Evaluation of embedded virtualization on real-time Linux for industrial control system

Sanjay Ghosh
Industrial Software Systems, ABB Corporate Research
Bhoruka Tech Park, Whitefield Road, Bangalore, India
sanjay.ghosh@in.abb.com

Pradyumna Sampath
Industrial Communications, ABB Corporate Research
Bhoruka Tech Park, Whitefield Road, Bangalore, India
pradyumna.sampath@in.abb.com

Abstract

Real time control applications in industrial control systems have long been trusted to run on specially designed dedicated embedded hardware like PLCs, controllers etc. The essential non real-time functionalities of the automation systems like engineering and HMI are separately executed on independent hardware. Over the last decade, advancements in RTOS, technologies such as virtualization and availability of more powerful COTS hardware have enabled the recent trends, where industrial PCs are being proposed to replace the hardware controller units. This paper describes the evaluation of our prototype control system on a general purpose hardware based on Linux rt-preempt. In our setup, the control logic is executed by a runtime control engine alongside the standard engineering framework, standard HMI, data acquisition, EtherCAT based IO and also general PC utility applications on a uni-processor system without impacting the deterministic performance. The paper discusses in detail, our performance evaluation results and the methodologies used in terms of, the test setup, boundary conditions, and the parameters measured under typical load conditions as in real industrial applications.

1 Introduction and Background

The automation industry has seen a trend in the usage of control systems based on the commodity hardware instead of being based on the standard controller hardware. The key advantages this solution provides are in terms of higher flexibility of configuration and ease of customization. In fact, with the availability of high processing and memory capabilities, PCs can significantly out-perform most of the commercial controllers that are being currently used in the industry, in terms of the available hardware resources to perform tasks. Therefore, control applications running on a resource rich PC-based control may consume a comparatively lesser of the total available CPU than in the case of a traditional controller, leaving a significant resource left over to run other essential non real time applications. Additionally, PCs are available in several configurations and form factors to meet the diverse user needs compared to the controller hardware. The use of commercial PC along with desktop operating systems does not guarantee the deterministic hard real-time requirements for discrete manufacturing. Hard real-time controllers, by their nature are meant to provide fast, deterministic and repeatable scan times without being affected by the other background activities undertaken by the operating system. Typical discrete manufacturing applications require deterministic, repeatable scan times to be as fast as one millisecond. Therefore the control engineering community, as they move to the PC-based control, expects it to deliver similar level of performance and reliability. The challenge yet remains for the PC based control
on one hand to enable all the benefits discussed above and on the other hand to achieve the reliability required of a controller.

1.1 Related Work

Some previous works have been reported in the literature which explores the concepts of PC based control. Work by Magro et.al. [1] include evaluation of time performances of a soft-PLC, OpenPCS by Info Team running on a native operating system Windows2000 with different configurations and loading. In one of the work [2], it was analyzed that, by carefully tuning the Linux rt-preempt based host hypervisor, and using hardware assisted virtualization along with a device emulation application, submillisecond scheduling latencies inside guests can be achieved. In fact, Wurmsdobler [3] in one of his article concludes based on their extensive testing that, for slow processes, Linux supports hard real time without any changes. There have been many real-time benchmarking studies on native operating systems [4], [5], [6] to observe the performance and capabilities of the real time patches. Most of these works conclude that; with appropriate tuning of real time priorities and also employing appropriate kernel configurations, appreciable real time behavior could be achieved, to support industry grade real time application demands. Schild et.al. [7] studied the interrupt-response times of a real-time operating system, Linux rt-preempt, hosting virtual machines using hardware assisted virtualization technology. A mechanism called .coscheduling., [8] i.e. dynamically boosting the guest VMs priority levels, is proposed in order to improve the CPU throughput of a general purpose OS VM on a RTOS host.

