# Exercise 2

Rick Veens Studentno: 0912292 r.veens@student.tue.nl
Huib Donkers Studentno: 0769015 h.t.donkers@student.tue.nl

July 3, 2015

## 1 Timers

### 1.1 Model

The model is shown in figure 1, and code used for the first two tests in codeblock 1. The improved code used for the third test is shown in codeblock 2. We used `octave` to parse the output as csv and perform some statistical analysis.

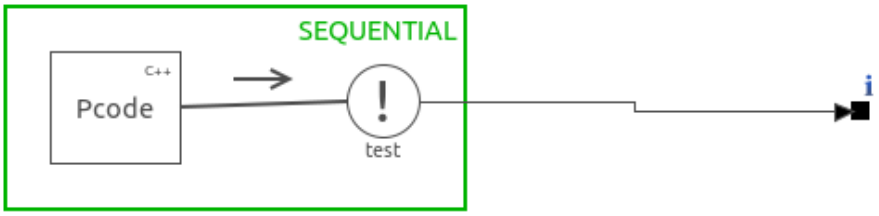
Codeblock 1: Pcode::execute

```
1  void Pcode::execute()
2  {
3    // protected region execute code on begin
4      static struct timespec t1;
5
6
7      struct timespec t2;
8      struct timespec res;
9
10     clock_gettime(CLOCK_REALTIME, &t2);
11     clock_getres(CLOCK_REALTIME, &res);
12     //long int elapsedTime = (t2.tv_usec - t1.tv_usec);
13
14     printf("%ld.%.9ld, %ld.%.9ld, %ld.%.9ld\n", res.tv_sec, res.tv_nsec, t2.tv_sec, ↩
           t2.tv_nsec, t1.tv_sec, t1.tv_nsec);
15
16     t1 = t2;
17
18     // protected region execute code end
19 }
```

Codeblock 2: Pcode::execute (improved)

```
1  todo
```


(a) Main model.




(b) submodel

Figure 1: Model used for testing the timer

## 1.2 Questions

1. Measured over 1522 intervals, the variation is $3.667 \cdot 10^{-8}$ s$^2$ or 0.03667 ms$^2$. Since the clock resolution of QNX in virtual box (0.000999848 s) is so much larger than the observed variation, our measurements are not accurate enough to result in a variation that is representative for the actual variation. With a clock resolution of 0.000999848 s, and a timer that ticks every 0.250000000 s without jitter, we expect $\lceil 0.25/0.000999848 \rceil - 0.25/0.000999848 = 96.2\%$ of the measurements to be $\lfloor 0.25/0.000999848 \rfloor \cdot 0.000999848 = 0.249962$ s, and the other 3.8% to be $\lceil 0.25/0.000999848 \rceil \cdot 0.000999848 = 0.250962$ s. In 1522 measurements we found 96.2% of the intervals to be 0.249962 s and 3.8% 0.250962 s. Therefore, our measurements do not provide evidence of jitter.

   Up to a jitter bounded by $0.250000 - 0.249962 = 0.00038$ s $= 380$ $\mu$s, we would still expect the same results. For larger jitter, we would expect more and more measurements to be 0.248962 s. So we can strengthen our claim: our measurements do not provide evidence of jitter larger than 380 $\mu$s.

   We performed a second series of measurements, this time with the timer interval set to 0.249962, an exact multiple of the clock resolution. We measured 1967 intervals. With this configuration, we are likely to pick up on jitter with a variation larger than $(0.000999848/1967 \cdot 10^6)^2 \approx 0.258$ $\mu$s$^2$.

   Results of the second test is shown in figure 2. This shows that there is indeed jitter. The variation of our measurements is 0.06 ms$^2$.

   For a third test we used a method of timing with a resolution that is much higher: `ClockCycles()` from `sys/neutrino.h`. We can retrieve the number of clock cycles per second, so we can deduce how much time has passed if we know how many clock cycles passed. We measured 3339 intervals using this method. The distribution of intervals

