WHITEPAPER

Multiprotocol Chip Interface for Industrial Communication

Hilscher Gesellschaft für Systemautomation mbH
www.hilscher.com
DOC140101AN01EN | NM | 2014-05 | Released | Public

Table of Contents

1 Introduction

The paradigm shift in embedded industrial designs from traditional Fieldbus to real-time Ethernet is regarded as transitional process. This typically derives from existing infrastructures and lifecycle management. One of the key reasons that real-time Ethernet has outpaced the traditional Fieldbus in factory automation is interoperability, which is perceived as a fundamental driver in industry evolution. Though, the desire for one common industrial standard from process down to field control has not been accomplished, rather different real-time Ethernet variants emerged to fulfill the individual needs of various segments and markets.

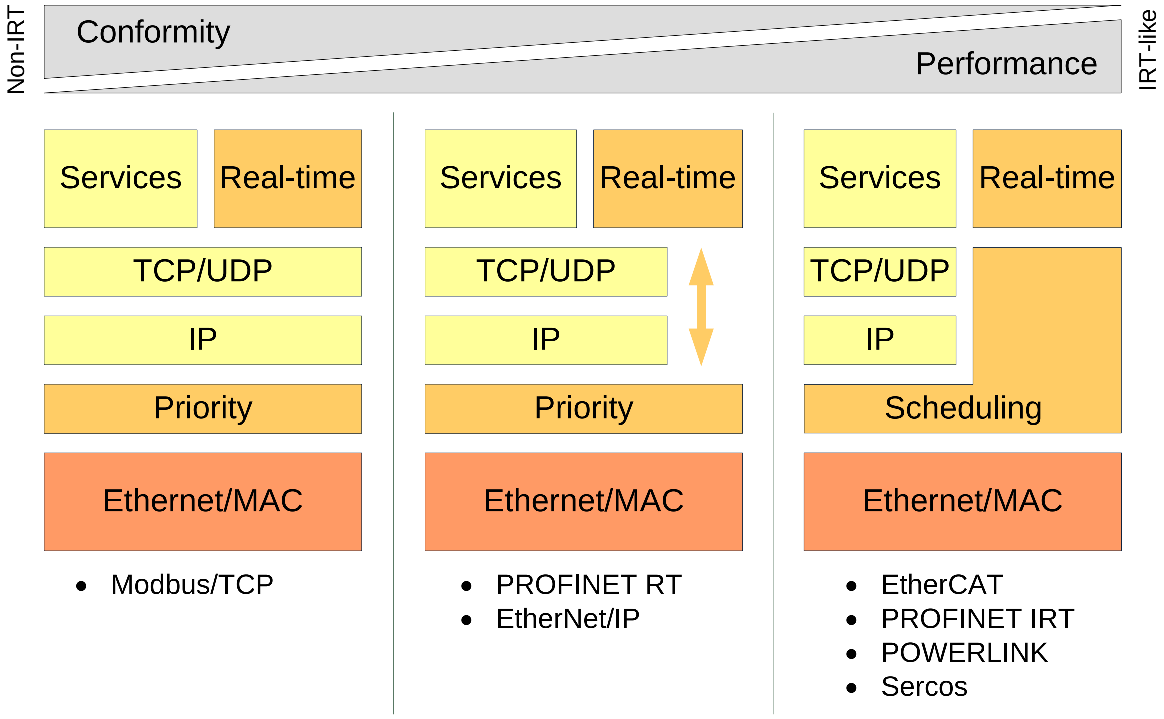


Figure 1: Classification of real-time Ethernet systems and standards [1]

The prime reason why someone prefers one variant over the other differs [2]. In applications that require advanced real-time capabilities EtherCAT is for instance one preferred choice. On the other hand, PROFINET is perceived as seamless migration strategy from Fieldbus to Ethernet because of its scalable nature. Other variants like EtherNet/IP offer effortless interconnections of information technology and industrial automation. Thus, as illustrated in Figure 1, the trade-off of conformity and performance that builds on the foundation of system and application requirements determines the network of choice.

The deployment of real-time Ethernet and preference of individual variants are influenced by altering infrastructures, different stakeholders and yet regulatory initiatives. The more important individual variants become in a region of interest, the more important it is to take part in those standards. But different technologies and manifold standards all lead to fragmented markets with various system-specific requirements. The best approach to keep entry barriers to all these markets low is to benefit from a multiprotocol chip interface at the lowest possible footprint.

1.1 Multiprotocol Capability

The challenge that product managers and application designers of embedded industrial systems face is the trade-off between technical and commercial constraints. The first question that comes in mind is in which product segments and markets to compete with as every real-time Ethernet standard requires its own protocol-specific software stack and some additionally a protocol-specific hardware. Standard devices available on the market frequently lack multiprotocol capabilities and exceed bandwidth when it comes to fast start-up for industrial control.

