

Header: Grade: FOR /20 DES /40 EXP /30 ORI /10 / TOT _100
Lab # 4

CS 5390 Summer 2017, Date of Submission: 08/04/17

Students Names: Adeel Malik (amalik@utep.edu)

Instructor: Yadira Jacquez

Section 1: Effort: 6 hours

- Planning and preparation: 1 hour
- Experiment: 4 hours (on simulator)
- Report writing: 1 hour

Section 2: Objectives

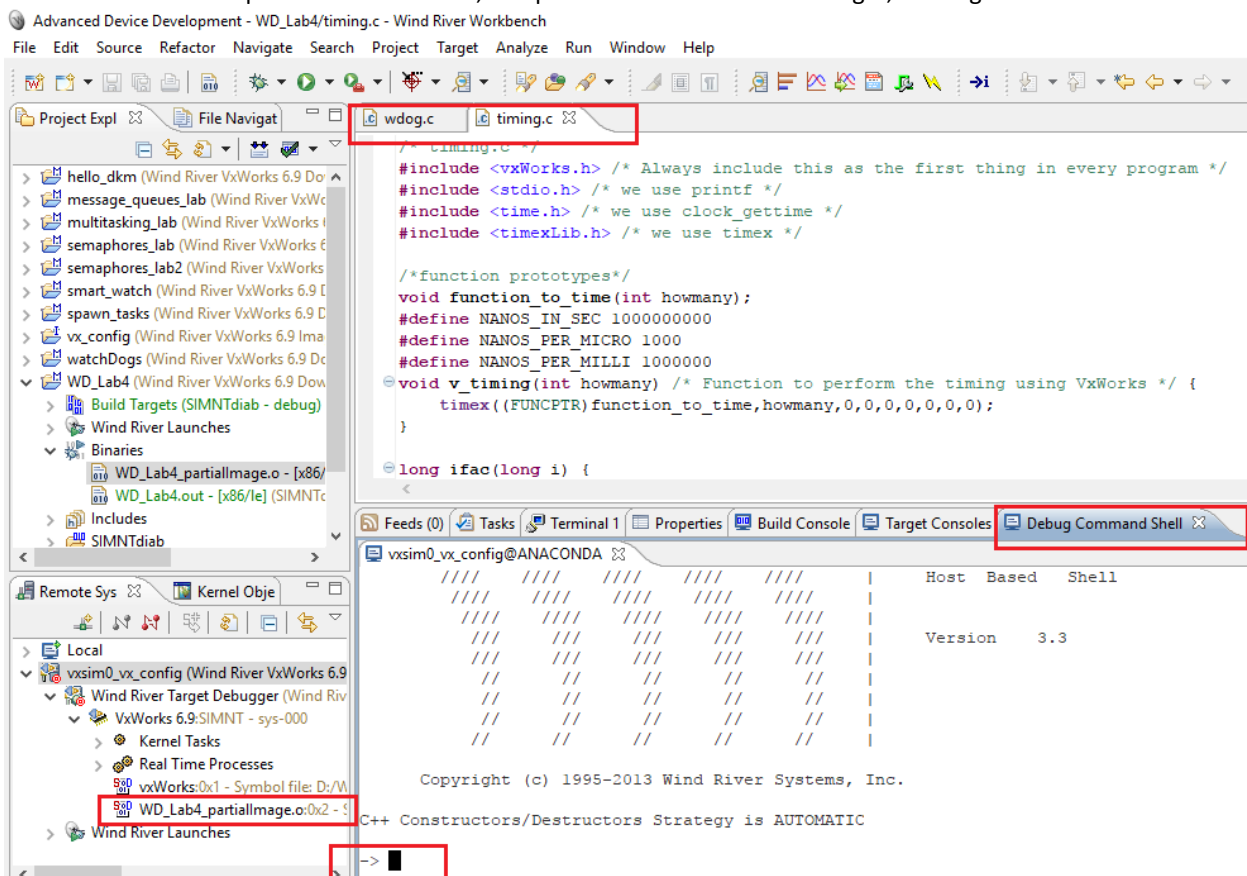
The objective of this experiment is to learn the concept of watchdog timer and use POSIX 1003.1b and VxWorks OS routines to delay task execution.

