POWERLINK and Real-Time Linux: A Perfect Match for Highest Performance in Real Applications

Josef Baumgartner
Bernecker + Rainer Industrie-Elektronik Ges.m.b.H
B & R Strasse 1, 5142 Eggelsberg, Austria
josef.baumgartner@br-automation.com

Stefan Schoenegger
Bernecker + Rainer Industrie-Elektronik Ges.m.b.H
B & R Strasse 1, 5142 Eggelsberg, Austria
Stefan.Schoenegger@br-automation.com

Abstract

In the automation industry many discussions around the various Industrial Ethernet concepts like POWERLINK, Profinet and EtherCAT are based on theoretical performance studies. This paper will outline the in reality achievable performance of the open-source POWERLINK technology operating on a standard x86 PC with real-time Linux and its potential application scenarios. The evaluation will include a study on the synchronization quality as well as on the resulting CPU load for large scale applications generated by the network protocol.

1 Introduction

Ethernet has been the standard networking technology in the home and office environment for years. In the automation industry, the conventional field busses are still the dominant communication technology. This is because standard Ethernet could not provide the deterministic behaviour required by many industrial applications. In the meantime there are several ethernet based fieldbusses available on the market. Whereas some provide only soft-realtime capabilities, like PROFINET or EtherNet/IP, some of them are able to fullfill hard real-time requirements needed for industrial applications.

One of these hard real-time Industrial Ethernet protocols is *POWERLINK* [11]. The availability of the openPOWERLINK network stack for the Linux operating system makes it very easy to implement a software based industrial control application on top of an standard PC running Linux. However, to provide sufficient accuracy for the network timing the operating system must provide some kind of real-time capabilities. Therefore, the Realtime Preemption Patch (RT-Preempt) provided by Ingo Molnar [7] is an ideal base to implement a deterministic POWERLINK master (Managing Node) using Linux.

Whereas most performance values for other ethernet based fieldbusses are very theoretical, this paper shows the real cycle times which could be reached using openPOWERLINK on a real-time Linux platform. The system load created by the protocol stack is analyzed and the accuracy of the POWERLINK network synchronisation is evaluated.

2 POWERLINK

2.1 Communication Principle

POWERLINK is a strict, deterministic real-time protocol based on Fast Ethernet (100 MBit) [1]. Time-isochronous transfer of data is supported along with asynchronous communication between network

nodes; a part of network bandwidth is reserved for this. Figure 1 shows a POWERLINK communication cycle.

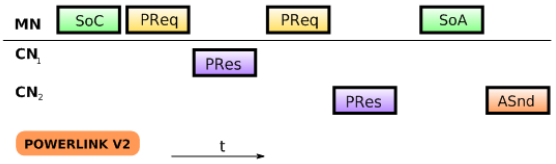

FIGURE 1: *POWERLINK cycle*

A POWERLINK device can be a managing node (MN) or a controlled node (CN). A POWERLINK network has exactly one MN. This regulates activity on the network. All other devices in the network are CNs. The *SoC* is sent as a multicast and can be received and processed by all other POWERLINK stations in the network. No application data is transported in the *SoC*, it is only used for synchronization.

Immediately after transmitting the SoC, the MN addresses each CN in the network with a *PReq* (poll request). Each CN responds with a *PRes* (poll response). The output data designated for a CN is transmitted in the PReq. All stations are addressed in order by the MN with a PReq. Immediately upon receiving the PReq, the addressed station responds with a PRes. This frame is sent as multicast and can therefore be received by the MN as well as by all other CNs in the network. Therefore, the PRes can not only send input data from the CN to the MN, but also allows cross-communication among the CNs. Direct cross-communication allows the times for data exchange between stations to be reduced considerably, since the data need not be copied in the MN.

A CN only transmits when it receives a directly addressed request (PReq) from the MN. The MN waits for the response from the CN. This prevents collisions on the network and enables deterministic timing.

A fixed time is reserved in the network cycle for asynchronous data. Asynchronous data differs from cyclic data in that it need not be configured in advance. Asynchronous data is generated on-demand by a POWERLINK station. Examples are visualization data, diagnostic data, etc. One asynchronous frame can be sent per POWERLINK cycle. The CNs can signal the MN in the poll response frame that they would like to send asynchronous data. The MN determines which station is allowed to send, and shares this information in the *SoA* (Start of Asynchronous) frame. Any Ethernet frame can be sent as an asynchronous frame (ARP, IP, etc.). However, a maximum length (MTU = Maximum Transfer Unit) must not be exceeded.