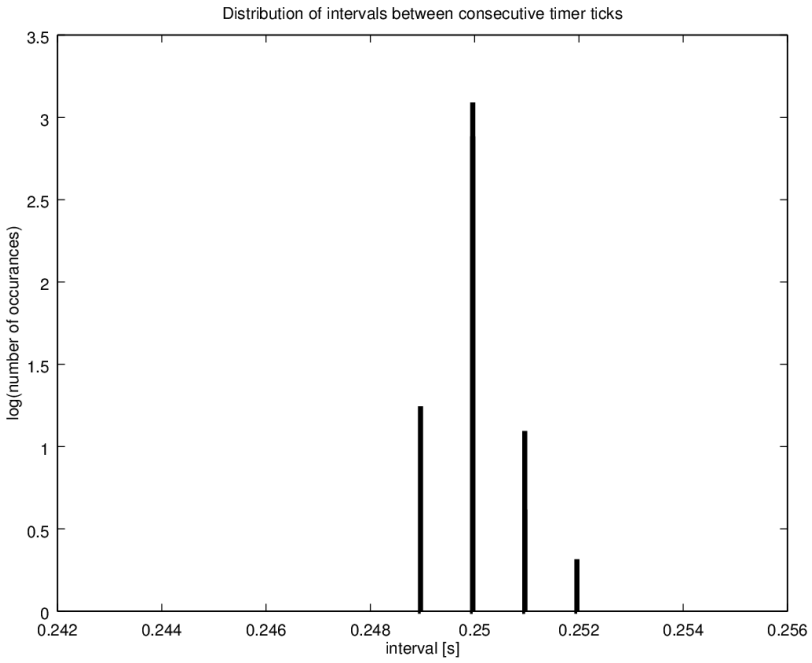



Figure 2: Distribution of intervals in the second test.

is shown in figure 3. We can clearly see hoe the jitter behaves. It is still distributed somewhat discretely, but smaller deviations are now clearly shown. In this test the variation of the measurements is 7.814 $ms^2$.

2. This jitter is the result of simulating a real time OS in an environment that is not real time. Since the host OS decided when the virtual OS can run, and the host OS cannot guarantee to meet real time requirements, the virtual OS is unable to meet those requirements either.

3. We observe from figure 2 that the timer tick is occasionally delayed by more than 1 ms. Such a delay is not acceptable in many real time applications. EXAMPLE.

4. No. Our virtual machines run on a non real time OS (GNU/Linux), so it can happen that the host OS is very busy with tasks that have a higher priority than the virtual machines, postponing execution of the real time OS unboundedly, disabling QNX to meet its real time requirements.

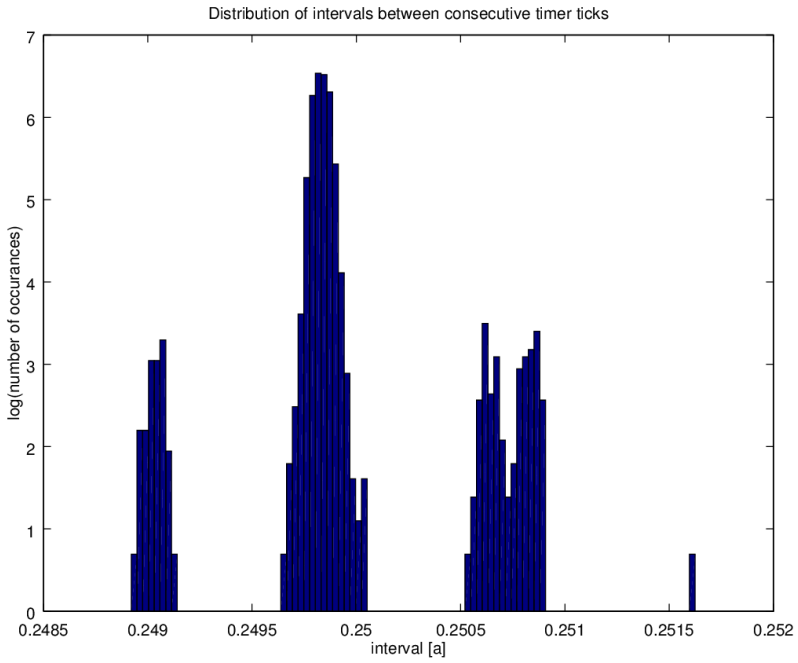



Figure 3: Distribution of intervals in the third test.

### 1.3 Validation

We used the clock cycle approach, as described in section 1.2 answer 1, to determine the exact interval.

### 1.4 Questions

1. TODO
2. TODO
3. TODO

## 2 JIWYIO Linkdrivers

### 2.1 JIWYIO driver components

UITLEG

### 2.2 Questions

1. TODO
2. WHAT!? ... YOU WANT US TO TELL YOU HOW THE TOOL SHOULD WORK? THIS ISN'T SOFTWARE SPECIFICATION, AND EVEN WORSE, THE TOOL IS NOT THE SUBJECT, IT'S THE TOOL...

3. The most obvious drawback is of course that you need to do the exact same edits every time you generate code from the models. Another drawback is that you have to adapt your code to run simulations for testing. It is not possible to test the actual code that will be run in the real environment.

### 2.3 JIWYIO driver components in the lab

NOTES UITWERKEN
run 1: 4Hz, $\text{timer}_test1.csv - oscilloscope$ : $500.0ms(sometimes501.0or499.0)$
run 2: 100Hz, $\text{timer}_test2.csv - oscilloscope$ : $20.0ms$

### 2.4 Questions

1. TODO
2. TODO

## 3 Controlling JIWY with QNX & CSP

### 3.1 Test at the lab

REPORT

### 3.2 Questions

1. TODO
2. TODO

### 3.3 Further functionality

EXPLAIN WE DIDN'T HAVE TIME AND IT'S NOT OUR FAULT

### 3.4 Questions

1. TODO
2. TODO
3. TODO
4. TODO