Section 3: Procedures and Results

Part A:

A1. Add the example source code to the project, compile, and download to the target. Start the shell.

Answer: Attached the provided source code, compiled and downloaded the target, running shell.



- A2.** Execute function `wd_test(wdID)` from command line. Observe the output and the currently running tasks.
- (i). Where is the output displayed? What tasks are running? Explain. Create required watchdog timer `wdID = wdCreate()` and re-execute. Check the status of watchdog using "show wdID". Show the results and comment on them.

Answer: Running the command `wd_test(wdID)` returns an error because the watch dog `wdID` is not yet created. Output is displayed in the shell. After creating watch dog with the command `wdID=wdCreate()` and re-executing the command `wd_test(wdID)` keeps the state of watch dog "IN_Q" as long as there are remaining ticks, once the ticks are over, the state changes to "OUT_OF_Q" and ticks are random every time. Results are displayed in the shell.

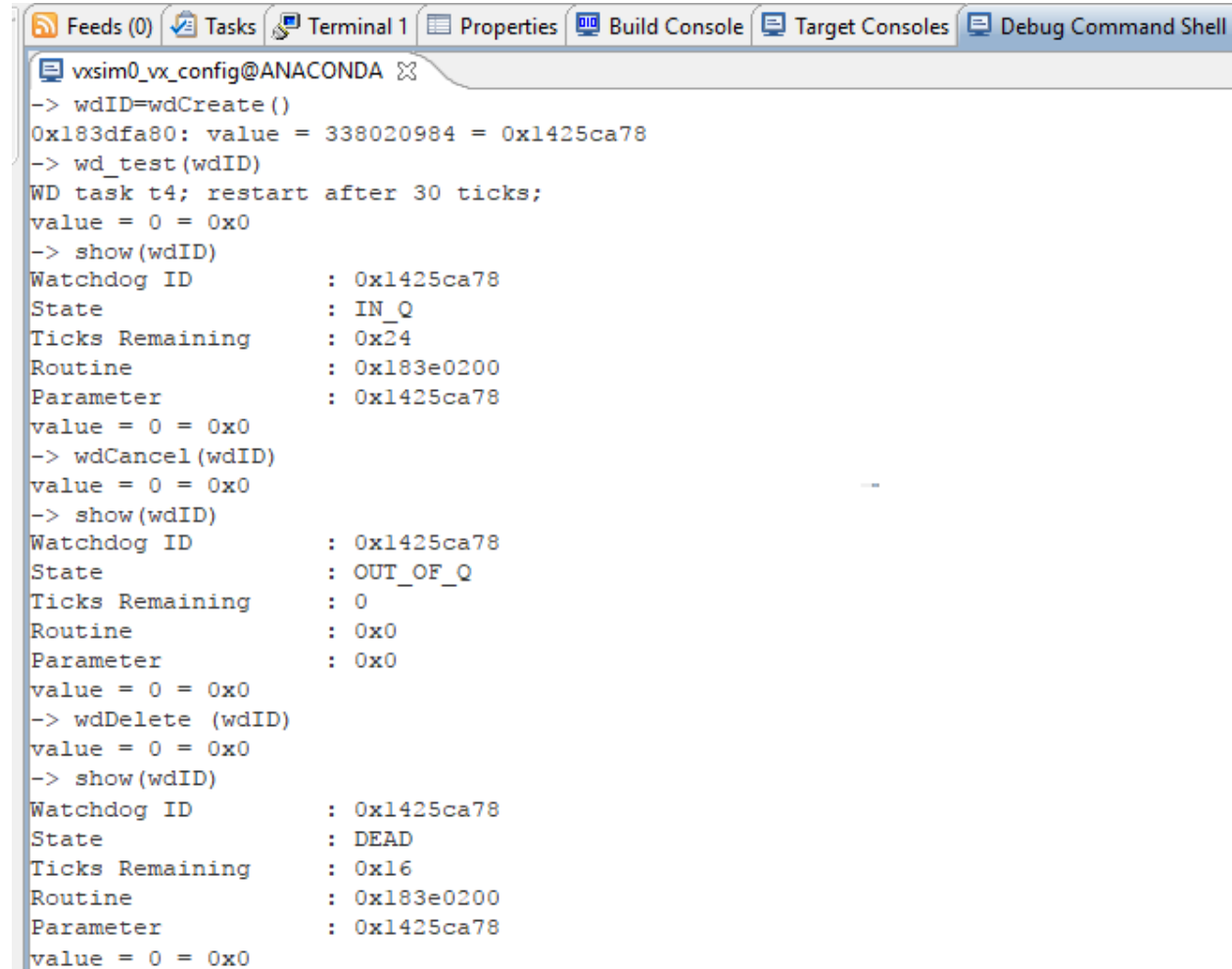
```

Feeds (0) Tasks Terminal 1 Properties Build Console Target Consoles Debug Command Shell
vxsim0_vx_config@ANACONDA
-> wd_test(wdID)
WD task t1; restart after 30 ticks;
value = -1 = 0xffffffff
-> wdID=wdCreate()
0x183dfa80: value = 339214208 = 0x1437ff80
-> wd_test(wdID)
WD task t3; restart after 10 ticks;
value = 0 = 0x0
-> i
value = 0 = 0x0 ITRY      TID    PRI    STATUS    PC      ERRNO    DELAY
-----
tJobTask    jobTask    0x14217a00  0 Pend    0x1015dbab  0x0      0
tExcTask    excTask    0x101e73e0  0 Pend    0x1015dbab  0x0      0
tLogTask    logTask    0x1421b740  0 Pend    0x1015b8eb  0x0      0
tShell0     shellTask  0x14377e40  1 Pend    0x1015dbab  0x0      0
tWdbTask    wdbTask    0x184a8fd0  3 Ready   0x1015dbab  0x0      0
ipcom_tick  ipcom_tickd 0x1437d970  20 Pend    0x1015dbab  0x0      0
tVxdbgTask  vxdbgEventTa 0x184a86d0  25 Pend    0x1015dbab  0x0      0
tAioIoTask  aioIoTask  0x1421fbe0  50 Pend    0x1015e446  0x0      0