In this paper we present the evaluation of our prototype PC based control based on Linux rt-preempt to run real-time control. This additionally allows Windows environment to run on the same hardware thus providing the best of both worlds. Section 2 discusses the requirements for the PC based controller from the perspective of the factory automation domain and describes our prototype in detail. Section 3 describes the evaluation setup based on our prototype PC based controller. Section 4 presents the evaluation result and our observations. Section 5 describes the related work. Section 6 concludes the paper by summarizing our findings and also a note on our vision for future work.

2 Industrial Control on commodity hardware

2.1 Requirements for PC based control

The key high level requirement for a PC based control is to run the control engine and the IO communication as user space tasks in a real time operating environment with a real time execution guarantee. In addition to this, the non real time applications like the application engineering tool, HMI, monitoring program and PC utility applications should be able to run on the same hardware without jeopardizing the real time response. Generally, all the application engineering tools as well as HMI interfaces are packaged as Windows based application. The need for Windows based engineering stations stems from existing knowledge base of using windows for such applications. Furthermore, the user should be able to perform shift reporting, batch reporting etc using the general PC utility applications (mostly Windows based) like Microsoft Office. In such context, we try to define the requirements of a system that is capable of providing an environment to achieve the above functionalities. The control engine [9] executes inside an independent execution environment as a user space application on the operating system and fully relies on the underlying OS to provide infrastructure to ensure determinism.

2.1.1 The requirements on control tasks

There is a clear difference in the resource requirements of the different real time and non real time tasks that are required to be present on the PC based hardware. The execution behavior of the control tasks is usually short or periodic [11]. The control engine executed in a ‘scan cycle’, wherein, it performs input, executes the control logic, and ultimately produces output. One full machine cycle of the control is divided into the execution cycle times and the slack times both adding up to make the fixed interval cycle time. The control engine executes the cycle as fast as possible, and the worst-case loop time determines the response time of the system [3].

2.1.2 The requirements on non-realtime tasks

For factory automation processes, HMI especially is required to be responsive to ensure an expected level of user experience. However, these applications are
scarcely needed during operation of a factory automation system. The resource demands of the control engine execution and should not starve the other non real time tasks, especially the engineering and the HMI.

2.2 Technology alternatives for PC based control

In order to host the control engine we identified and selected Linux rt-preempt [12] over other available real time extensions to the Linux kernel such as RTAI and RT-Linux, mainly based on the requirements for our domain [13]. Other reasons for choosing rt-preempt is that a lot of the features in this patch is has been already included into the mainline kernel in parts. It is definitely important to ensure the availability of the community support that is sustainable over long product life cycles. rt-preempt patch implements real time behavior by allowing nearly the entire kernel to be preempted, with the exception of a few very small regions of code. Further by inclusion of the high resolution timers (hrtimers) hard real-time behavior can be achieved in rt-preempt [14]. Thus the control engine in the PC based control is to be run as user space task in Linux preempt RT with appropriate real-time priorities. In order to achieve co-existence of both the real time and the non real time tasks, a significant degree of isolation in terms of memory space is required. The Linux rt-preempt kernel with its implementation of a virtual memory model provides for this feature with real-time responses. Combining virtualization and real-time gives several use cases for embedded systems. Full virtualization options (such as Intel VT-x) are now commonly available in commodity hardware having x86 [10] architecture.

We also selected Kernel Based Virtual Machine (KVM) [15], [16] which utilizes the hardware virtualization extension to enable the virtualization in our PC based control prototype. Since 2.6.20, KVM as an active open source project has been a part of the mainline kernel as a Linux kernel module with a strong developer community. KVM runs unmodified guest operating systems on the host OS providing each virtual machine to own private virtualized hardware: a network card, disk, graphics adapter, etc.. It relies on the host OS for tasks like scheduling, native interrupt, memory management, hardware management, etc. using a kernel-space device driver (/dev/kvm) and hence categorized as type-2 hypervisor. This uses a user-space component QEMU [17] for all device emulation. KVM adds an operating mode in addition to the default, the kernel and the user modes, in Linux called the guest mode, which in turn has its own kernel and user modes [17]. This resulting guest VMs physical memory can be mapped to the virtual memory of the host hypervisor. Using the corresponding host process for the VM guest process, standard configurations in terms of priority, affinity etc can be configured in order to flexibly influence the scheduling of virtual machines during runtime [2]. This is another advantage which a PC based control using a hardware assisted virtualization over a native operating system scores over the traditional control.

