Simulation-Based Engineering Lab
University of Wisconsin-Madison
Technical Report TR–2023–17

VxWorks A-Stack Progress Report

Deepak Charan Logavaseekaran, Rakshith Macha Billava, and Dan Negrut

Department of Electrical and Computer Engineering, University of Wisconsin – Madison

February 10, 2024

Contents

1 Introduction

The ART autonomous scale vehicle [1], in its current state, relies on ROS for communicating between various nodes. ROS is an open-source framework widely known for its ease of management and its plug-and-play nature for rapid prototyping [2]. Although ROS is highly regarded for its interoperability and modularity, it lacks features to support real-time operations which is crucial for our ART system. In the recent October 2023 ROS Conference, the developers have identified this and have assured that real-time operations will be supported in ROS2 in the future. For our current needs, however, we need to explore and evaluate other real-time solutions to make an educated decision on whether we need to pivot from ROS for applications when we are investigating the gap between simulation and reality. In our lab, the simulation is carried out using the Chrono platform [3], drawing on sensor models described in [4, 5].

VxWorks seems to be an attractive and competitive alternative in this domain. VxWorks is a real-time operating system (RTOS) developed by Wind River Systems. VxWorks focuses on determinism and has a performance-driven approach that aligns with our goals. As some of the NASA rovers have already been using VxWorks, this would be a worthwhile alternative for us to consider. The main intention here is to design a barebone proof of concept model to measure the performance, explore the deterministic behavior and the overall fit of VxWorks for our requirement.

2 Solution Description

2.1 Specifications

For the purpose of evaluation, we will be considering an architecture with just three nodes – the IMU node, the GPS node, and the control node. The IMU and the GPS nodes would be responsible for providing the respective sensor data. The control node performs the sensor fusion, evaluates the current state of the bot, and provides the next command. The approach that we will be taking is to make these tasks time-driven instead of the previously event-driven ROS nodes. In this way, each of these nodes will be independent of each other and will be running on their own time periods. The IMU and the GPS tasks can run at a time period of t1 and t2 intervals respectively. The control task then runs independently at the t3 time period. It takes the most updated data from these sensor tasks and performs the computation. The sensor data is passed by the sensor tasks to the control tasks through message queues which is supported by VxWorks. In addition to this, there also must exist a manager task which is responsible for launching all the tasks and managing the resources.

2.2 Virtualizing Time

We would also like to explore the concept of virtualizing time in our implementation. The A-Stack is expected to be functional when deployed in three subsystems:

- L0, which is the real hardware. The sensors are physically connected to the A-Stack. Data

is read from these sensors and the command is generated. This command then actuates the motors.

- L1, which is the simulation environment wherein the sensor data will be provided by the simulation. The frequency at which the data is now provided tends to be slower than the real sensors.
- L2, which is also a simulation environment. However, the expectation here is that the simulation data appears at a much higher rate than the real hardware. The hope is to have the A-Stack to be functional even with faster frequencies.

The time period of each of the tasks now varies depending on the environment the A-Stack is in. The time period will be scaled appropriately. However, to guarantee the real-time nature of the tasks, we should ensure that the task is completed within its provided time budget or "time capsule". In other words, even though the time periods of these tasks vary accordingly, the task is completed within its time capsule and will wait till it is invoked at the beginning of the next time period. Another key feature that needs to be implemented is the hard deadline. Any failure to meet the time period or the time capsule restrictions should cause the systems to run a recovery phase so that the system is aware of the failure and acts accordingly. However, for our proof-of-concept implementation, we will not be having the recovery phase but we will be exploring ways to enforce these hard deadlines.