tAioIoTask  aioIoTask  0x14238750  50 Pend    0x1015e446  0x0      0
tNet0       ipcomNetTask 0x1423cdf0  50 Pend    0x1015dbab  0x3d0001  0
ipcom_sysl  ipcom_syslog 0x1423f4a0  50 Pend    0x1015e446  0x0      0
tNetConf    ipnet_config 0x1429f900  50 Pend    0x1015dbab  0x0      0
tAioWait    aioWaitTask 0x1421f728  51 Pend    0x1015dbab  0x0      0
value = 0 = 0x0
-> show(wdID)
Watchdog ID      : 0x1437ff80
State            : IN_Q
Ticks Remaining  : 0xE
Routine          : 0x183e0200
Parameter        : 0x1437ff80
value = 0 = 0x0
Watchdog ID      : 0x1437ff80
State            : IN_Q
Ticks Remaining  : 0x9
Routine          : 0x183e0200
Parameter        : 0x1437ff80
value = 0 = 0x0
-> show(wdID)
Watchdog ID      : 0x1437ff80
State            : OUT_OF_Q
Ticks Remaining  : 0
Routine          : 0x0
Parameter        : 0x0
value = 0 = 0x0

```

A3. Experiment with wdCancel, wdStart, and wdDelete from the shell line. Explain and show your results.

Answer: wdStart starts the watchdog timer. The image below shows that a watchdog timer is created to execute after 30 ticks. wdCancel cancels the started watchdog timer and changes the state to OUT_OF_Q. wdDelete deallocates a watchdog timer and cancels any previous start.



```
vxsim0_vx_config@ANACONDA
-> wdID=wdCreate()
0x183dfa80: value = 338020984 = 0x1425ca78
-> wd_test(wdID)
WD task t4; restart after 30 ticks;
value = 0 = 0x0
-> show(wdID)
Watchdog ID      : 0x1425ca78
State            : IN_Q
Ticks Remaining  : 0x24
Routine          : 0x183e0200
Parameter        : 0x1425ca78
value = 0 = 0x0
-> wdCancel(wdID)
value = 0 = 0x0
-> show(wdID)
Watchdog ID      : 0x1425ca78
State            : OUT_OF_Q
Ticks Remaining  : 0
Routine          : 0x0
Parameter        : 0x0
value = 0 = 0x0
-> wdDelete (wdID)
value = 0 = 0x0
-> show(wdID)
Watchdog ID      : 0x1425ca78
State            : DEAD
Ticks Remaining  : 0x16
Routine          : 0x183e0200
Parameter        : 0x1425ca78
value = 0 = 0x0
```

A4. What VxWorks system function you must use to display clock resolution? Use it, show result and explain.