2.3 Prototype PC based control

Based on the requirements for the PC based Control mentioned in section 2.1 and considering the technology solution mentioned in section 2.2, we came out with a prototype PC based Control. For the control engine we selected one of the control engine that provides support for Linux operating system on x86 architecture hardware. Programming IDE runs on Windows guest OS and is based on the open international standard IEC 61131 and is also supplied by the same vendor. The control engine also supports visualization on the device where the HMI is executed as a user space process on the host operating system. The control engine executes as a high real time priority on the Linux rt-preempt based host while the engineering and other PC utility applications are executed inside one or many Windows based guests with comparatively lower process priorities. The execution model of the prototype PC based control is shown below in figure-1.

![FIGURE 1: Execution Model](image.png)

Figure 2 below shows the communication model of the prototype PC based control. The network configuration between the host operating system (Linux rt-preempt) and the guest operating system (Windows) is using the local Ethernet based soft bridge
(bridge configuration using /etc/network/interfaces . auto br0). Communication between the control engine (also known as the Target Device) executing on the host and the programming IDE (also known as the Engineering) executing on the guest is based on the industry standard OPC Server. We used the industry standard real time communication protocol, EtherCAT for IO communication in our prototype. The EtherCAT master stack executes as a high real time priority user space process in the host Linux rt-preempt operating system. This enables the Ethernet network interface card of the PC to communicate to the EtherCAT slave module connected to it using the EtherCAT protocol. An oscilloscope is used to display and verify the real time output from the PC based control.

3 Evaluation Setup

The earlier section described our PC based control prototype in general. Going further, this section describes the specific evaluation setup and configurations of the prototype we used for our experiments.

3.1 System and Runtime Environment

The host system in our evaluation setup consisted of an Intel core two duo processor (Core2 G6950 at 2.8GHz) with 2 GB of RAM memory. The CPU is x86 architecture based and has support for hardware assisted full virtualization with the Intel VT-x technology. The system had a 1Gbps Ethernet network interface card. Host was running the Linux distribution Ubuntu10.04. The host kernel was built out of 2.6.33.7.2-rt30 kernel with the rt-preempt patch. CONFIG_PREEMPT, CONFIG_PREEMPT_RT and the HRTTIMER were enabled and CONFIG_ACPIPROCESSOR was disabled to achieve the best possible latencies. Both the host as well as the guests used the native TSC (constant_tsc flag in /proc/cpuinfo) as clock source so that the time measurements obtained within the guest are reliable. In-kernel tracing functions i.e. CONFIG_LATENCY_TRACE were built in, but were kept disabled during the experiments.[2]

One of the CPU out of the two in the core two duo system was kept disabled during the experiments

<code>echo 0 > /proc/sys/kernel/ftrace_enabled</code>
<code>echo 0 > /sys/kernel/debug/tracing/tracing_enabled</code>

One of the CPU out of the two in the core two duo system was kept disabled during the experiments

<code>echo 0 > /sys/devices/system/cpu/cpu1/online</code>

The experimental setup consisted of two non real time custom created guests with 512MB virtual RAM and a single VCPU created using Virtual Machine Manager application. Both of these virtual machines were based on Windows XP SP3. One of the guests VM runs the application engineering while the other was meant to running PC utility applications. We also disabled the USB legacy option in system BIOS, as it was reported in some earlier work[8] that the USB legacy device is one key factor that causes latencies arising due to SMIs. Instead we used PS/2 keyboard and mouse. One of the experiments requires running the application engineering on a separate networked system. The other PC consists of an x86 architecture based Intel Core i5 CPU, M520, 2.4GHz, 2 GB RAM and running Windows XP SP3.

