Real-Time Systems (2IMN20)
- Practical training 2015/2016 -

Leo Hatvani, Erik J. Luit and Reinder J. Bril
Technische Universiteit Eindhoven (TU/e),
Department of Mathematics and Computer Science,
Group System Architecture and Networking (SAN),
Den Dolech 2, 5612 AZ Eindhoven, The Netherlands
L.Hatvani@TUe.NL, E.J.Luit@TUe.NL, R.J.Bril@TUe.NL

Week 2: Basic experiments with `SchedulerNPBasic.c`

- The submission deadline for the exercises of week 3 is Thursday, February 25th, 2016 (23:59h), i.e. *after* the Carnaval.
- You will be graded on a scale from 0-10; with each of the questions you can earn 1 point.
- You are expected to motivate your answers briefly.

1 Preparation

Install the environment on your PC[1]. On windows, you can access the directory-structure of the virtual machine via for example `\\192.168.192.128`. The commands to (un-) mount the drive of the virtual machine are shown in each new terminal window in the virtual machine.

Documentation about the "*Virtual machine for MSP430 development*" can be found in OASE. A package containing a makefile `Makefile` will be available in OASE to complete the set of sources. The following actions will allow you to get started with your experiments:

- *Compile the program, generate a trace, and visualize the result*:
  Execute the following commands

  ```
  $ make
  $ wsim-iclbsn2 --ui --trace=sched.trc --mode=time --modearg=5s SchedTest.elf
  $ wtracer --in=sched.trc --out=wsim.vcd-NPBASIC --format=vcd
  $ gtkwave wsim.vcd-NPBASIC
  ```

  The last command opens a GTKWave window; see Figure 1[2].

- *Set-up the visualizer*:
  By left-clicking `led` in the `STT` window, `blue`, `green`, and `red` are shown in the `Signals` window *below* the `STT window`. Note that `YELLOW` in the code (`Led.h`) is represented by the `blue` signal. By dragging these three signals (one by one)

  - *from* the `Signals` window *below* the `STT window`
  - *to* the `Signals` window *to the right* of the `STT window`,

[1] `http://www.win.tue.nl/san/education/2IN26/`
[2] The image of this figure has been created using the "snippingtool" of Windows.