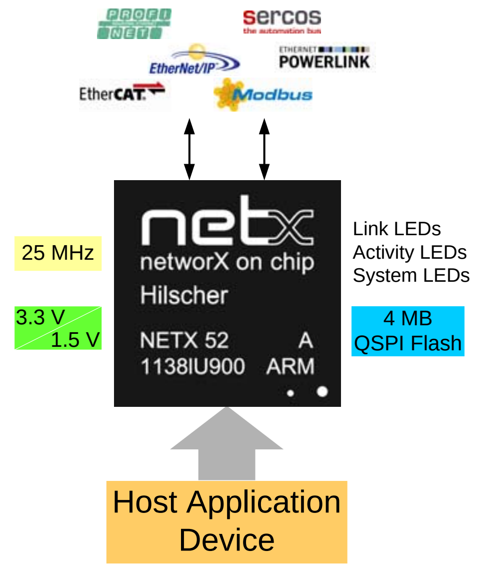
Market pressure to adapt to changing conditions due to various standards and limited resources to update, replace and launch products require a well-balanced design approach to re-use embedded components and software modules that shorten development cycles, improve time to markets and lower design risks. This entails that an embedded multiprotocol chip interface needs to address:

- A low footprint that reduces the number of additional components to a minimum
- A complete set of software protocol stacks for all major real-time Ethernet variants
- A very standardized and flexibly configurable interface to any host application device
- A pre-certified solution that can be easily integrated and leveraged across multiple platforms

The netX 52 from Hilscher has been specially designed to addressing all these needs. The device embodies a multiprotocol real-time Ethernet interface with switch capability and integrated dual-PHY that can be vertically and horizontally integrated across segments and markets for every slave-type of application.

1.2 Ready-to-Use Solution

Hilscher's ready-to-use approach as illustrated in Figure 2 reveals a superior solution in form of a software-defined multiprotocol chip interface as companion chip with host interface. In contrast to traditional switch ICs for industrial Ethernet, the netX 52 with its ARM® core offloads the host application device by fully moving the communication tasks to the multiprotocol chip interface. This reduces the effort of a likely re-certification if the application at the host site has to be changed, updated or enhanced, and it also simplifies the maintenance of software protocol stacks.



Multiprotocol Chip Interface:

- Ready-to-use solution with host interface
- Software protocol stacks as loadable firmware
- Free CIFX API/C-Toolkits for host application device
- Parallel/serial host interface access to dual ported memory

- Compact interface with reduced external components count
- Key components consumes less than 25 x 25 mm^2 of PCB area
- BOM in higher quantities below 16 € per reference design [5.3]

- Software toolkits and evaluation boards with add-on modules
- Complete set of programming options and configuration tools
- Uniform and protocol stack independent CIFX API/C-Toolkits

Figure 2: Software-define multiprotocol chip interface

The following paper commences with a brief overview about the netX 52 network controller, before it then moves on to outline the steps and components involved in bringing the software-defined multiprotocol chip interface alive. The paper finally concludes in summarizing potential ways to get started with, whereas the reference design included and information provided make development efforts considerably low.

2 Design for Flexibility

The netX 52 is built from the ground up for low latency, determinism and highest flexibility for the design of deeply embedded real-time applications. The block diagram in Figure 3 creates insights into how the internal data switch brings together the on-chip ecosystem. The heterogeneous multi-core architecture features an ARM® core, coupled with a peripheral interface controller (xPIC) and a flexible communication (xC) subsystem.

The ARM® core, with its complete set of peripherals and internal RAM banks, is intended for use by the real-time operating system and software protocol stacks. The xPIC, with its deterministic processing and predictable interrupt latency, provides an unmatched flexibility to develop applications for fast IO processing such as required for electronic control and gateway functions. The multiplex matrix (MMIO) offers a highly flexible IO pin mapping for the on-chip peripherals.

netX 52 - Use Case Companion Chip:

- Highest interoperability for embedded hosts
- Compact and small footprint with dual-PHY
- Multiplex matrix for flexible IO pin mapping
- High throughput rate host interface access
- Applicable for all major network protocols
- Offers multiprotocol and switch capability
- Network event and cycle synchronization

The two-channel communication interface, with integrated dual-PHY and IEEE 1588 support, builds up one coherent xC subsystem for the 10/100 Mbps real-time Ethernet ports. The integrated dual-PHY operates in two modes, one for twisted pair (10 Base-T / 100 Base-TX) and one for fiber optic (100 Base-FX).