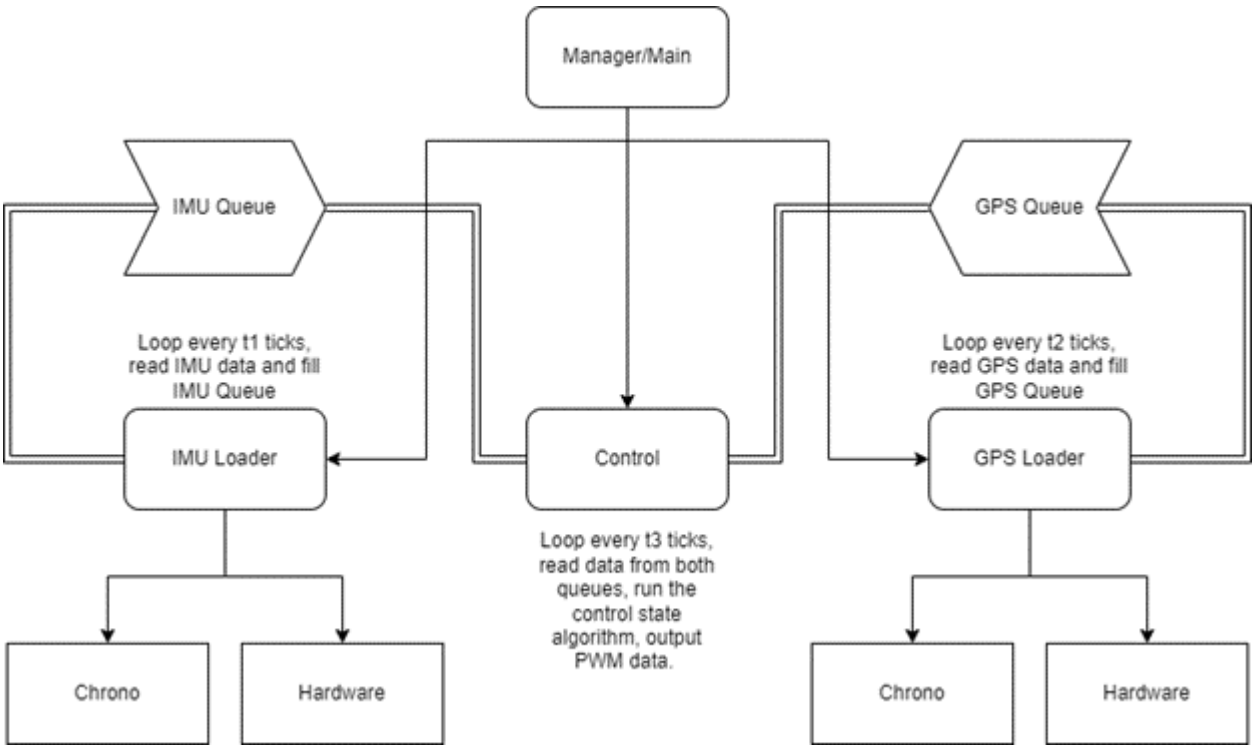

Figure 1: Task Structure of the VxWorks A-Stack

3 Getting Started with VxWorks

3.1 Registering an Account

Before getting started with VxWorks, we will need the VxWorks Development Suite for which we will have to first create a Wind River account. Use this link to register and create a Wind River account - Registering for a Wind River support account.

3.2 License Information

License information during the time of writing the document is as follows -

License information hidden. Please contact SBEL for further information.

3.3 Installation

Download the installer from this link - My Products under the Online Installation subsection. The installation guide can be found here - Installation Guide.

Once this is done, download and install the License Administrator Kit using this link - Wind River License Administration Tools

3.4 Activating the License

License activation consists of two parts - the host and the devices. The host is responsible for launching or hosting the license server. At least one host is required to activate the license. On the other hand, the device is the entity using the license. The license portal can be accessed using this link - License Portal.

To add a new host -

- Click on Manage Hosts and then Add New Host.
- Fill out the details in the form and click on Create.
- Please be careful while filling out the Host ID. This would be the physical address of the host and cannot be changed. The only way to change is to revoke all licenses hosted by the current host, delete the host and create a new host.

To activate a new license -

- Click on Manage Licenses.
- You should find the License already listed. In case it is not listed click on Add New License and add the license.
- If the license is already listed, select Activate Products in the Action dropdown.
- Select the product to be activated and the quantity and then generate the license file. This should be saved as a .lic file.
- Place the license file in the following directory - <vxworks_installation_directory>\license. By default, this should be - C:\WindRiver\license.

- You will now need an include file that adds the current user to the include list. Create a file - wrsd.opt in the same license directory. Add the following into the file - INCLUDE UU_SE_VX7_R4_Cfg1 USER <user_name>, where <user_name>is the user name of the machine.
- Launch lmtools.exe which can be found in C:\WindRiver\license\admintools-1\x86_64-win
- From the Service/License File tab, verify that the Configuration Using Services option is selected.
- Select the checkbox LMTOOLS Ignores License File Path Environment Variables.
- From the Config Services tab, use the Service Name drop-down menu to select Flexlm Service 1.
- Path to the lmgrd.exe file should be provided as - C:\WindRiver\license\admintools-1\x86_64-win\lmgrd.exe.
- Provide the path to the license file.
- Provide the path to the debug log file. This can be arbitrary.
- Check Use Services and Start Server At Power Up.
- Click Save Service and confirm.
- From the Start/Stop/Reread tab, verify that the correct server name is highlighted. Then click Start Server.
- Verify that the license server has started (within 30 seconds). You should see the message "Server Start Successful" at the bottom of the dialog.