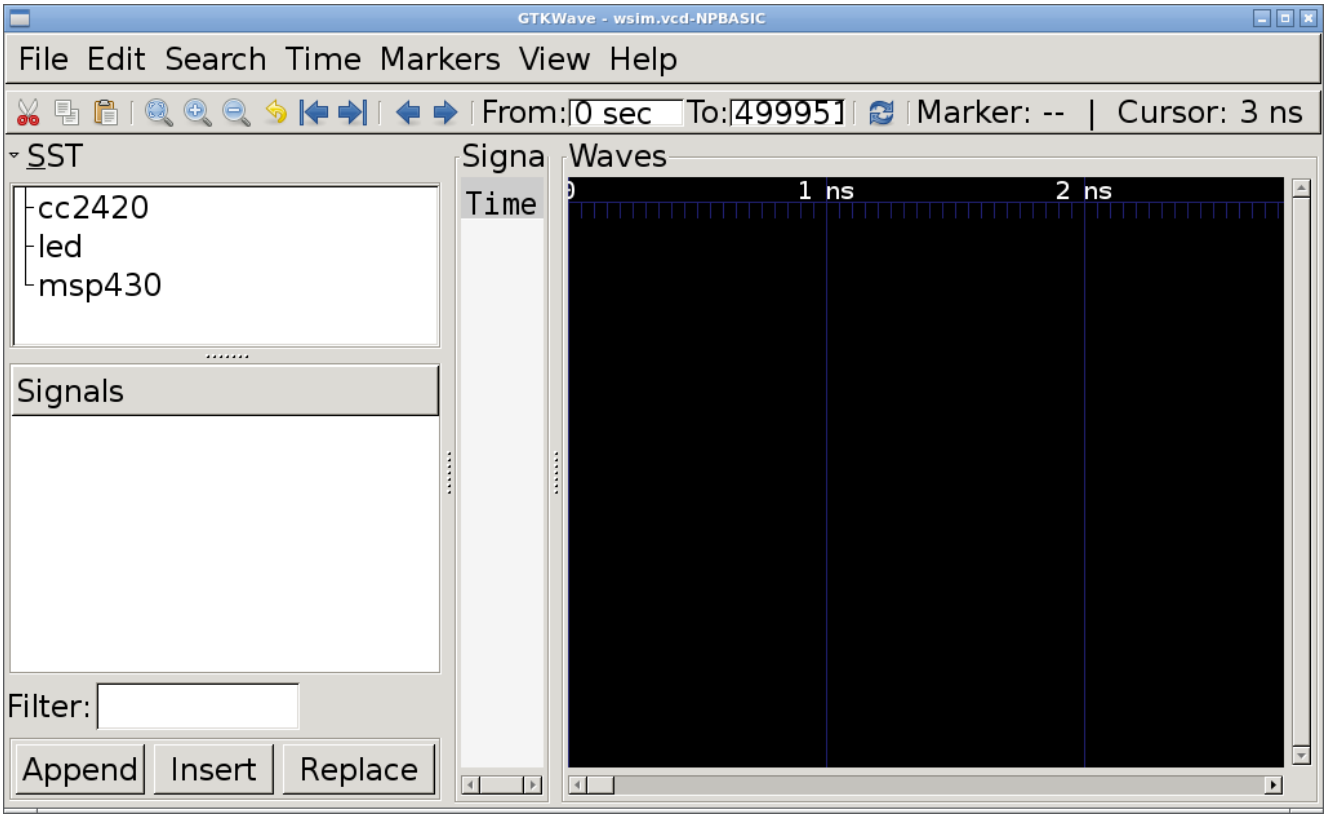

Figure 1. GTKWave window for `wsim.vcd-NPBASIC`.

additional lines become visible in the `Waves` window.

Left-click `msp430` in the `STT` window, and drag `intr_num[7:0]`

- *from* the `Signal` window *below* the `STT window`
- *to* the `Signal` window *to the right* of the `STT window`.

Right-click `intr_num[7:0]`, and subsequently select `Data Format`, `Analog`, and `Step`.

- *Basic use of the simulator*:
  Repeatedly click "zoom-out", i.e. the icon with a magnifying-glass and an "−", to get a timeline covering at least 4 *ms*.

  By left-clicking on the rising edge of `blue` in the `Waves` window, a red line appears and the `Marker` provides detailed timing information.

- *Advanced use of the simulator*:
  Document `Virtual machine for MSP430 development-2016.pdf` contains various hints for using the simulator. The usage of markers (see Section "Trace analysis" in that document) may considerably reduce workload.

Finally, more information regarding interrupt handling can be found in the user guide of the MSP430 hardware, see Section 2.2.3 in `http://www.win.tue.nl/san/education/2IN26/MSP430\%20-\%20general.pdf` (or `MSP430-general.pdf` in OASE).

2 Hypotheses of week 1

In the following questions, we will take a close look at the actual behavior of the system. Moreover, we will validate and reflect on your hypotheses of week 1.

2.1 Questions about detailed start-up behavior

We first look at *the order* in which leds are changed.

Specific questions

1. Visualize the behavior of the system during 7 *ms* and include the resulting figure in your report. What can be observed from this timeline?
   Hint: consider the rising and falling edges of `blue` and `green` and the intervals during which `intr_num[7:0]` is high.
2. Answer the following questions and reflect on your answers to the corresponding questions 5 and 6 of week 1.
   (a) Which function has a higher priority: `BlinkYellow` or `BlinkGreen`?
   (b) The period of `BinkRed` is 0. What happens with the `BlinkRed` function?

2.2 Questions about Coarse-grain behavior

During week 1, we made various assumptions about the behavior of the system. We now revisit those assumptions.