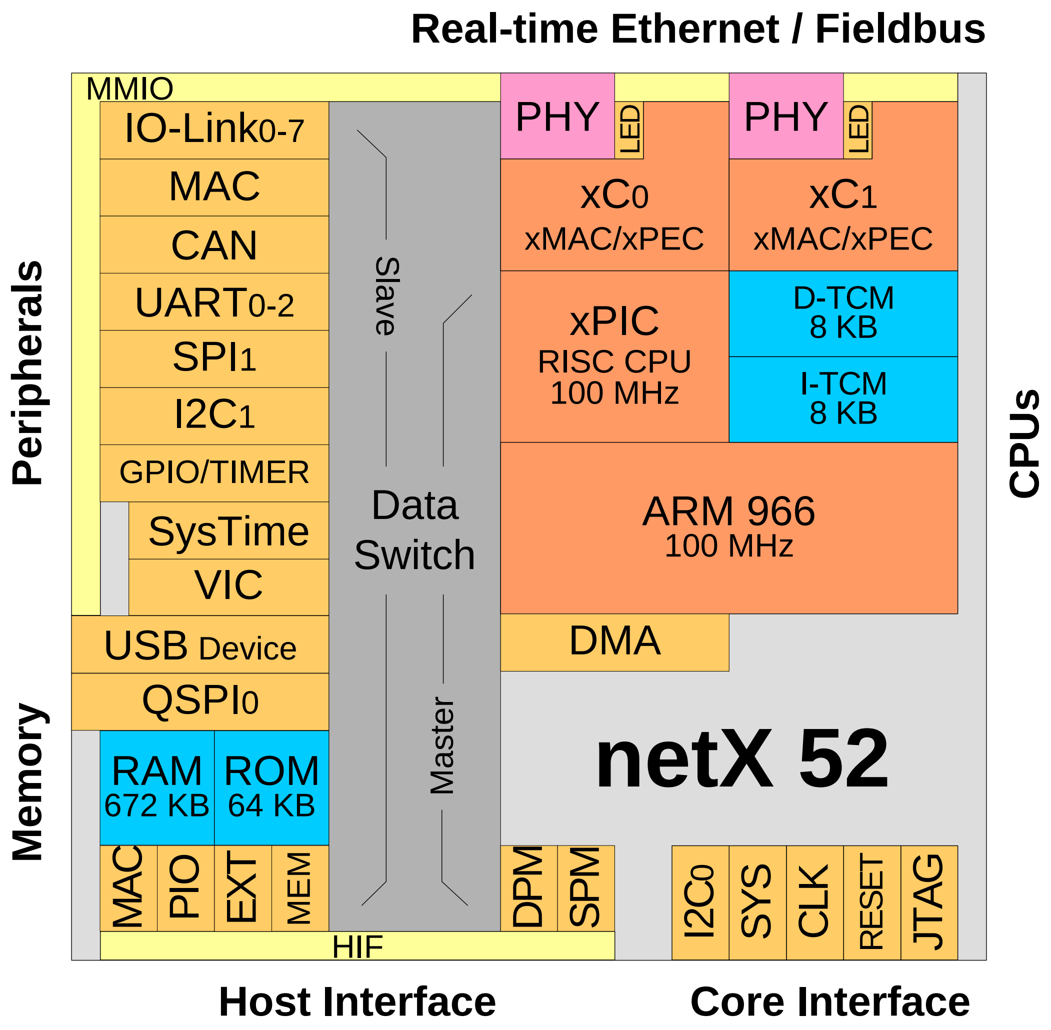


Figure 3: netX 52 functional overview

2.1 xC Technology

The xC subsystem with its inherent flexibility represents a two-channel communication interface with switch capability that is able to realize any protocol-specific hardware for real-time Ethernet. As illustrated in Figure 4, each of the two xC channels consists of two xMAC and two xPEC units specially designed to meeting the requirements of versatile network interfaces.

Due to its event-driven, multi-core and multi-threaded processing capability, the implemented xC technology fulfills the performance requirements of PROFINET V2.3 with fast forwarding, dynamic frame packing and fragmentation for cycle times down to 31.25µs. The technically feasible IO size depends on the network topology and its trade-off between cycle time, payload and number of devices [3].