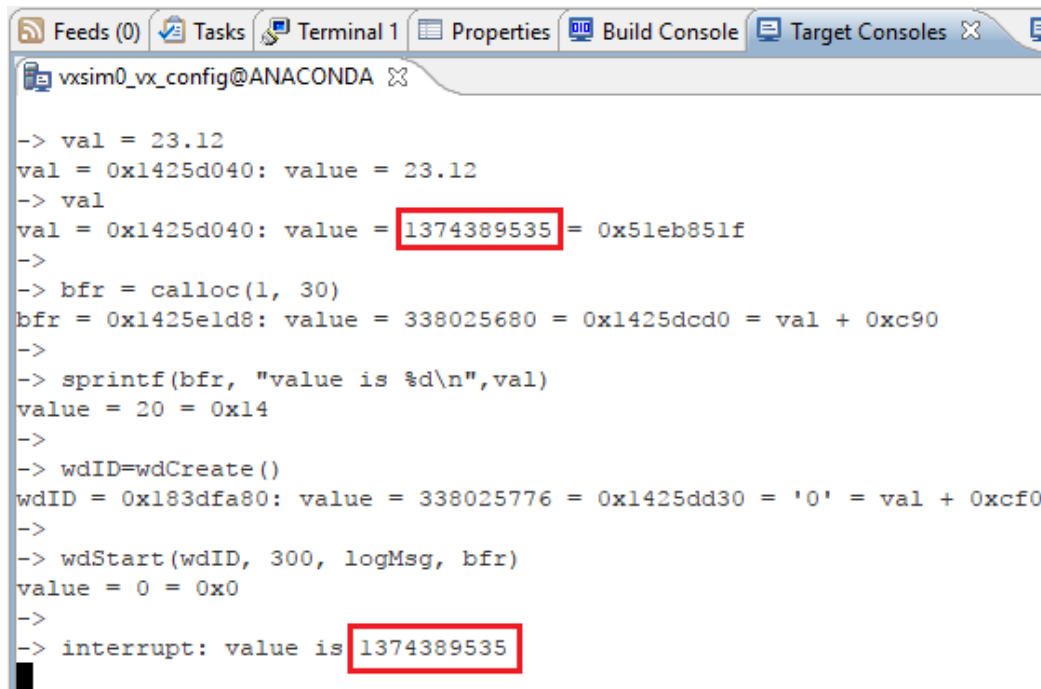
Answer: Clock resolution can be displayed by using the command sysClkRateGet(), default clock rate is 60. This can be changed by the command sysClkRateSet(). However, this should only be done at system startup. Along with these functions, we can also get and set the tick values to get the actual time taken by different tasks.

```
-> sysClkRateGet()
value = 70 = 0x46 = 'F' = _vx_offset_TRIGGER_status
-> sysClkRateSet(70)
value = 0 = 0x0
-> sysClkRateGet()
value = 70 = 0x46 = 'F' = _vx_offset_TRIGGER_status
-> tickGet()
value = 30182 = 0x75e6
-> tickSet(100)
value = 0 = 0x0
-> tickGet()
value = 301 = 0x12d = _vx_offset_PARTITION_curWordsAllocatedInternal + 0x19
```

- A5.** Describe the experiments showing the VxWorks shell command lines required to:
- (a) add `val = 23.12` to the symbol table,
 - (b) create buffer of 30 characters and fill it with string "value is XX.XX" (where XX.XX is the numerical value of variable `val`),
 - (c) start the watchdog to print the string with 5 seconds delay. Show your commands and results.