4 VxWorks: Key Concepts

This section documents some of the important keynotes after going through the VxWorks training videos. For more detailed documentation please visit - Wind River Learning Portal.

VxWorks software stack mainly consists of three major components:

- VxWorks Source Build Project (VSB) - VSB consists of the default set of layers and libraries for the kernel image. VSB is specific to the CPU and the BSP. The kernel libraries are compiled as a kernel archive.
- VxWorks Image Project (VIP) - This project is used to build the kernel image using the kernel archive. System parameters can be changed here. RTP (real-time components) can be included here using INCLUDE_RTP. When built it generates a VxWorks image file.
- Application Project - Any application that runs on the kernel. It is also possible to add an application to a VIP so that VxWorks starts an application automatically.

5 VxWorks Project Setup for Intel NUC

This section provides a quick summary of the steps required to setup the VxWorks project workspace for running an RTP on the Intel NUC. Launch Workbench 4 and create the workspace in a suitable directory. For this example, the workspace will be created in this directory - C:\Users\sbel\Desktop\VxWorks\a_stack\nuc

5.1 VSB

- Create a VxWorks Source Build project and name it VSB.
- BSP that needs to be selected is - itl_generic_3_0_0_3
- Active BSP should be ALDERLAKE
- Leave the other options as is and click Finish.
- Build the VSB.

5.2 VIP

- Open the Windows command prompt and cd to the VxWorks installation directory. By default it is C:\WindRiver and further instructions will assume the same. Please change it accordingly if needed.
- Run 'wrenv -p VxWorks'. This sets up the environment for VxWorks.
- cd to the newly created workspace and run the command - 'wrtool -data .' This sets the active workspace.
- Run the following command to create a VIP with the name VIP - 'vxprj create -smp itl_generic VIP -profile PROFILE_INTEL_GENERIC -vsb VSB'
- cd into the VIP directory.
- Run - 'vxprj vip component add DRV_CONSOLE_EFI'
- Next run - 'vxprj vip bundle add BUNDLE_STANDALONE_SHELL'
- Also run - 'vxprj vip bundle add BUNDLE_RTP_DEVELOP'
- Build the VIP using the IDE or the CLI.

5.3 RTP

- Create a Real Time Process based on the previously created VIP.
- Delete the existing rtp.c file.
- Add the files from the repo - VxWorks-AStack
- Build the RTP.

5.4 ROMFS

ROMFS project creates a file system on the target device. It makes it easier for the deployment of the RTP on the NUC.

- On the same terminal used to create the VIP, cd into the workspace directory.
- Run the command - 'prj romfs create ROMFS'
- The output of the RTP application is a .vxe file and this needs to be added to the ROMFS project.
- To do this run the command - 'prj romfs add -file <path-to-vxe file>ROMFS'. For example - prj romfs add -file C:\Users\sbel\Desktop\VxWorks\a_stack\nuc\a_stack\ VSB_ALDERLAKEllvm_LP64_ld\a_stack\Debug\a_stack.vxe ROMFS
- Add the ROMFS project to VIP as a subproject - 'prj subproject add ROMFS VIP'

5.5 UEFI Bootloader

- A bootloader is needed to boot the OS. To build the bootloader first cd to - C:\WindRiver\vxworks\23.09\source\boot\uefi
- Run - make clean
- Once that is complete run - make
- You should be able to find BOOTX64.EFI file in C:\WindRiver\vxworks\23.09\workspace\uefi_x86_64 directory once the build completes.

5.6 Deploying the RTP on the NUC

- Format the USB drive to FAT32 format.
- Place the BOOTX64.EFI file in E:\EFI\BOOT
- Copy vxWorks (VxWorks image file) from C:\Users\sbel\Desktop\VxWorks\a_stack\nuc\VIP\default and name it bootapp.sys
- Connect the USB drive to NUC and boot the system. Ensure that UEFI is enabled and Boot from USB is selected.
- Once the device is booted, run - 'cd "/romfs"'.
- Now to run the application - 'rtpSp "a_stack.vxe"'

6 A-Stack Folder Structure

The VSB (VxWorks Source Build project and the VIP (VxWorks Image project) are created in their own directories as separate projects. These files vary for each BSP and need to be created accordingly. The RTP is then created. As we do not have much control over the VSB and the VIP project files, this section discusses only about the RTP project files. The RTP source files are also maintained through a git repository - VxWorks-AStack

- src:
  - This directory contains all of the A-Stack source files.
  - Files that are generic or used by most nodes exist in this directory.
  - Task files are also placed in this directory.
- src\<node>:
  - This directory contains driver files specific to the A-Stack node. For example, src\imu.
  - The driver interface APIs are generally placed in the <node>_driver.h/ .c files.