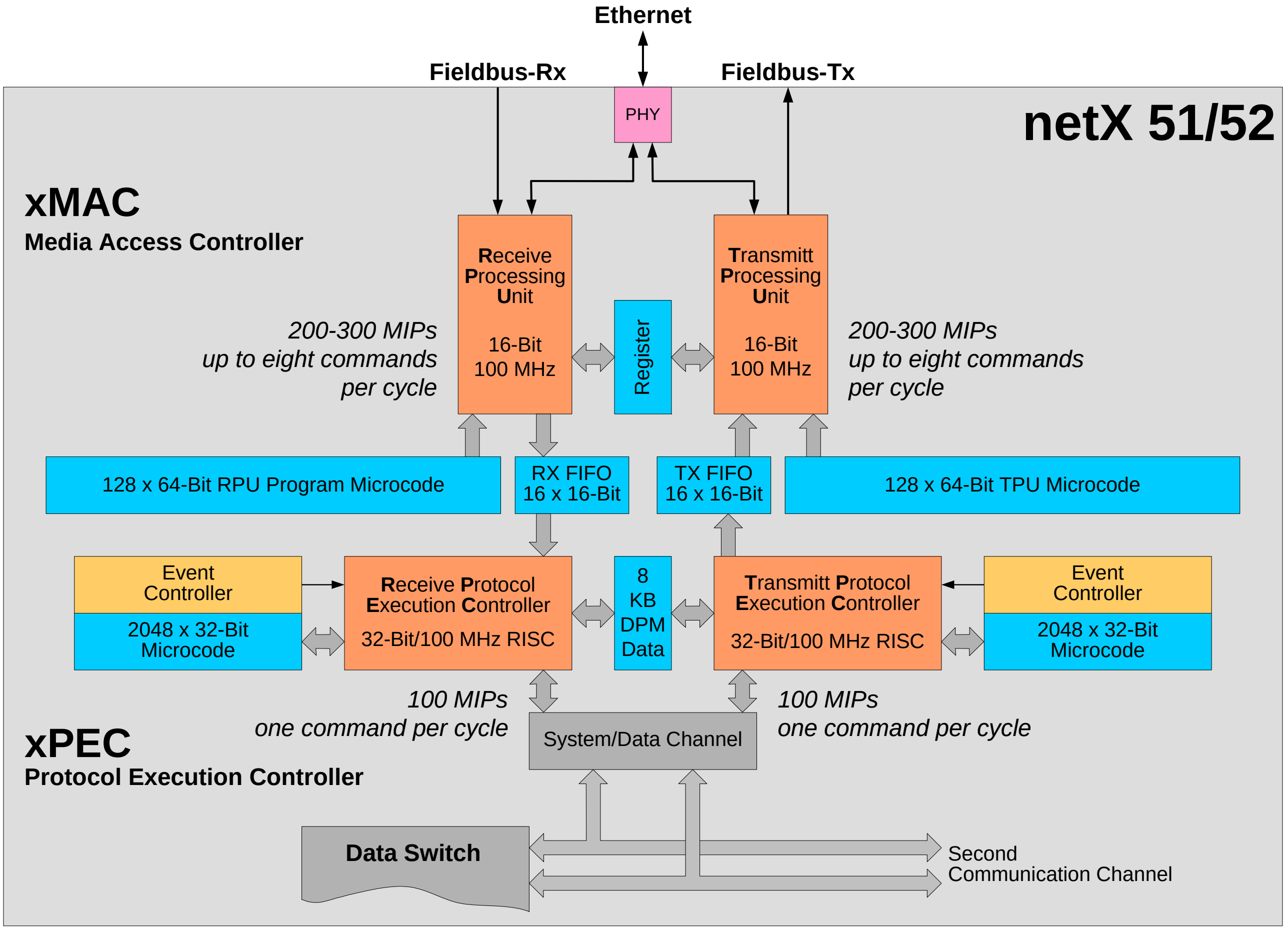

Ethernet
Fieldbus-Rx
Fieldbus-Tx
PHY
netX 51/52
xMAC
Media Access Controller
Receive Processing Unit
16-Bit 100 MHz
Register
Transmitt Processing Unit
16-Bit 100 MHz
200-300 MIPs up to eight commands per cycle
200-300 MIPs up to eight commands per cycle
128 x 64-Bit RPU Program Microcode
RX FIFO 16 x 16-Bit
TX FIFO 16 x 16-Bit
128 x 64-Bit TPU Microcode
Event Controller
2048 x 32-Bit Microcode
Receive Protocol Execution Controller
32-Bit/100 MHz RISC
8 KB DPM Data
Transmitt Protocol Execution Controller
32-Bit/100 MHz RISC
Event Controller
2048 x 32-Bit Microcode
100 MIPs one command per cycle
System/Data Channel
100 MIPs one command per cycle
xPEC
Protocol Execution Controller
Data Switch
Second Communication Channel

Figure 4: xC channel block diagram

During its almost ten years of success story, the xC technology has been successfully gaining traction in markets due to Hilscher's scalable and innovative netX family platform concept.

2.2 Host Interface

The netX seamlessly integrates with any host application device at access level through the high-performance interconnect host interface with configurable interrupt lines. The interoperability is ensured by the provision of selectable host interface modes with access to dual ported memory either through a parallel or serial assembly. In order to support a wide range of MCU, MPU and DSC devices available on the market, the host interface unveils a set of standardized and flexibly configurable peripheral modes as briefly summarized in Table 1.

The DPM interface provides physically a parallel memory bus with data widths of 8/16/32-bit and address lines of 11-20 bits, and is therefore fully functional with any external memory interface and read access times down to 55 ns, whereas parameters and timings are flexibly adaptable to application needs, i.e. handshake, multiplexing, etc.

Host Interface				
Dual-port Memory (DPM) Interface			Serial-port Memory (SPM) Interface	
8-bit	16-bit	32-bit	SPI	QSPI

Table 1: Host interface configuration options

On the contrary, the SPM interface represents a high-speed serial bus either as SPI with up to 125 MHz clock speeds or Quad SPI (QSPI) with up to 33 MHz clock speeds. Figure 5 depicts the protocol sequence that runs on the SPI bus to access the dual ported memory. It typically follows a master-slave relation and precedes either with a write or read command.