Specific questions

3. We assumed that the time to start-up the system can be neglected.
   (a) How long does it take before the real-time clock actually starts?
   (b) What needs to be accomplished during the initialization?
   Hint: Are there requirements imposed by the C-standard?
4. We assumed that, apart from `CountDelay(60000)` the duration of all functions could be neglected. Using the simulator, we immediately see that the execution time of the timer interrupt handler is fluctuating due to the activations (and executions of the functions) of tasks. Measure the additional time needed to handle the two tasks activations in the first cycle.
   Hint: Measure the length of the first and second interval in which `intr_num[7:0]` is "high" (or "ON"). For ease of presentation, let
   - $\Delta_{\mathrm{IH},n}^{\mathrm{ON,NPBASIC}}$ denote the length of the n^{th} interval,
   - $r_{\mathrm{IH},n}^{\mathrm{NPBASIC}}$ denote the time of the n^{th} *rising* edge of `intr_num[7:0]`, and
   - $f_{\mathrm{IH},n}^{\mathrm{NPBASIC}}$ denote the time of the n^{th} *falling* edge.

   We now get
   $$\Delta_{\mathrm{IH},n}^{\mathrm{ON,NPBASIC}} = f_{\mathrm{IH},n}^{\mathrm{NPBASIC}} - r_{\mathrm{IH},n}^{\mathrm{NPBASIC}}.$$
5. Typically, *scheduling overhead* is ignored during schedulability analysis. For our resource-constrained system, it turns out that the interrupt handler implementing the scheduler takes a significant fraction of the time. What is the utilization spend by timer-interrupt handling?
6. The frequency of the clock interrupts is approximately 1024 Hz, yielding a clock period $T_{\mathrm{Clk}} \approx 1/1024\ s = 0.9765625\ ms$.
   (a) Measure the inaccuracy (i.e. the *jitter*) of the clock period (T_{Clk}) from the first 7 cycles.
   (b) What causes this inaccuracy?

We now revisit the ON/OFF outputs over a longer interval of time. To that end, visualize the behavior of the system for 3.5 s.

Specific questions

7. Measure the execution time $C_{\texttt{CountDelay(60000)}}$ of `CountDelay(60000)`.
   Hint: Use full precision in your calculations and give end results in *ms*. Include your calculations in your report. Try to calculate the most accurate result, i.e. choose intervals carefully. For ease of presentation, let

   - $\Delta^{\text{ON}}_{\texttt{green},n}$ denote the length of the n^{th} interval during which `green` is high;
   - $r_{\texttt{green},n}$ denote the time of the n^{th} rising edge of `green`; and
   - $f_{\texttt{green},n}$ denote the time of the n^{th} falling edge of `green`.

   We now get
   $$\Delta^{\text{ON}}_{\texttt{green},n} = f_{\texttt{green},n} - r_{\texttt{green},n}.$$

2.3 Additional question about detailed behavior

We now take a closer look at the interrupt level after the second falling edge of `green`. To that end, visualize the behavior of the system in sufficient detail.

Specific questions

8. Consider the arrival of the interrupts and their handling.
   (a) What is the first time after the falling edge of `green` that the execution of the timer-interrupt handler is "in-sync" again, i.e. at which time is the interrupt handler immediately executed (rather than delayed by the handling of previous interrupts)?
   (b) How much time is the execution of the timer handler that starts at roughly $2,143.9$ *ms* delayed?

We want to determine how many timer interrupts are lost due to the execution of `CountDelay(60000)`. According to the MSP430 user guide, interrupts may be delayed, there may be at most 1 interrupt pending, and once an interrupt is pending, subsequent interrupts will be lost/missed. For ease of presentation, we make the following assumptions:

- C_{IH} represents the execution time of the interrupt handler;
- T_{Clk} represents the period of the interrupt;
- preemptions by higher priority interrupts are ignored.

Specific questions

9. Formulate a sufficient condition (in terms of C_{IH} and T_{Clk}) for the system where n interrupts are lost and 1 interrupt is pending.
10. Assume the frequency of the clock is "fixed", e.g. $T_{\text{Clk}} = 1/1024s \approx 0.977$ *ms*. How many interrupts are missed due to `CountDelay(60000)`?