**COSC 6384: Real-Time Systems**

**Name: Srinivasa Karthic Ponguru**

**INTRODUCTION:**

* Here, we use VxWorks and its functions to run a set of three tasks whose priorities change dynamically during execution. A character array is taken as an input in a separate header file. The letter present in each word are sorted alphabetically using 3 different sorting techniques:

1. Insert Sort
2. Select Sort
3. Heap Sort

* The sorted output is then printed.

**PROGRAM EXPLANATION:**

* In this program we use the following header files for using VxWorks functions
* VxWorks.h
* tickLib.h
* taskLib.h
* taskHookLib.h

taskHookInit( ) - initialize task hook facilities. This is called automatically if the configuration macro INCLUDE\_TASK\_HOOKS is defined.

* Here, we have a source string (sourceString) that is included in the header file “text.h”. This source string contains special characters and spaces which have to be removed.
* Now, the special characters in the character array are eliminated by the “removespecialchars()” function and the special character are replaced by spaces. This is done by using the ASCII values of the characters.
* Then the spaces are removed between the words to make the character array compatible for sorting. This is done by using the “removes()” function by the use of pointers.
* The tickSet() is set to 0 initially to initiate the timer.
* We also calculate the clock cycle time of the system by using the function “syClkRateGet()” and taking its inverse.
* Task Hooks are created and added to the three different events

taskCreateHookAdd((FUNCPTR)tstart): invokes the “tstart()” routine. This saves the start time of each task using “tickGet()” function.

taskSwitchHookAdd((FUNCPTR)tswitch): invokes the “tswitch()” routine. This trackes the difference between the tracking variable “note\_ticks” and current time “ctime” using “tickGet()” function in VxWorks.

taskDeleteHookAdd((FUNCPTR)tdone) : invokes the “tdone()” routine. This tracks the end time of each task using the same “tickGet()” function.

* Then the task set of the three tasks are spawned using the “taskSpawn()” function in VxWorks. “taskSpawn()” will create and activates the task with a given priority. These methods will actually call the sort methods defined and then print the result of insert sort, selction sort and heap sort (each letter of the words in the character array are sorted alphabetically).
* The three sort functions define are:

void insert\_sort(): for doing insert sort.

void select\_sort(): for doing selction sort.

void heap\_sort(): for doing heap sort.

* We have declared three different priorities initially as: (These are the initial priorities when each task is spawned)

PRIORITYL 255: for low priority.

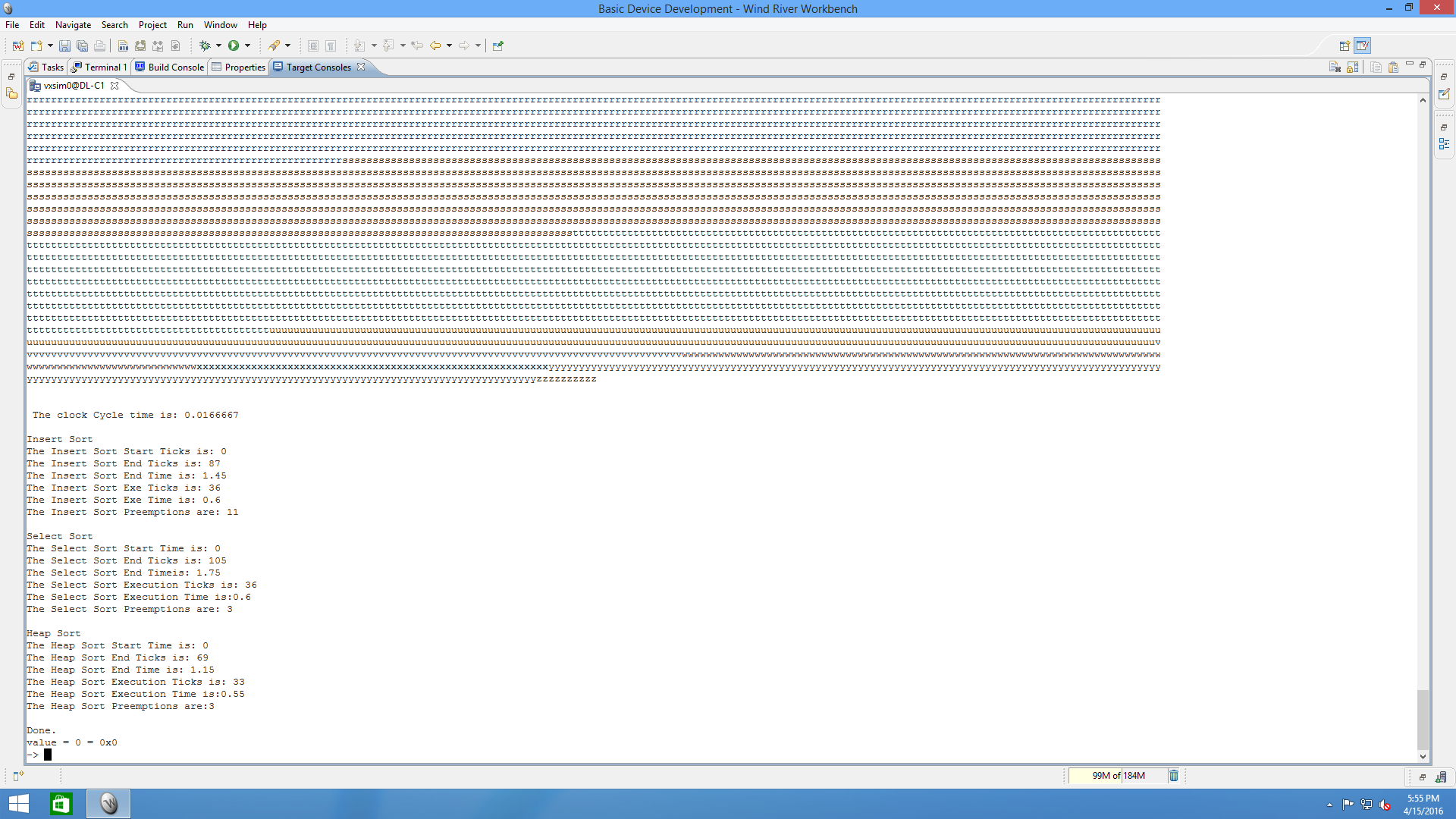
PRIORITYM 128: for medium priority.