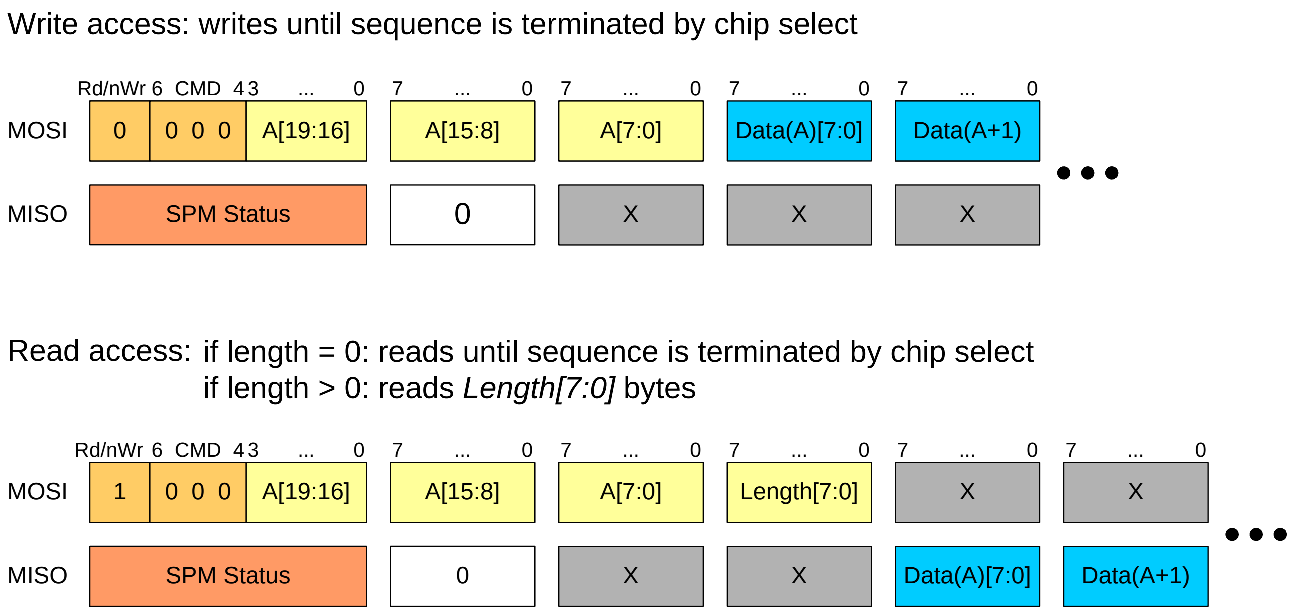


Figure 5: SPM interface with SPI bus protocol sequence

3 Multiprotocol Chip Interface

This chapter draws on the previous sections to outline the steps and components involved in bringing the software-defined multiprotocol chip interface alive. The software architecture in Figure 6 highlights how the different layers get encapsulated to provide a uniform interface to any customer application.

The combination of bootloader and firmware provided as software binary file represents an executable image that is stored in the external flash memory. The built-in ROM code loads the bootloader from external media with the objective to prepare the hardware for the design and to eventually start the firmware from internal RAM. This includes setups and handlings for firmware updates, diagnostics, storage of configuration data and the dual ported memory with access to protocol stack.