2.2 PollResponse Chaining

The POWERLINK protocol supports an additional mode called *PollResponse Chaining*. Instead of requesting the CNs sequentially through PReq frames, the CNs are requested alltogether by the PResMN frame which is sent as multicast. The data, usually sent by the MN in the PReq frames, is mapped into the PResMN frame of the MN. This increases performance if many nodes with small amount of process data are connected, because instead of sending many small packets only one packet containing the data for all CNs needs to be sent. In the conventional POWERLINK cycle a CN is only allowed to send a PRes frame after receiving its PReq frame. With PRes Chaining this rule is obsolete. Now the PRes frame is sent time triggered. Each CN is configured by the MN to send its PRes frame at a specific point in time. It is still possible to use conventional PReq/PRes nodes in combination with PResChaining nodes. Figure 2 shows a POWERLINK cycle with both PRes Chaining and conventional nodes. PollResponse Chaining is specified in [2].

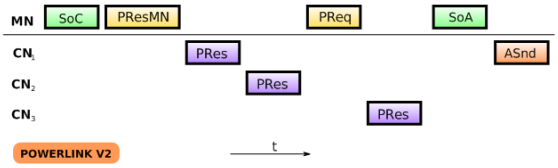

FIGURE 2: *POWERLINK cycle with PollResponse Chaining*

2.3 Synchronization Parameters

Because of the POWERLINK communication principle there is one critical timing parameters in a POWERLINK communication cycle, the SoC Jitter.

The MN generates the SoC frame to start a new POWERLINK cycle. For a software solution the accuracy of the SoC generation is mainly determined by the operating system and its network stack. A high-resolution timer is required to provide an accurate cycle timing. Additionally the delay generated in the network driver from receiving the packet until it is sent out to the network determines the cycle quality.

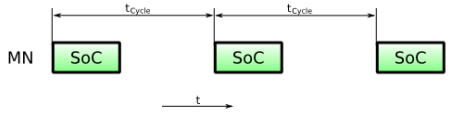

FIGURE 3: *POWERLINK SoC timing*

3 Linux

The requirements of industrial automation systems generate very high demands on the operating system. For this purpose, POWERLINK was mainly implemented on real-time operating systems such as VxWorks in the past. As the real-time capabilities of the Linux operating system were enormously enhanced in the last years, it grew up to a comparable alternative platform for implementing a POWERLINK MN as a baseline for a competitive automation target.

3.1 Realtime Preemption Patch

The standard Linux kernel only meets soft real-time requirements but there are several real-time extensions available for Linux right now. One of them is the Realtime Preemtpion Patch (RT-Preempt) developed by Ingo Molnar. Unlike other Linux real-time extensions RT-Preempt doesn't use a micro-kernel but brings hard real-time capabilities directly into the Linux kernel. The big advantage of this solution is that the user can use his standard linux tools for development, using the POSIX API for his applications and doesn't need to lern special real-time APIs.

3.2 High Resolution Timers

Precondition for an accurate SoC timing in a POWERLINK MN is a very accurate system timer. The high-resolution timers introduced by Thomas Gleixner are part of the Linux kernel since 2.6.16. The new timer system does no longer depend on the periodic tick of the operating system and allows nanoseconds resolution. However, the resolution depends on the available timer hardware of the system. On an Intel X86 architecture there are different clocksources available (hpet, tsc, acpi_pm) which provide a usable timer resolution in the microsecond range.

3.3 Interrupt Load

Because the network load for POWERLINK is very high and many small packages will be transferred across the network the interrupt load is very high. Therefore, effective interupt handling is required in the operating system. Furthermore, the performance could be enhanced if the hardware provides interrupt throttling and the network driver is designed to supports this function.

3.4 openPOWERLINK Stack

The openPOWERLINK stack is a POWERLINK stack developed by SYS TEC electronic. SYS TEC published the POWERLINK stack under the Open-Source BSD license[11]. openPOWERLINK contains all functionalities and services required for implementing a POWERLINK MN and CN. It runs on Linux and other operating systems and platforms. Although there are Linux solutions available for other Ethernet based fieldbusses, these are mostly Linux drivers for proprietary hardware. With the openPOWERLINK stack a pure software based solution is available which runs on a standard PC and no proprietary hardware is needed.

Figure 4 shows the software architecture of the openPOWERLINK stack. The Linux implementation of the openPOWERLINK stack runs completely in kernel space. The interface to the user space application is provided by the EPL API Layer.