3.2 Test Applications

The benchmarking control programs referred by the PLCopen [18] have been used for performance evaluation of the PC based control in adherence to
the standard guidelines from the PLCopen. These scripts are written in IEC 61131 language structured text format [19] and are used as standard programs to benchmark the performance of PLCs. The application engineering interface used in our prototype allows application programming using the standard IEC 61131 language. There are two types of benchmarking scripts, for application oriented tests and for language oriented tests. Application oriented benchmarks are used to measure the whole cycle of the control i.e. from receiving an input signal, the internal processing, till writing an output signal. These are a set of different types of applications and their mixtures, which are typically used in the factory automation. Language oriented benchmarks evaluates the computational performance of a controller while performing all the available language constructs in 61131-3 language. We have evaluated our prototype PC based control using seven of the benchmarking programs including both the application oriented tests and the language oriented tests. Almost similar results were observed for all of these tests. For the sake of conciseness, in this paper, we present only the results of one of the experiments i.e. the language oriented test for the control statements. This test evaluates the performance of the PC based control in operation of one thousand instances of different control statements e.g. IF, CASE, FOR, WHILE etc. Repetition time is coded inside the test project scripts using looping. All the experiments were performed for more than two hours with the control engine continuously executing the test program during this duration with different test conditions. The application engineering tool used in the prototype also allows developing visualization applications too. The visualization program executes on the control engine along with the application test program, however with an execution cycle time, typically, two orders larger than that of the control application. We have created a simple visualization program which shows six visualization objects on the screen and linked to these to monitor and update the status of six different variables (i.e. six OPC tags) in the application test program.

While performing the performance evaluation of any real time systems, it is a usual practice to load or stress the system under test in order to observe the performance in such conditions. In order to stress the Linux rt-preempt host, we used the open source tool ‘Stress’ 1.0.4. [20] to run as a background process in order to impose a configurable amount of CPU, memory, I/O, and disk stress on the system. An example of the command line used for executing the stress program is as follows

```
stress -cpu <c> -io <i> -vm <m> -timeout <t>&
```

Similarly, in order to load or stress the Windows based guest VMs, Windows based freeware HeavyLoad 3.0.0.159 [21] was used. In order to stress the system resources, HeavyLoad writes a large test-file to the temp folder, allocates physical and virtual memory and draws patterns in its window.

### 3.3 Evaluation Parameters

For the performance evaluation of the PC based control, we mainly focus on measuring the upper bound or the worst case latencies of the cyclic execution of the control application. The key to appropriate performance evaluation is accurately measuring the cycle time. The actual cycle time is the time span between start and end of a test cycle excluding any possible overheads such as task startup process, IO access time etc. However, system specific overhead (like timer tick and task scheduler) gets included by default to the cycle time measurement. Therefore the time measurements are instrumented within the application test program. At every iteration cycle, at the start of the operations, the current system timestamp is stored in a variable. Then the operations are being performed according as per the application logic. Right at the end of all the operations, the system time is stored again and the time elapsed while executing the operations is being calculated. This is the termed as the Execution Cycle time or more commonly Cycle Time. Based on these measurements per cycle, average, minimum and maximum execution cycle times is calculated. Another measurement parameter which is of interest is the Jitter in the execution of the control logic. This is the measure of how low or how late the execution cycle starts with reference to the desired time of start. Prior to performing the actual performance evaluation experiments, as the first step, it is also important measure the processing capabilities of a control system. In order to estimate the processing capability of the control system for the particular control logic, we execute the test control program with different interval cycle times and the watchdog enabled. Watchdog in this context is defined as a monitor inbuilt within the runtime engine which indicates an exception when the actual execution time of the IEC61131 application exceeds the designated interval time. That is estimated as the minimum possible interval cycle time of the system under test for that particular control application. Further, for all the performance evaluation experiments on the system using that test control application, the interval cycle time is configured to be the minimum
possible interval time.