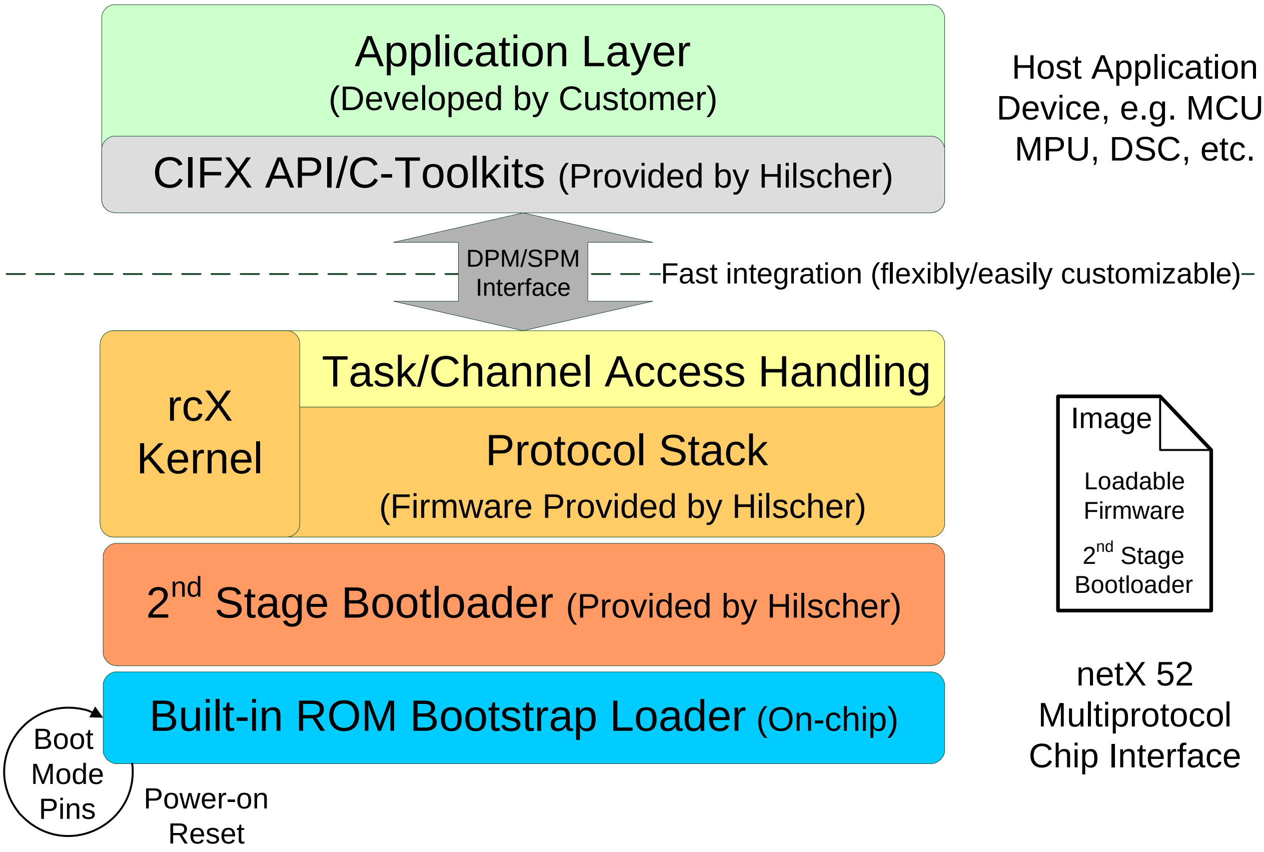

Application Layer
(Developed by Customer)
CIFX API/C-Toolkits (Provided by Hilscher)
Host Application Device, e.g. MCU, MPU, DSC, etc.
DPM/SPM Interface
Fast integration (flexibly/easily customizable)
rcX Kernel
Task/Channel Access Handling
Protocol Stack
(Firmware Provided by Hilscher)
Image
Loadable Firmware
2nd Stage Bootloader
2nd Stage Bootloader (Provided by Hilscher)
netX 52 Multiprotocol Chip Interface
Built-in ROM Bootstrap Loader (On-chip)
Boot Mode Pins
Power-on Reset

Figure 6: Layered software architecture with interface to customer application

The following sections aim to provide an understanding how the software-defined multiprotocol chip interface can be finally integrated into embedded designs. It firstly reviews the hardware components required and then explains with which configuration steps and software toolkits the interface to customer applications can be brought up to initial operation.

3.1 Lowest Possible Footprint

In order to integrate and leverage the multiprotocol chip interface across multiple platforms, a lower footprint count is of utmost importance. The illustration in Figure 7 brings down the number of additional components required for the netX 52 design-in to a minimum. Stabilized voltage rails with monitoring circuitry are normally present in embedded system designs and power-on reset is typically asserted by the host application. The integrated dual-PHY reduces the need of external parts to only a few passive components and one dual-jack with integrated magnetics.

The demand made for QSPI flash memory devices in embedded PC applications has helped to bring its prices down to lower levels. Thus, the netX 52 design is equipped with a QSPI flash based memory that ensures the bandwidth for fast start-up as required by PROFINET IO. The capacity of the external memory typically exceeds the size of the executable image because parts of the flash are being formatted with a file system by the bootloader. The application allocates this extra storage for configuration data, e.g. to identify the device on the network. If the network protocol supports file access, this data will be kept in the local file system with read and write access through the host interface.