Answer: This can be achieved by the following set of commands...

```
val = 23.12 //setting the value
bfr = calloc(1, 30) //creating a buffer of 30
sprintf(bfr, "value is %d\n", val) //copying the exact value of val into the buffer
wdID=wdCreate() //creating a watch dog
wdStart(wdID, 300, logMsg, bfr) //starting the watch dog to print the buffer after 5 seconds
```



```
vxsim0_vx_config@ANACONDA
-> val = 23.12
val = 0x1425d040: value = 23.12
-> val
val = 0x1425d040: value = 1374389535 = 0x51eb851f
->
-> bfr = calloc(1, 30)
bfr = 0x1425e1d8: value = 338025680 = 0x1425dcd0 = val + 0xc90
->
-> sprintf(bfr, "value is %d\n", val)
value = 20 = 0x14
->
-> wdID=wdCreate()
wdID = 0x183dfa80: value = 338025776 = 0x1425dd30 = '0' = val + 0xcf0
->
-> wdStart(wdID, 300, logMsg, bfr)
value = 0 = 0x0
->
-> interrupt: value is 1374389535
```

Part B:

- B1.** Add program `timing.c` to the project. Explain the functionality of functions: `ifac()`, `function_to_time()`, `p_timing()`, and `v_timing()`.

Answer: **Function ifac** runs with the long parameter "i" and it is triggered from the function `function_to_time`. Based on the value of i, the function returns a value. If "i=0 or i=1" it returns value 1, else computes the value based on the formula "i*ifac(i-1)" and returns the value.

Function function_to_time has the integer parameter "howmany", there are 3 integer variables i, j and k for 3 loops. First loop does nothing except wasting time, executing till the value is less than "howmany". 2nd loop calculates the value of sum, runs for 499 times. 3rd loop is inside the 2nd loop and runs 9 times calling `ifac` function with different values of j(1 to 9). And at the end, the function prints the value of sum.

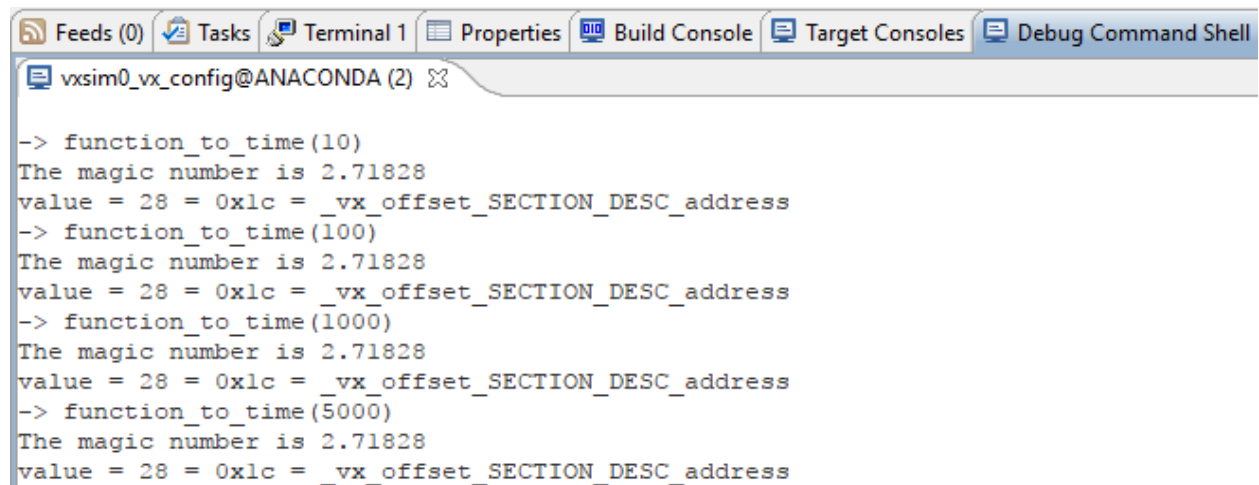
Function p_timing() calculates the time taken by the function `function_to_time` and prints the execution time in Microseconds and Milliseconds.

Function v_timing() calculates the execution time using VxWorks timing utility "timex"

- B2:** Execute `function_to_time()` in WindSh with different arguments. For example, type `function_to_time(10)` to specify a value of 10 for the integer argument `howmany`. What shell command you must use? What are the results?