7 A-Stack Design Decisions

This section discusses several design decisions made during the development of A-Stack and the reason for the decision made. This also acts as a log for all the decisions made that drift away from the original methodology.

7.1 Developing A-Stack as an RTP rather than a kernel application

A kernel application is an application bundled with the kernel image. It executes in the same mode and memory space as the kernel. This also can be configured to start automatically and is ideal for production systems.

Kernel image -

- has access to the full range of memory space.
- runs in the supervisor mode.
- cannot be recovered or protected.
- is capable of invoking system calls comparatively faster than RTP.
- cannot be recovered or protected.
- generally used for drivers.

RTP -

- gains access to the protected memory environment.

- runs in the user mode.
- it has a higher memory footprint.
- it is secure and is standalone.

For the time being, the decision was made to develop A-Stack as an RTP as RTP is comparatively simpler, is secure and more importantly can be recovered in case of a failure which is crucial for the current use case. RTP is more aligned with the current application. However, this discussion can be revisited in the future as the scope of the project is better defined.

To start the kernel application automatically (if needed in the future) -

- Configure with INCLUDE_USER_APPL component.
- Add application entry point function in usrAppInit() API in usrAppInit.c in VIP.

7.2 Using queues for inter-task communication

VxWorks offers the following inter-task communication options -

- Message Passing
  - Queues
  - Pipes
- Shared Memory

Shared Memory is not a viable option because -

- Users need to create access routines
- Synchronization and mutual exclusion need to be provided. High probability of race conditions which could be difficult to resolve
- All tasks would share the same memory space and this might result in thrashing

The current implementation primarily uses queues for inter-task communication for the following reasons (in addition to the above) -

- Message queue library support exists
- Multiple senders and receivers are allowed if needed
- Synchronization and mutual exclusion are provided
- Comparatively lightweight and easier to use than pipes

7.3 Avoiding semaphores

One of the ways hard deadlines can be maintained is through the use of semaphores. The node completes its execution and returns the semaphore. The butler task on the other hand waits on the semaphore with a predetermined capsule time. In case the semaphore is not received and it times out, the application is crashed.

This approach is not used as this makes the application slightly event-driven. The butler relies on the node to return the semaphore. Although the goal to maintain the capsule time is achieved a more strictly time-driven approach is sought out. In addition to this, a more simplistic approach is preferred.

7.4 Steward Task

To maintain the capsule time, a different approach is used. Steward task (previously known as butler task) is used for this purpose. The node records its start time before performing any of its operations. Once its execution is completed, it records its end time. It then sends both the start and the end time to the steward task.

The steward task receives the start and the end time. It then computes the difference and compares it with the predetermined capsule time. If the deadline is not met, it crashes the application. To ensure that the steward task does not wait forever or hang, a timeout is implemented.

This approach is used as it is more time-driven, comparatively simple, scalable, and easier to debug.

7.5 Maintaining time period

VxWorks does not offer an inbuilt method to call a task periodically. So to maintain the time period, a different approach is needed. As a workaround, another task - Time period manager is created. This task just waits for the time period and once the time period is complete it resumes the node.

The node once it completes its execution and sends the time to the steward task, suspends itself thereby waiting for the time period manager to resume it. This ensures that the node is called every time period and the time period is enforced.

References

[1] A. Elmquist, A. Young, T. Hansen, S. Ashokkumar, S. Caldararu, A. Dashora, I. Mahajan, H. Zhang, L. Fang, H. Shen, X. Xu, R. Serban, and D. Negrut, "Art/atk: A research platform for assessing and mitigating the sim-to-real gap in robotics and autonomous vehicle engineering," 2022.

[2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[3] A. Tasora, R. Serban, H. Mazhar, A. Pazouki, D. Melanz, J. Fleischmann, M. Taylor, H. Sugiyama, and D. Negrut, "Chrono: An open source multi-physics dynamics engine," in

High Performance Computing in Science and Engineering – Lecture Notes in Computer Science (T. Kozubek, ed.), pp. 19–49, Springer International Publishing, 2016.

[4] A. Elmquist and D. Negrut, "Methods and models for simulating autonomous vehicle sensors," *IEEE Transactions on Intelligent Vehicles*, vol. 5, pp. 684–692, 2020.

[5] A. Elmquist, R. Serban, and D. Negrut, "A sensor simulation framework for training and testing robots and autonomous vehicles," *Journal of Autonomous Vehicles and Systems*, vol. 1, no. 2, p. 021001, 2021.