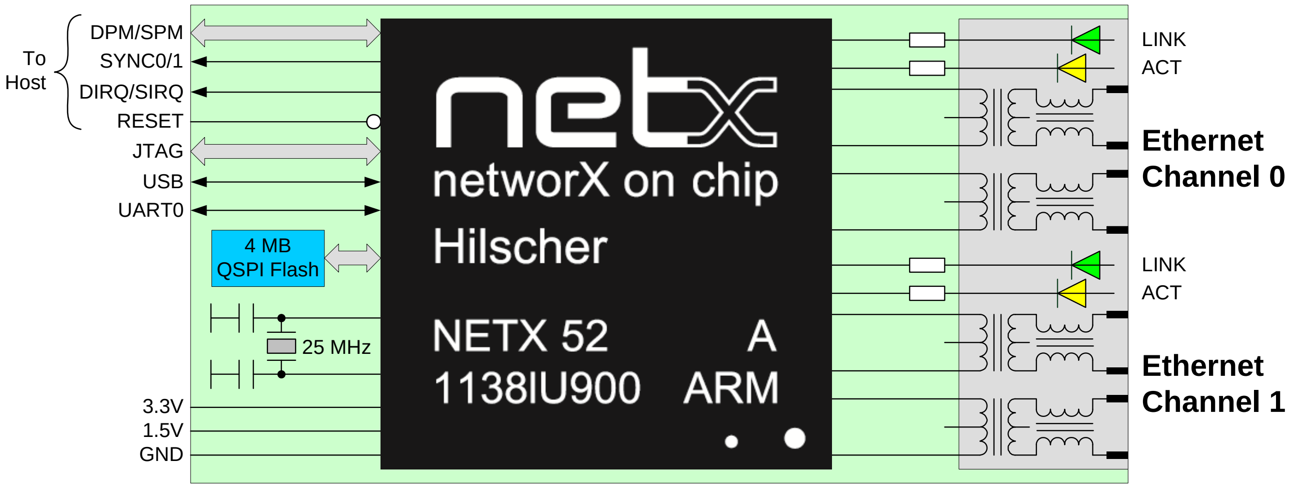


Figure 7: Multiprotocol chip interface embedded components

The host interface builds the common ground to bring the host application device and multiprotocol chip interface together either through a parallel or serial interconnection. For industrial control, the exchange of cyclic IO data is determined by events of the underlying network. Additional hand-shake and synchronization lines ensure that the host keeps track of its cycle time and tasks. Protocol stacks with IRT-like capabilities offer additional trigger signals for advance control tasks, e.g. synchronization, data latch, sample rate, etc.

3.2 Software Configuration Steps

As the software for the multiprotocol chip interface is provided as loadable firmware, which consists of the real-time operating system and protocol stack, designers can purely focus on the application design that initially starts off with setting up the embedded components.

Key interfaces for the embedded application can be customized using the netX TagListEditor tool from Hilscher. Due to the uniform layout with special header tags, the format of the binary file can be simply imported and parameters adapted to application needs. It is thus not necessary to write any line of programming code, rather to configure parameters by a graphical user interface tool.

Settings	Binary File	Software Configuration
netX HIF	2nd Stage Bootloader	Configure DPM/SPM interface for host application device
UART	2nd Stage Bootloader	Enable/disable UART0 interface handling
USB	2nd Stage Bootloader	Enable/disable USB interface handling
MMIO	2nd Stage Bootloader	Change/configure MMIO settings for UART0, xC Trigger0/1 or LEDs
LEDs	Loadable Firmware	Enable/disable link/activity LEDs for dual-port jack and/or protocol
Ethernet IF	Loadable Firmware	Select active ports and enable/disable fiber optic interface *(if supported)*
Product ID	Loadable Firmware	Define vendor ID and/or device ID *(depends on protocol specification)*
Diagnostic	Loadable Firmware	Enable/disable diagnostics using UART0/USB interface *(if supported)*

Table 2: Configurable software settings by the TagListEditor tool

Table 2 lists all settings that can be configured. Especially for the host interface, the tool provides a group of input masks to set individual device registers directly. Additionally, the bootloader offers to autodetect if a DPM or SPM interface is externally present, which can be useful for evaluation purposes. In case of this function is set, both DIRQ and SIRQ are then temporarily used as mode pins during the software bootloader sequence.

3.2.1 Available Protocol Stacks

The generic setup generated by the bootloader makes it simply possible to change the protocol stack the firmware is based on to any other real-time Ethernet standard. Table 3 highlights all protocol stacks available from Hilscher at the time of writing this paper. The license model for netX 52 firmware does not require any per-unit royalty reporting. The purchase of a software license at a competitive price level permits the user to integrate and leverage the firmware as on many platforms as possible.

netX 52 Protocol Stack	Loadable Firmware	Reference
EtherCAT Slave	Available	www.ethercat.org
EtherNet/IP Adapter	Q2 2014 *(end of quarter)*	www.odva.org
Modbus/TCP Client/Server	Available	www.modbus.org
POWERLINK CN	Q1 2015 *(preliminary)*	www.ethernet-powerlink.org
PROFINET IO RT Device	Available	www.profinet.com
PROFINET IO IRT Device	Q3 2014 *(preliminary)*	www.profinet.com
Sercos Slave	Available	www.sercos.de/en

Table 3: Available real-time Ethernet protocol stacks for slave-type of applications