Answer: Executed the function with the arguments (10, 100, 1000, 5000), the results are shown on the host shell.

```
function_to_time(10)
function_to_time(100)
function_to_time(1000)
function_to_time(5000)
```



```
vxsim0_vx_config@ANACONDA (2)
-> function_to_time(10)
The magic number is 2.71828
value = 28 = 0x1c = _vx_offset_SECTION_DESC_address
-> function_to_time(100)
The magic number is 2.71828
value = 28 = 0x1c = _vx_offset_SECTION_DESC_address
-> function_to_time(1000)
The magic number is 2.71828
value = 28 = 0x1c = _vx_offset_SECTION_DESC_address
-> function_to_time(5000)
The magic number is 2.71828
value = 28 = 0x1c = _vx_offset_SECTION_DESC_address
```

- B3:** Execute `v_timing()` and `p_timing()` functions with an argument of 200 five times. DO NOT INCLUDE RAW OUTPUT IN THE REPORT. Are the results the same? Where do you get the information on the timing? What are the units? What is the average?

Answer: The results are not same, execution time returned by both functions is almost the same. The results are shown on the target console. Units are in microseconds and the average is 28571.

Executing the `p_timing` function with argument 200, returns smooth results using the POSIX clock. `v_timing` function on the other hand, which uses VxWorks timing utility `timex`, is sometimes not able to calculate the time returning the message "...execution time too short to be measured meaningfully..."

- B4:** Repeat experiment B2 for different arguments (use e.g. 100 and 300) recording the execution time. Use both `p_timing()` and `v_timing()`. Build a table repeating each experiment ten times and computing the average and standard deviation for each of the timing method.

Answer: Running the experiment with argument 100 for both `p_timing` and `v_timing`. Function `p_timing` runs fine as usual whereas `v_timing` is not able to run with argument 100 returning message "execution time too short to be measured meaningfully in a single execution". The execution is so fast relative to the clock rate that the time is meaningless in case of VxWorks `timex` utility.

argument (100)	Time Taken(p_timing)	Time Taken(v_timing)
Experiment 1	14285.714	NA
Experiment 2	14285.714	NA
Experiment 3	14285.714	NA
Experiment 4	14285.714	NA
Experiment 5	14285.714	NA

Experiment 6	28571.428	NA
Experiment 7	14285.714	NA
Experiment 8	14285.714	NA
Experiment 9	14285.714	NA
Experiment 10	14285.714	NA

Average	15714.2854	#DIV/0!
SD	4517.539424	#DIV/0!

When argument 300 was used, the results are as below...

argument (300)	Time Taken(p_timing)	Time Taken(v_timing)
Experiment 1	42857.142	28571
Experiment 2	28571.428	28571
Experiment 3	42857.142	28571
Experiment 4	42857.142	42857
Experiment 5	28571.428	42857
Experiment 6	42857.142	42857
Experiment 7	42857.142	42857
Experiment 8	28571.428	42857
Experiment 9	28571.428	42857
Experiment 10	42857.142	28571

Average	37142.8564	37142.6
SD	7377.110988	7377.258678

Average time taken and standard deviation for both functions is almost the same in this case.

B5: What are the clock resolutions observed using the POSIX and VxWorks routines? Confirm your answers based on the above experiments.

Answer: I have observed that POSIX clock returns the lowest possible execution time and VxWorks timing utility timex has a limitation to calculate time.

B6: Repeat experiment B2 with smaller arguments (try 20, 10, 5, 2) for both p_timing() and v_timing(). What happens? What do you need to modify in the v_timing() function to get the timing in the case when the message says: "...execution time too short to be measured meaningfully..."?

Answer: p_timing with argument 20 sometimes return the time, lower argument values are returning time 0 most of the times. v_timing returns the message saying "...the execution time is short.."

To overcome the issue with v_timing, we need to use timexN() routine instead of timex().

Section 4: Observations, Comments, and Lessons Learned

I have learned about watchdog timers, POSIX and VxWorks timex utility. I also have learned based on the experiments that VxWorks timex has some limitations whereas POSIX clock works fine for many cases.