As mentioned in the description of our evaluation prototype, an oscilloscope is connected to the IO module in order to display and verify the real time output from the PC based control. The IO cycle time (also known as the scan time in the control engineering), for the purpose of all the experiments is kept constant and it is equal to the interval cycle time. Our experimental prototype consists of real time processes such as the control engine and EtherCAT IO master stack; and non real time processes such as a guest VM for running the application engineering tool and another guest VM for running the PC utility applications. Among all the user space processes running on the host system, the control engine process is assigned the highest real time priority followed by the EtherCAT IO master stack process.

In order to limit the size of the test matrix, we decided to focus only upon few specific aspects of the performance evaluation and hence we have fixed few of the variable parameters for all the experiments. For all the experiments we have fixed the real time priorities of the control engine process and the EtherCAT IO master stack process as 68 and 50 respectively. As mentioned in the earlier section, the stress program is used to produce controlled load on the host Linux system. The stress program continuously runs for the whole duration of the experiment with a constant value of the load. Value for the last 15 minutes average load from /proc/loadavg is considered to be the measure of the constant system load. Based on this measure, for the purpose of our experiments, we have defined the categories of the system load level as Light <2, Moderate 5, Heavy 10 and Very Heavy <=20. The reference numbers mentioned in the parenthesis approximately represents the /proc/loadavg values for each of the categories. According to the standard guidelines from the PLCopen each of the standard benchmark control programs needs to be repeated atleast 10,000 times to get accurate results. We ran all our tests for a longer duration of at least two hrs each. If the interval cycle time be 1 millisecond, then during the duration of a test, the control application would run >= 7x106 execution cycles, which is typically large enough to observe the maximum latencies.

4 Evaluation Results and Discussions

The objective of the performance evaluation is to measure the worst case latencies and jitters observed while executing the control applications on the PC based control. Also this gives the performance of the control applications and real time IO communication, when the engineering application interface, HMI and PC utility applications all run on the same hardware.

4.1 Test 1: Interval time evaluation

This experiment was performed in order to approximately measure the performance of the PC based control in terms of the minimum possible interval cycle times it can support for the execution of a control logic. For the PC based control, a comparative measurement was performed among the two possible scenarios. Scenario-1, when the application engineering tool, visualization as well as PC utility applications run on the same hardware with the control engine; scenario-2, when all these applications run on a different system. In this experiment, we tested the execution of the system by step wise reducing the interval cycle time of the control logic till the system throws watchdog exceptions. Following were our observations while performing the experiment for both of the above mentioned scenarios.

<table>
<thead>
<tr>
<th>Interval(µs)</th>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>Watchdog exception at start</td>
</tr>
<tr>
<td>350</td>
<td>Watchdog exception at start</td>
</tr>
<tr>
<td>400</td>
<td>Watchdog exception at start</td>
</tr>
<tr>
<td>450</td>
<td>Watchdog exception after few minutes</td>
</tr>
<tr>
<td>500</td>
<td>Smooth Execution</td>
</tr>
</tbody>
</table>

Table 1: Execution time evaluation for Scenario-1

<table>
<thead>
<tr>
<th>Interval(µs)</th>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>Watchdog exception at start</td>
</tr>
<tr>
<td>350</td>
<td>Watchdog exception at start</td>
</tr>
<tr>
<td>400</td>
<td>Watchdog exception at start</td>
</tr>
<tr>
<td>450</td>
<td>Watchdog exception after few minutes</td>
</tr>
<tr>
<td>500</td>
<td>Smooth Execution</td>
</tr>
</tbody>
</table>