PRIORITYH 0: for high priority.

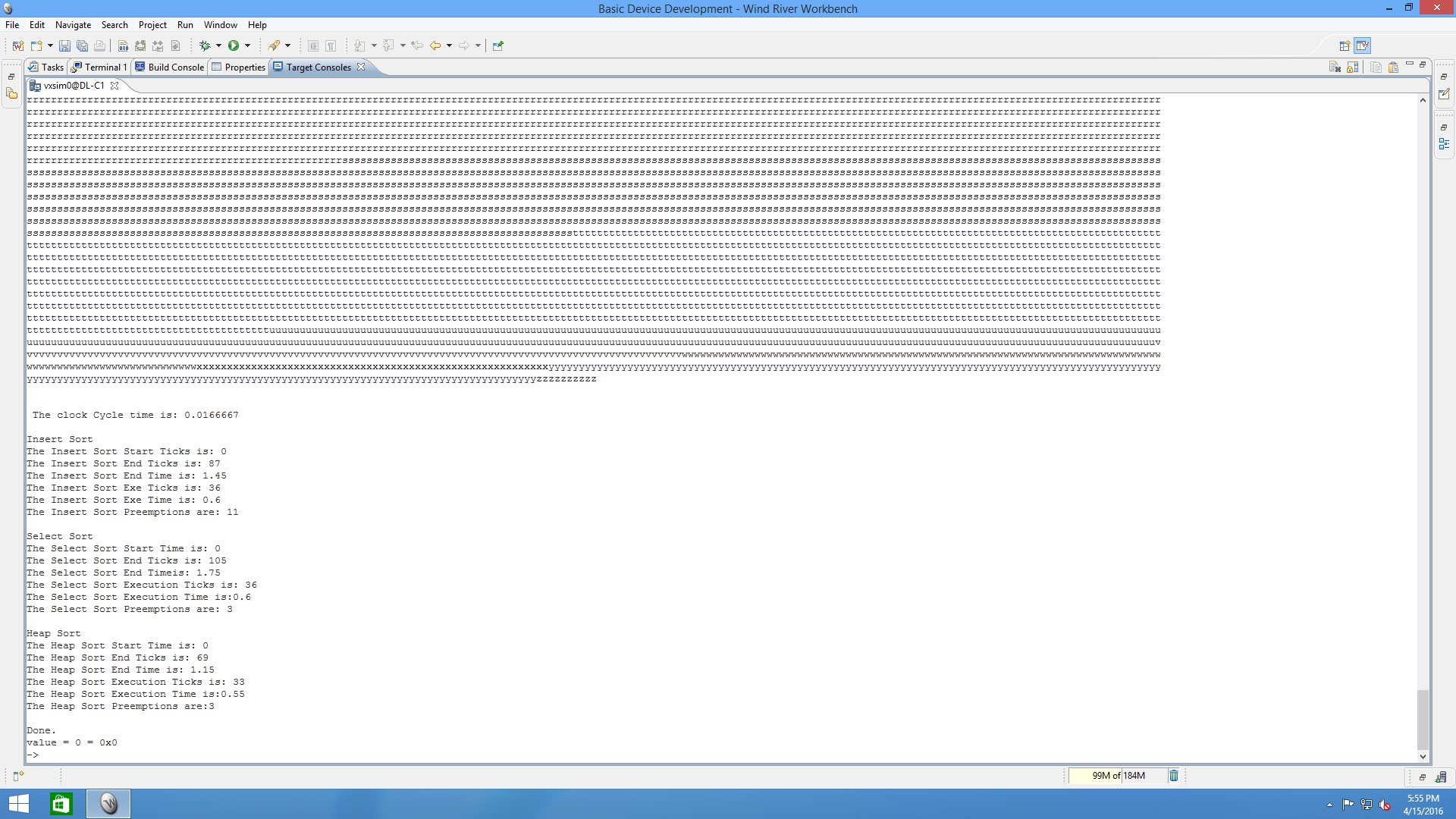
* Here, reshuffling of priorities is necessary. Hence we use a function named “randomize\_priorities()”. This function is used for a simple exchange of values between the task set containing the three tasks.
* Now we need to set priorities dynamically. This is accomplished by shuffling the priorities at certain spots in each sorting function as needed. The reshuffling of priorities is achieved by calling the “randominze\_priorities()” function in:
* Insert Sort
* Select Sort
* Heap Sort
* Then, the task hooks are deleted from the three different tasks using the following functions:
* taskCreateHookDelete((FUNCPTR)tstart)
* taskSwitchHookDelete((FUNCPTR)tswitch)
* taskDeleteHookDelete((FUNCPTR)tdone)
* Finally we output the task information by using the “task\_info\_put()” function.
* This function prints the following information of each task:
  1. Start Time – value saved at task creation
  2. End ticks – value saved at task deletion
  3. End Time (in seconds) - calculated by taking the end time in ticks multiplied by clock cycle time
  4. Execution ticks – value accumulated at task switch
  5. Execution Time (in seconds) – calculated by taking the execution time in seconds
  6. Number of Preemptions – given by number of times each sort is preempted by using a counter variable and incrementing it every time preemption occurs.

