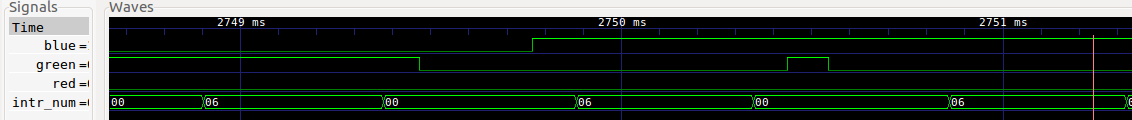
IN4343 – Lab 3

Casper van Wezel (4209192) & Erwin de Haan (4222814) – 2017-03-19

# Correctness check

1. The order of the tasks now is correct. The yellow (blue) signal now activates before the other pending green tasks.



# Efficiency improvement

2a. Hypothesis: HandleTasks is really only affected when a task is activated. Otherwise the if statement immediately makes the method return. You could also shorten the TimerIntrpt function by also starting with the highest registered task prio, this will stop NUMTASKS being relevant for performance. Although this seems to be outside the scope.

2b. As a consequence the execution time of CountDelay(60000) will not be impacted by much because the interrupt handler is not affected in execution time.

3a.

3b. The highest pending task needs to be set in the Interrupt handler. And the HandleTasks needs to break the while loop if the becomes higher than the current executing tasks’ priority. also needs to be initialized to . Also make signed.

3c. (git diff)

diff --git a/Lab3/SchedulerNP.c b/Lab3/SchedulerNP.c

index f3fe626..056c163 100644

--- a/Lab3/SchedulerNP.c

+++ b/Lab3/SchedulerNP.c

@@ -48,7 +48,7 @@

#include "Scheduler.h"

Task Tasks[NUMTASKS]; /\* Lower indices: lower priorities \*/

-uint8\_t Pending = 0; /\* Indicates if there is a pending task \*/

+int8\_t HPrioPendingTask = -1; /\* Indicates if there is a pending task \*/

uint16\_t IntDisable(void) {

uint16\_t sw;

@@ -114,9 +114,10 @@ uint8\_t UnRegisterTask(uint8\_t t) {

\*/

void HandleTasks(void) {

- while (Pending) {

- int8\_t i = NUMTASKS - 1; Pending = 0;

- while (i >= 0 && !Pending) {

+ while (HPrioPendingTask >= 0) {

+ int8\_t i = HPrioPendingTask;

+ HPrioPendingTask = -1;

+ while (i > HPrioPendingTask) {

Taskp t = &Tasks[i];

if (t->Activated != t->Invoked) {

if (t->Flags & TRIGGERED) {

@@ -124,6 +125,7 @@ void HandleTasks(void) {

} else t->Invoked = t->Activated;

} else i--;

}

+

}

}

@@ -135,10 +137,11 @@ interrupt(TIMERA0\_VECTOR) TimerIntrpt(void) {

if (t->Remaining-- == 0) {

t->Remaining = t->Period - 1;

t->Activated++;

- Pending = 1;

+ if(HPrioPendingTask < i)

+ HPrioPendingTask = i;

}

} while (i--);

- if (Pending) ExitLowPowerMode3();

+ if (HPrioPendingTask >= 0) ExitLowPowerMode3();

}

#endif

3d. The HandleTasks function should not be interrupted when the while (HPrioPendingTask >= 0) { int8\_t i = HPrioPendingTask; lines have passed but the HPrioPendingTask = -1; line has not been executed already. If the new task was a higher priority it will not be execute that round and will need until the next task activation to actually run.

Put

uint16\_t sw;

sw = IntDisable();

Before the two statements and

RestoreSW(sw);

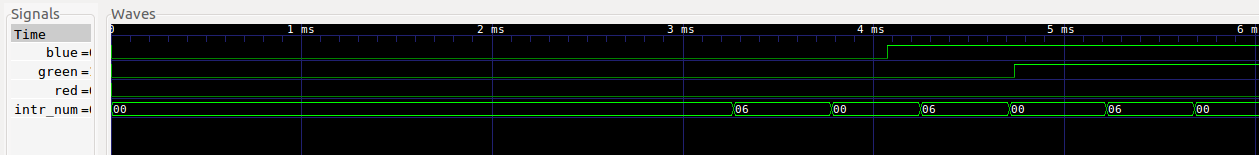
After.

4a. The measured execution time of the interrupt handler is 501 us. This changes about 1 us when there are two task activations, this is within the variances.

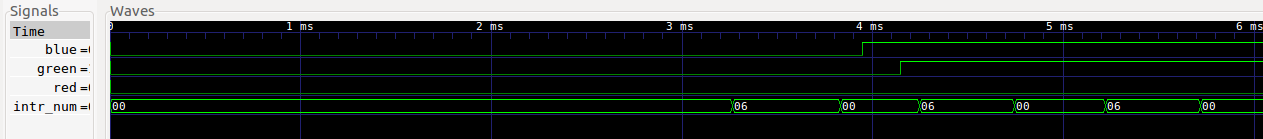
4b. The measured duration is 1.745-0.5=1.245 seconds. This is slightly longer. Propably due to the extra if statement in the interrupt handler.

5a. The startup behavior.

Old:

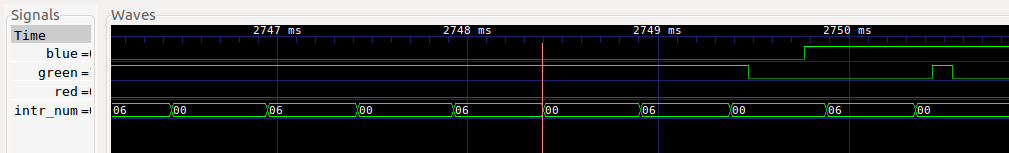


Improved:

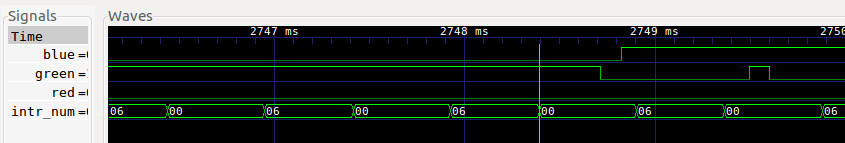


5b. Right after CountDelay

Old:



Improved:



6a. The activation delay is the following:  
Improved: Green 829 us. Yellow 582 us.  
Old: Green: 1492 us, Yellow 843 us  
Reduction: Green: 663 us; Yellow: 261 us

6b. The improvement in event latency of BlinkYellow is clearly visible after BlinkGreen happened, in the images of 5b. It is measured as about a 230-250 us reduction.

7. The Handling of tasks is more efficient, but as mentioned before the Interrupt Handler could use information from RegisterTask to reduce the loop iterations. This could have a much bigger impact due to the amount of times the Interrupt handler is called.