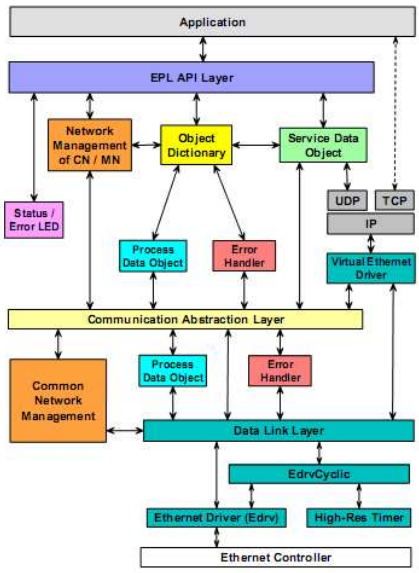

FIGURE 4: *openPOWERLINK software architecture*

To provide maximum performance the openPOWERLINK stack does not use the Linux network drivers but provides its own optimized network drivers.

4 Performance Evaluation

In our evaluation we analyzed the lowest POWERLINK cycle times which could be achieved on a Linux POWERLINK MN with the current openPOWERLINK stack and how much system load it generates. Additionally we measured the quality of the POWERLINK timing on the network.

4.1 Test Environment

The following test setup was used for the evaluation:

- MN: APC810 industrial PC
- CNs: B&R X20 BC0083 (X20 DI4371, X20 DO4322)
- B&R POWERLINK Analyzer X20 HB8815

Figure 5 shows the testsytem with three bus controllers connected to the MN.

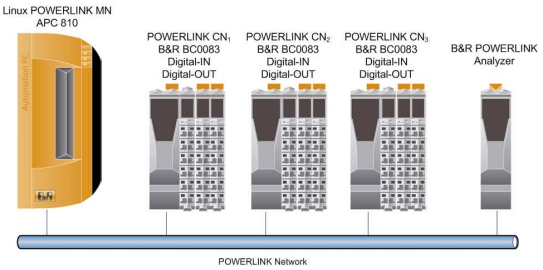

FIGURE 5: *POWERLINK Test System*

4.1.1 POWERLINK MN

The POWERLINK MN test systems were implemented on B&R APC810 industrial PCs.[3] We used two differently equipped PCs to compare the results of a high-end industrial PC with a solution in the range of current embedded platforms.

High-End Industrial PC

The first APC810 was equipped with a Intel Core2Duo U7500 dual core processor running at 1.06 GHz, 1 GByte DDR2 PC2-5300 DRAM and a 40GB harddisk drive. The Intel 945GME chipset contains the Graphics Media Accelerator GMA 950. The onboard network interface based on a Realtek 8111B Gigabit Ethernet adapter was used for the network stress tests. The POWERLINK network was connected through the second onboard Intel 82573L based Ethernet controller.

Embedded PC

The second APC810 was equipped with a Intel Celeron M 423 processor. The processor clock was reduced to 533MHz to simulate the processing power of an embedded system. The remaining hardware configuration was the same as with the first APC810.

Software

The installed operating system was a 32-bit version of Ubuntu 10.04LTS Desktop running a 2.6.31.12-rt21 kernel. The current openPOWERLINK network stack version 1.7 was installed.

4.1.2 POWERLINK CNs

For the POWERLINK CNs, B&R X20 BC0083 bus controllers [4] were used. A digital input modul X20DI4371 [5] and a digital output modul X20DO4322 [6] was connected to each bus controller. The DI4371 module is equipped with four digital inputs, the DO4322 module is equipped with four digital outputs. In contrast to other systems a B&R POWERLINK CN is not restricted to a few I/O ports. If additional I/O was needed, one would typically add additional I/O modules to one node. Up to 253 I/O modules could be connected to a single bus controller. As we would like to evaluate the performance on differently sized networks we used a changing amount of bus controllers and connected only one digital input and one digital output module. The PResMN frame from the MN contains the data for the digital outputs. The PRes frames from the CNs contain the data of the digital inputs and some additional status information. Table 1 shows the size of the payloed for the differently sized networks.

Number of CNs	**3**	**10**	**20**	**40**
Input size in Bytes (Sum of all CNs)	18	60	120	240
Output size in Bytes	3	10	20	40

TABLE 1: *Payload Size of test system*

4.1.3 POWERLINK Analyzer

Due to the limited accuracy, network timing measurment with WireShark was not sufficient. Therefore, a B&R POWERLINK analyzer was connected in order to get high quality network timing measurements. The implementation of a special MAC controller (openMAC) in a FPGA makes it possible for the POWERLINK analyzer to measure timestamps of network frames with a resolution of 20ns.