We can see that the given input character array string is sorted and the outputs are taken.

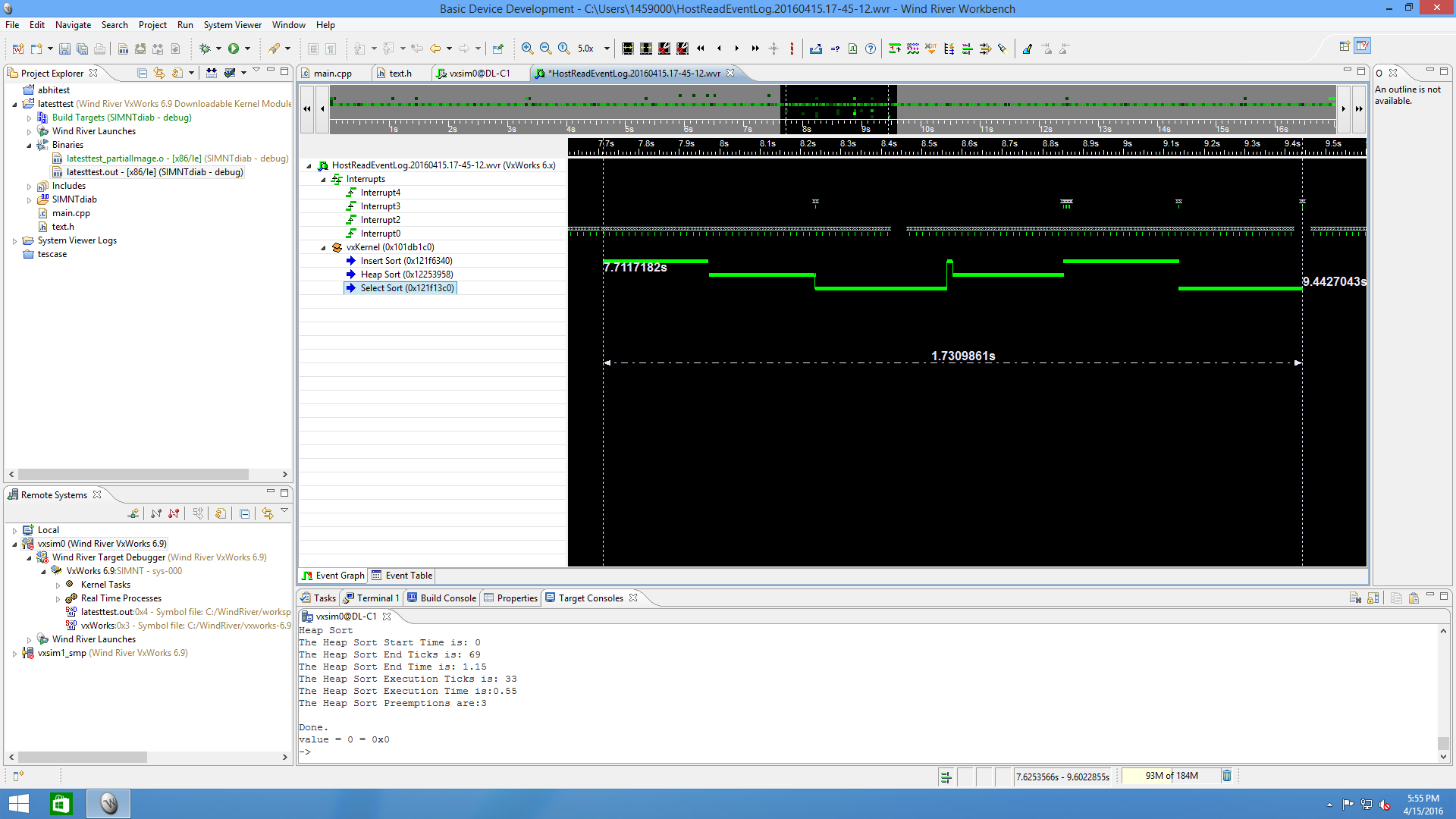
**OUTPUT ON TARGET CONSOLE:**

****

**RESULTS: (OUTPUT)**



Now, we use the File System viewer, we observe the following Event Log (graph):



From the above event log, we can see that the run time for all three tasks is approximately 1.7s which is the end time of select sort (1.75s) which is the last and final task to be executed.

**Run Time and Ticks for each task is:**

The run time for all three tasks is approximately 1.7s which is the end time of select sort (1.75s) which is the last and final task to be executed.

