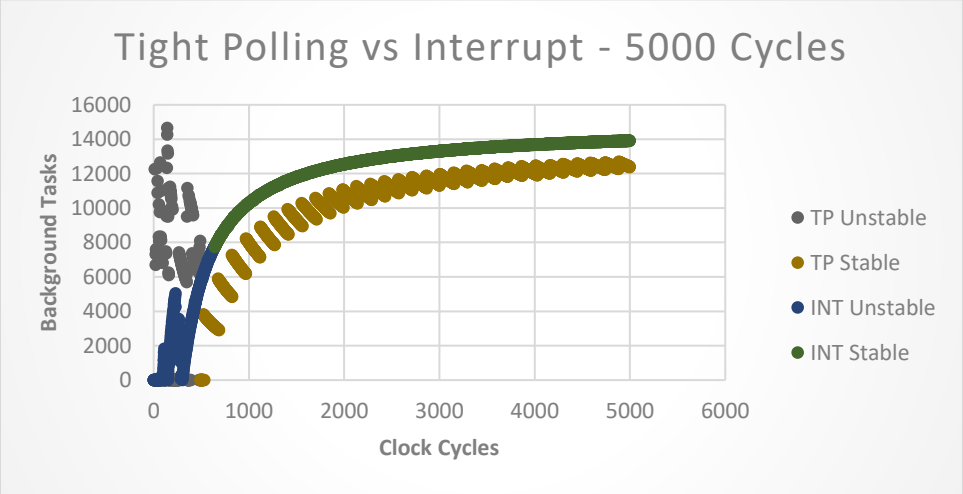


Latency



As can be seen in the graph above the latency is significantly lower for the tight polling approach, this is due to interrupts taking a significantly longer portion of time due to switching execution modes, popping and writing to stack etc. It can be also be seen that tight polling reaches the stable state before the interrupt does, another advantage. The latency of interrupt is nearly completely constant once the stable state is reached; this is very advantageous is some scenarios where a minimum time for an operation has be maintained. The constant nature of the time is because it does not rely upon any form of polling and it has high priority over the entire system. This in comparison with the tight polling which instead has some variation for the latency once it reaches the stable state. This is due to the fact that the poll only occurs once the background tasks have been run and any changes in the execution time of the background task function will affect the latency.

Interrupt vs Tight Polling



The major differences that can be observed between the two approaches are as follows. Firstly the interrupt completes more background tasks at any given period because does not have to spend any time polling the data register of stimulus in. Instead the interrupt will directly suspend execution of a background task and then resume as soon as it is finished. The time spent polling for the data of stimulus in results overall in significant portion of time which cuts into the time spent running background tasks.

The second major difference is the predictability of either approach in the number of background tasks at a given clock cycle. The interrupt conforms extremely nicely to a trend line which if translated into a function, could be used to easily predict the number of the background tasks at any given number of clock cycles. The tight polling on the other hand is nearly impossible to predict to any given accuracy since there are the large discontinuities and the negative slopes between them. Therefore if predictability of the approach is a priority the interrupt would be significantly better.