4.2 Cycle Time and System Load

To evaluate which cycle times could be achieved and how much system load the POWERLINK stack generates with differently sized networks we connected a changing amount of CNs to the POWERLINK MN and measured the system load. Table 2 and 3 show the results of the system load measurement.

	Number of CNs			
	3	10	20	40
250 μs	37%	N/A	N/A	N/A
500 μs	18%	28%	43%	N/A
1 ms	8%	14%	21%	39%
2 ms	3%	5 %	9%	18%
5 ms	< 1%	1 %	4%	6%
10 ms	< 1%	< 1%	< 1%	2%

TABLE 2: *System Load Measurement, High-End PC*

	Number of CNs			
	3	10	20	40
250 μs	50%	N/A	N/A	N/A
500 μs	25%	29%	50%	N/A
1 ms	11%	14%	23%	41%
2 ms	5%	7%	11%	19%
5 ms	< 1%	2%	3%	7%
10 ms	< 1%	1%	1%	3%

TABLE 3: *System Load Measurement, Embedded PC*

Cycle times of 250 μs could be reached on both systems. The measured system load is the load of all POWERLINK threads on a single core. This means that the overall system load on the dual core system is much less and leaves enough processing power for applications.

4.3 SoC Timing Evaluation

4.3.1 Methodology

We measured the SoC timing accuracy while the system was stressed with different stress tests. Figure 5 shows the test system which was used for the measurements. The following stress tests were applied:

1. **Idle**
   The first measurement was done on an idle system as a reference for the different stress tests.

2. **CPU load**
   For the CPU stress test, the tool *cpuburn* was used [9]. It is designed to load X86 CPUs as heavily as possible for the purposes of system testing.

3. **Hard Disk I/O Load** The tool *dd* was used to read and write large amounts of data from and to the hard disk drive.

4. **USB I/O Load**
   As for the hard disk, *dd* was used on an USB drive to produce USB I/O load.

5. **Network Load**
   Heavy network stress was caused by an external *flood ping* on the first Ethernet interface.

6. **Scheduling load**
   Heavy process scheduling load was caused by *hackbench* [10]. It spawns over a hundred processes which are communicating by sending signals to each other.

7. **Miscellaneous Load**
   To cause miscellaneous system load a linux kernel compilation was started.

4.3.2 Results

The following section shows the results of the SoC jitter measurements.

High-End System, Intel Core2Duo

The following test parameters were applied:

Reference Cycle Time: 500 μs
Measured Cycles: $10 \cdot 10^6$
Clock Source: hpet
Linux Kernel: 2.6.31.12-rt21

Stress Tests	Min Cycle	Max Cycle	Deviation
Idle	460.3 μs	548.8 μs	48.8 μs
CPU	474.6 μs	525.9 μs	25.9 μs
Hard Disk I/O	451.2 μs	552.6 μs	52.6 μs
USB I/O	443.5 μs	556.5 μs	56.5 μs
Network	438.1 μs	560.4 μs	61.9 μs
Scheduling	447.4 μs	553.2 μs	53.2 μs
Miscellaneous	445.7 μs	552.4 μs	54.3 μs

TABLE 4: *SoC Jitter, Intel Core2Duo*

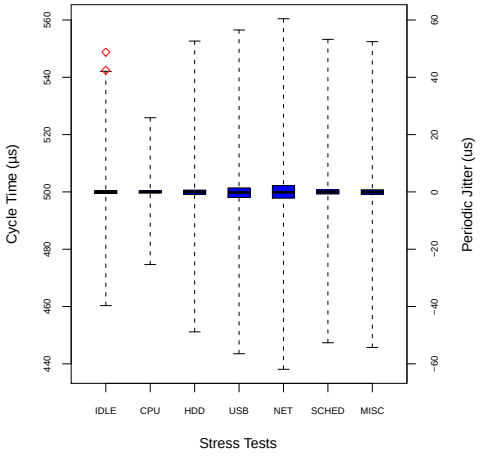

FIGURE 6: *SoC Jitter, Intel Core2Duo*

Embedded System, Intel Celeron

The following test parameters were applied:

Reference Cycle Time: 500 μs
Measured Cycles: $10 \cdot 10^6$
Clock Source: hpet
Linux Kernel: 2.6.31.12-rt21

Stress Tests	Min Cycle	Max Cycle	Deviation
Idle	396.1 μs	597.4 μs	103.9 μs
CPU	473.2 μs	532 μs	32 μs
Hard Disk I/O	400.3 μs	603.2 μs	103.2 μs
USB I/O	397.2 μs	609.2 μs	109.2 μs
Network	456.8 μs	543 μs	43.2 μs
Scheduling	463.7 μs	533.1 μs	36.3 μs
Miscellaneous	402.5 μs	600.5 μs	100.5 μs