**Time/Ticks**

Insert Sort: 1.45s / 87 ticks

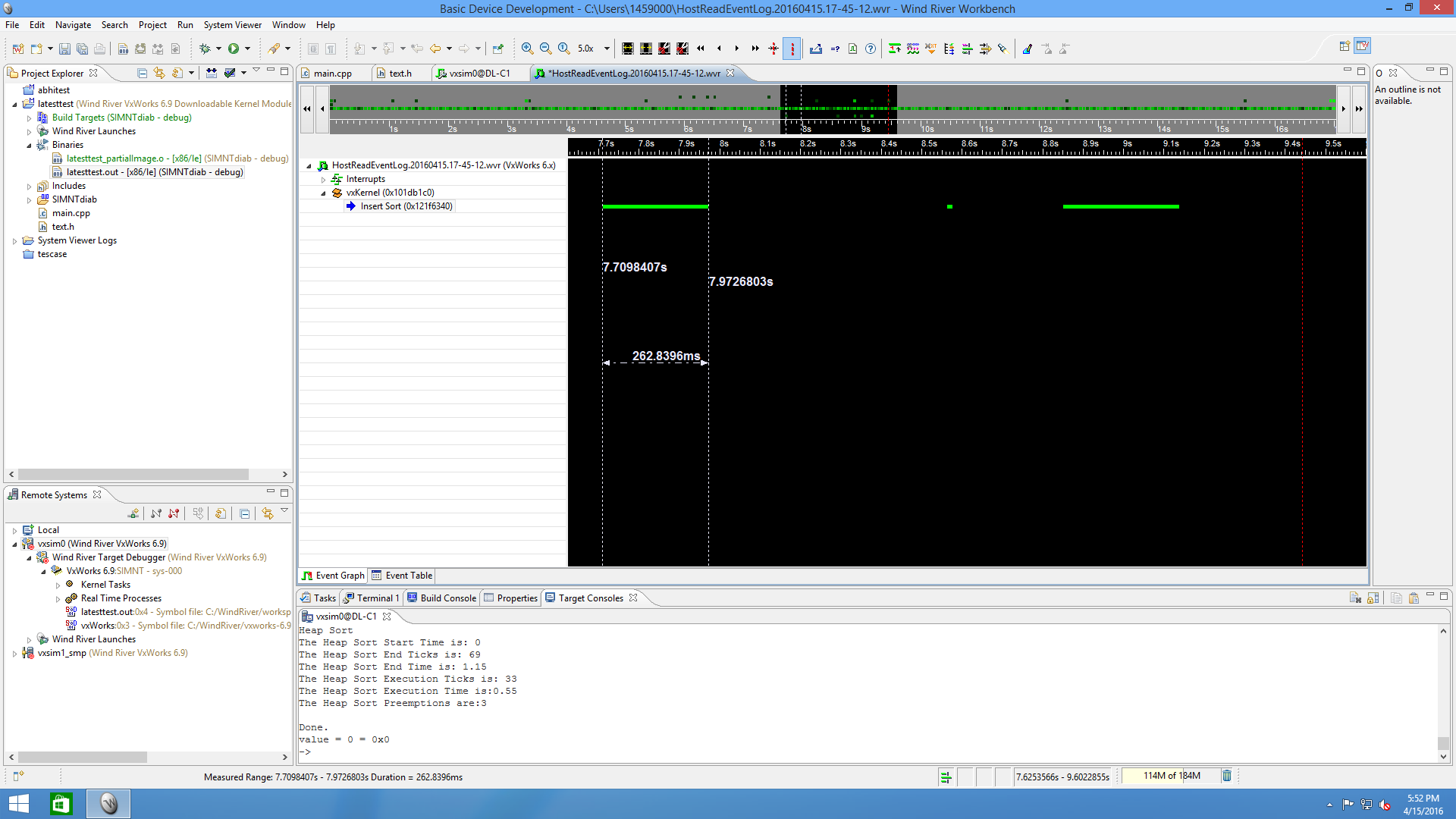
Select Sort: 1.75s / 105 ticks

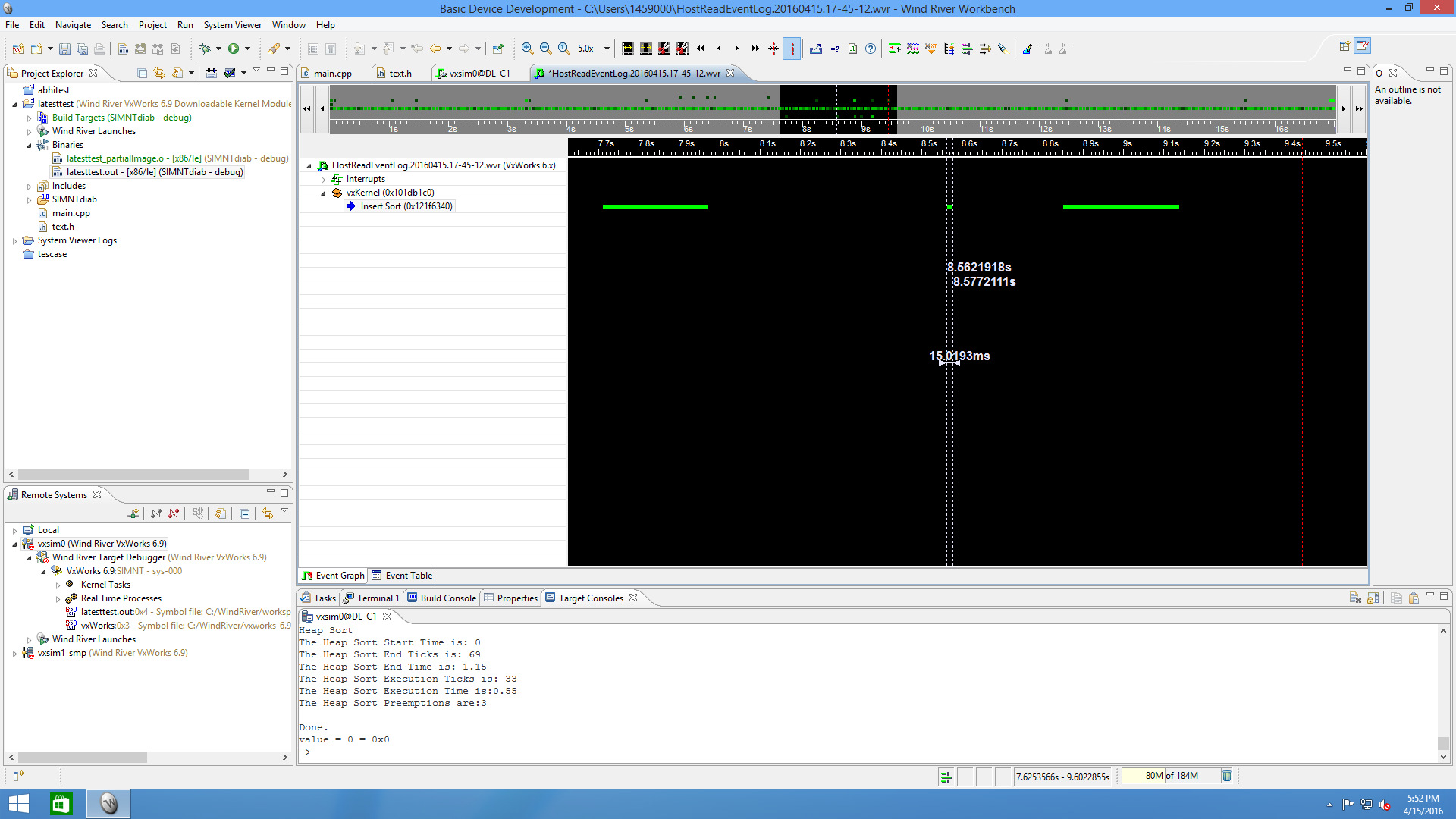
Heap Sort: 1.15s / 69 ticks

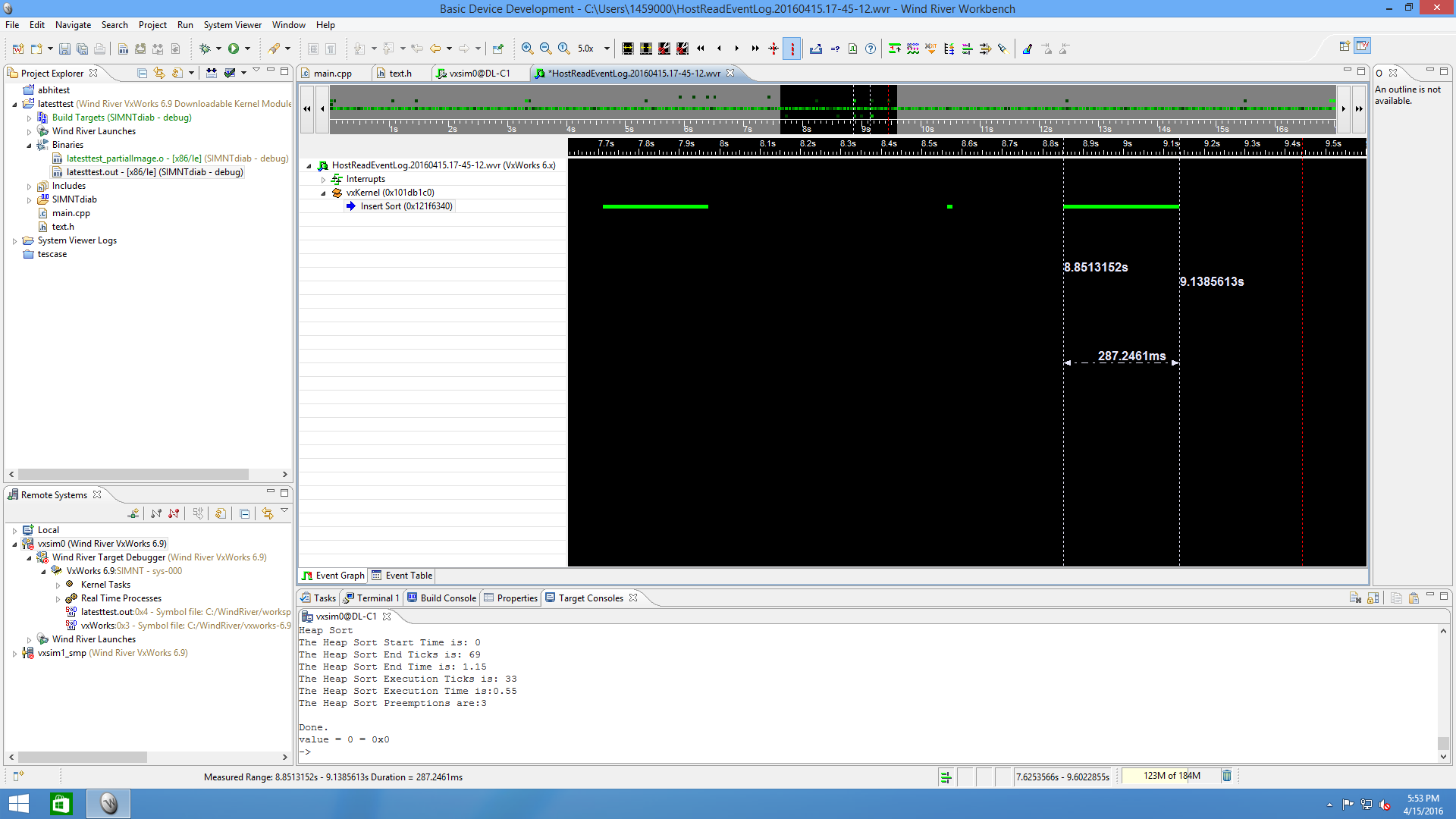
**Execution Time for each task**

To determine the execution time, the program notes the tick value from the last context switch and calculates the difference from the tick value in the current executing context switch to get the number of ticks since the last context switch. This value is accumulated for each task.