Table 2: Execution time evaluation for Scenario-2

The measurements presented in the table 2 shows that, the PC based control was able to smoothly execute a particular test control application, when configured to run with interval cycle time of 450µs. This is only when the system runs just the control engine and the IO communication run as real time processes in the Linux rt-preempt host. In another scenario,
where the necessary non real time tasks are also run alongside these real time processes, the smooth execution of the control application is found possible only beyond the interval cycle time of 500µs. However, typical factory automation applications require scan times of as small as one millisecond. In both of the above mentioned scenarios, it was observed that the average execution time the system requires for executing one cycle of the test control logic was less than 225ms, i.e., well below even the 50% of the interval cycle time. As the control engine and the IO communication runs as high real time priority processes, these tasks are never preempted from the scheduler by other non real time tasks, during their execution. However, it is during these available slack times the non real time processes are scheduled if required.

4.2 Test 2: Latency and jitter evaluation

This experiment is meant to evaluate the performance of PC based control prototype for reliably running the control applications and real time IO communication, even in the presence of the necessary non real time tasks, running on the same hardware. For the purpose of this experiment we executed the test control application on the PC based prototype under seven different test conditions identified as the seven setup configurations, C1, C2, ..., C7. These configurations are defined based on what all tasks are being run on the system

- Base Configuration (Base): Control engine + IO communication + Visualization on host + Windows VM1 running application engineering tool
- C1: Base
- C2: Base + Low stress on host
- C3: Base + Moderate stress on host
- C4: Base + Heavy stress on host
- C5: Base + Very Heavy stress on host
- C6: Base + Very Heavy stress on host + Max possible load on Windows VM1
- C7: Base + Very Heavy stress on host + Max possible load on Windows VM1 + Windows VM2 running text processing application

Table 3 below presents the measurements of the execution cycle times and jitter for the test control application and scan times for the EtherCAT based output observed using an oscilloscope.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Control Application</th>
<th>External IO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycle Time (µs)</td>
<td>Interval time Jitter (µs)</td>
</tr>
<tr>
<td>Ave</td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>C1</td>
<td>219</td>
<td>206</td>
</tr>
<tr>
<td>C2</td>
<td>219</td>
<td>206</td>
</tr>
<tr>
<td>C3</td>
<td>218</td>
<td>210</td>
</tr>
<tr>
<td>C4</td>
<td>228</td>
<td>209</td>
</tr>
<tr>
<td>C5</td>
<td>224</td>
<td>208</td>
</tr>
<tr>
<td>C6</td>
<td>224</td>
<td>209</td>
</tr>
<tr>
<td>C7</td>
<td>229</td>
<td>209</td>
</tr>
</tbody>
</table>

FIGURE 3: Execution Model

The measurements presented in the table 3 shows that, the PC based control was able to accommodate the execution of typical non real time tasks required in a control system, on the same hardware, without compromising the real time guarantee of the control applications. Further, it was also observed that, even by deliberately applying heavy loads on the host system as well as the guest systems, there is only a negligible change on the real time execution behavior of the control application.

In one of the scenario, the reliability of the isolation between the Linux rt-preempt host and the Windows based guests partitions was also evaluated. Even during the deliberate crashing or during re-booting of the Windows guests, the control engine and the real time IO continues to perform unaffected in terms of cycle times and jitter.

5 Conclusion and Future Work

Our experiments with the prototype PC based control device has demonstrated that; it is possible to be able to achieve deterministic responses by using Linux rt-preempt along with KVM as the host RTOS. Irrespective of the concurrently running application load on the windows guests, the deterministic behavior of the user space applications running on the rt-preempt kernel is not affected. The choice of real-time tasks and their priorities must still be carefully managed and the host and guest must still follow the traditional separation of concerns i.e. real-time and non-real time respectively. The results of the tests performed also indicate that, such a system may be conceivable for certain industrial application domains, but maybe inappropriate for applications
which demand more stringent real-time constraints (such as closed loop motion control).

Going forward we believe that there is potential for further work in the area. One such activity might involve the confluence of multi-core, virtualization and real-time. Comparative studies between SMP Virtualization and AMP virtualization for real-time systems is an area where the industry and academia might see benefit.

References