TABLE 5: *SoC Jitter, Intel Celeron*

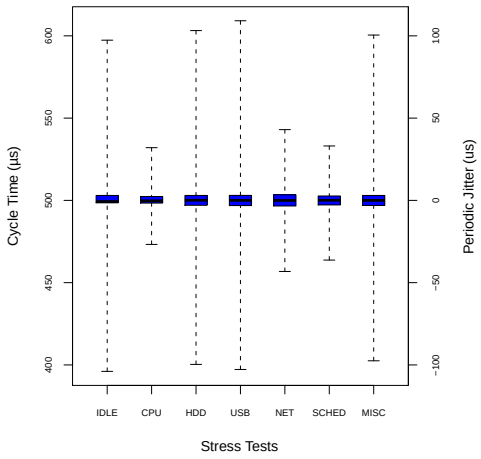

FIGURE 7: *SoC Jitter, Intel Celeron*

Conclusion

The measured SoC jitter is in the expected range. On the high-end system a maximum deviation of 61.9 μs could be reached. On the embedded system a maximum deviation of 109.2 μs could be reached. However, it was not clear why the jitter was so high on an idle system and gets smaller if the CPU is heavily loaded.

5 Conclusion and Future Work

The performance evaluation showed that the Linux operating system together with RT-Preempt is an ideal platfrom for implementing a high-quality POWERLINK MN. The high-resolution timers ensure a very high cycle time accuracy which is sufficiant for many industrial applications. However, it is not clear why we get the best results for the SoC jitter measurement when the CPU is heavily loaded. This needs further investigation.

The cycle time could be lowered down to 250 μs which allows the implementation of industrial systems which require very low cycle times, such as motion control systems. The generated system load is low enough to implement small systems using an embedded platform or medium systems by using high-end industrial PCs. Due to the measured values there is evidence that even larger networks could be realized without problems. We will continue testing with larger networks and other architectures in the future to provide comprehensive performance values.

With the openPOWERLINK stack an Open-Source solution is available for Linux which allows

everyone to implement a cost effective industrial control solution on top of a standard x86 PC. The current implementation of the openPOWERLINK stack implements its own proprietary network interface driver. Whereas this assures the maximum performance it limits the stack to use one of the few network cards, supported at the moment. To avoid implementing drivers for the huge amount of network cards available on the market, the design of the stack should be changed to use standard Linux network drivers. This may require some optimizations in the network subsystem like preallocated SKBs or a optimized traffic shaper to get the needed performance for a deterministic real-time Ethernet protocol. Additionally this whould assure compatibility with future kernel versions.

B&R will continously drive the further development of the openPOWERLINK stack on Linux and its long term goal will be to bring POWERLINK functionality into the official kernel sources enabling everyone using a standard Linux machine to use it for industrial control applications.

References

[1] *EPSG Draft Standard 301, Ethernet POWERLINK, Communication Profile Specification*, 2008, Ethernet POWERLINK Standardisation Group, V 1.1.0

[2] *EPSG Working Draft Proposal 302-C, Ethernet POWERLINK, Part C: PollResponse Chaining*, 2009, Ethernet POWERLINK Standardisation Group, V 0.0.3

[3] *APC 810 User's Manual, Version 1.20*, October 2009, Bernecker + Rainer Industrie-Elektronik Ges.m.b.H, Austria

[4] *X20 System User's Manual, Version 2.10, 9.6 BC0083*, Bernecker + Rainer Industrie-Elektronik Ges.m.b.H, Austria

[5] *X20 System User's Manual, Version 2.10, 9.6 DI4371*, Bernecker + Rainer Industrie-Elektronik Ges.m.b.H, Austria

[6] *X20 System User's Manual, Version 2.10, 9.6 DO4322*, Bernecker + Rainer Industrie-Elektronik Ges.m.b.H, Austria

[7] *The RT Wiki, CONFIG PREEMPT RT Patch*, https://rt.wiki.kernel.org/index.php/CONFIG_PREEMPT_RT_Patch

[8] *The RT Wiki, High resolution timer design notes*, https://rt.wiki.kernel.org/index.php/High_resolution_timer_design_notes

[9] *The cpuburn homepage*, http://pages.sbcglobal.net/redelm/, Robert Redelmeier

[10] *Hackbench homepage*, http://devresources.linux-foundation.org/craiger/hackbench/

[11] *openPOWERLINK Protocol Stack Source*, http://openpowerlink.sourceforge.net/