**Execution Time for Insert Sort:**

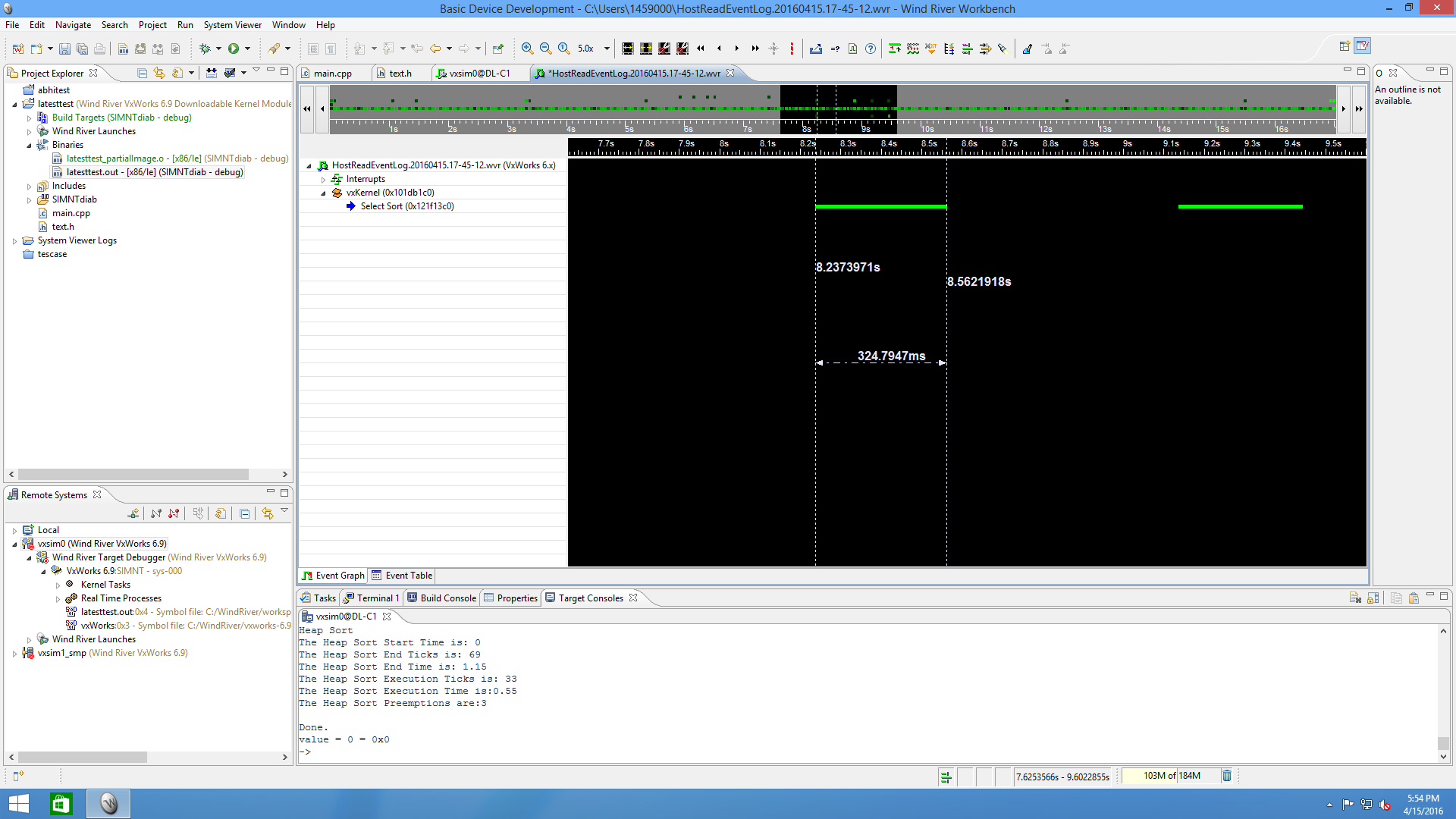
****

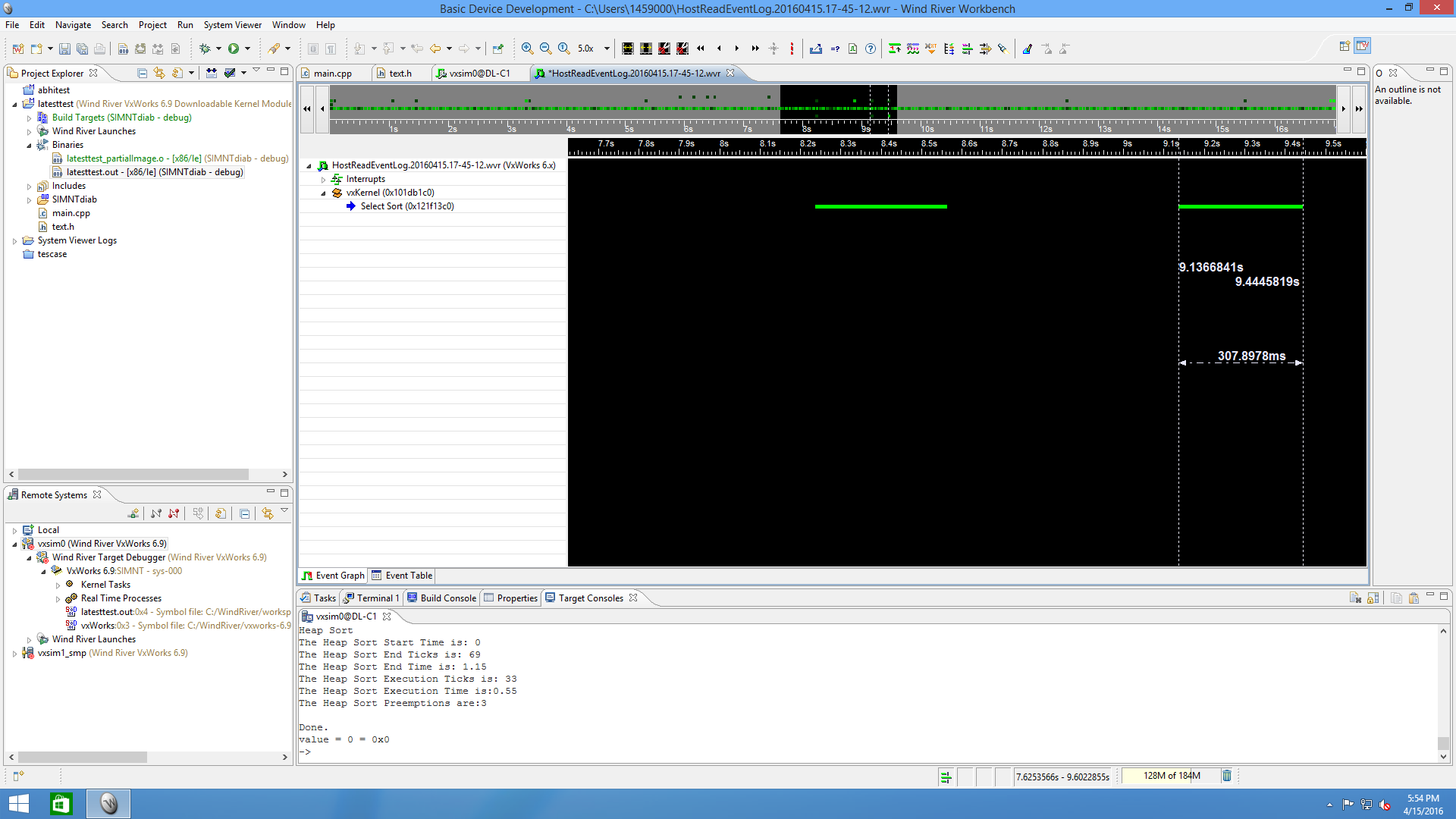
****

****

**Execution Time is for Insert Sort is approximately: 0.6s / Execution Ticks: 36**

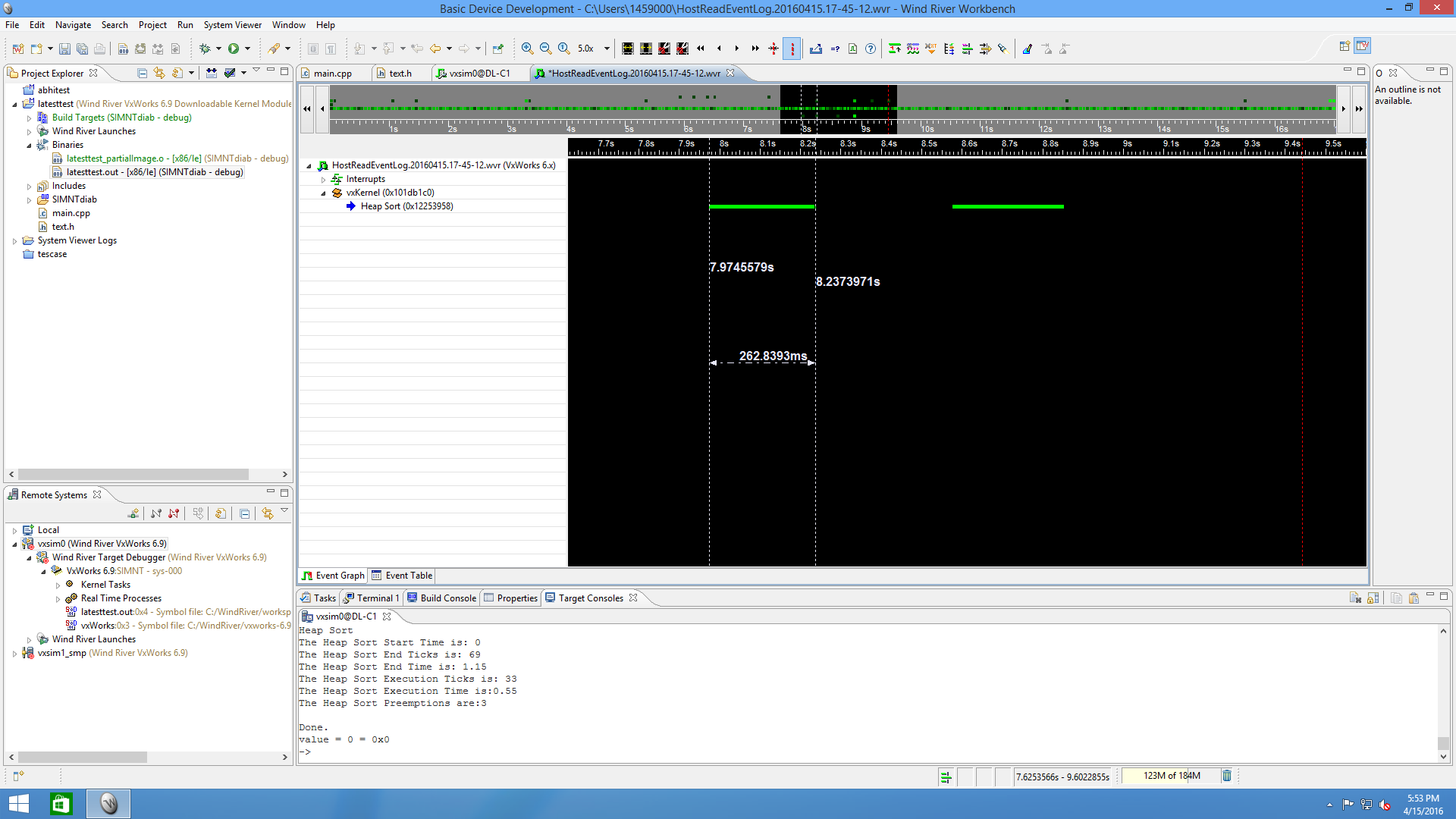
**Execution Time for Select Sort:**

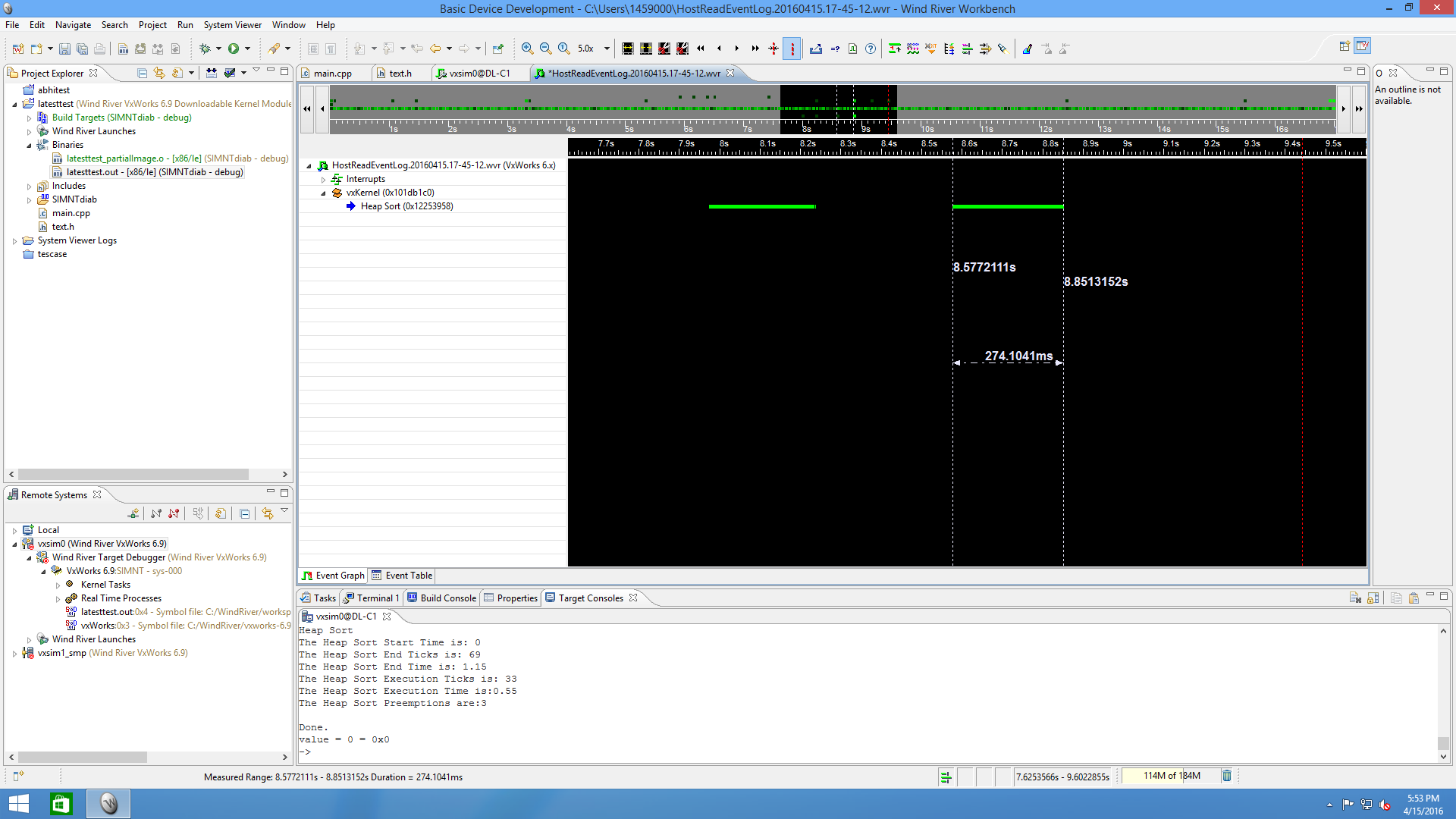
****

****

**Execution Time is for Select Sort is approximately: 0.6s / Execution Ticks: 36**

**Execution Time for Heap Sort:**





**Execution Time is for Heap Sort is approximately: 0.55s / Execution Ticks: 33**

The required tasks are completed.