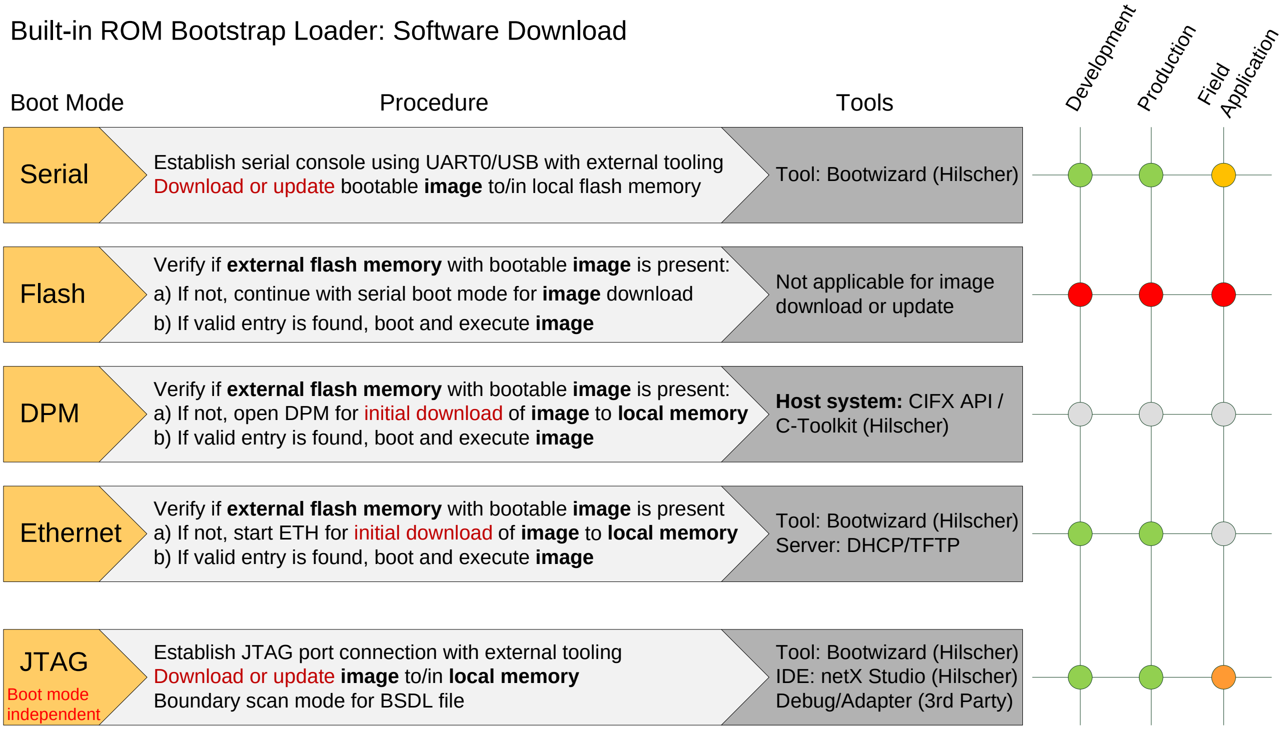
3.2.2 Software Download/Update

The illustration in Figure 8 summarizes the chip boot modes and outlines potential ways for the download of executable images to serial flash memory. The boot mode is determined by mode pins RDY and RUN after power-on reset. By default, it is recommended to connect two dual-LEDs to these pins so that the built-in ROM bootstrap loader can indicate its current state.

One way for the download of executable images is using the Bootwizard tool from Hilscher. This tool grants access to external flash memory connected to netX from any Windows PC by UART, USB, Ethernet or JTAG. The Ethernet boot option is most common for the initial download of images during end of line programming at production facilities, but not limited to, since this interface is always present. Thus, after the initial download, it is not needed to change the boot mode pins by hardware again, for instance as required for IP67 devices with resin coating.

The executable image may consist either of bootloader and firmware or bootloader only. The former one is most appropriate for end of line programming to flash the application at once. If the latter one is the case, the netHOST tool from Hilscher can be used to download the firmware from any Windows PC, independently from the chip boot mode, as the bootloader offers a marshalling functionally for USB and UART access. A brief summary of most used configuration tools is included in Appendix 5.2.

Figure 9 sketches the scenario of software update procedures for field applications, in conjunction with the host application device and access functions to dual ported memory part of the CIFX API/C-Toolkits, e.g. as implemented webserver at the host site for remote access. If the host application device does not provide enough resources for, Hilscher offers a solution based on the netX 51 with external SDRAM that features a loadable firmware with embedded webserver.



Built-in ROM Bootstrap Loader: Software Download

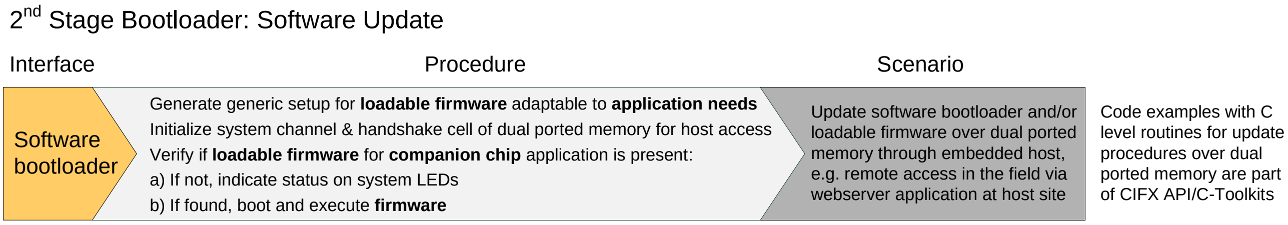
Boot Mode	Procedure	Tools	Development	Production	Field Application
Serial	Establish serial console using UART0/USB with external tooling Download or update bootable **image** to/in local flash memory	Tool: Bootwizard (Hilscher)	🟢	🟢	🟡
Flash	Verify if **external flash memory** with bootable **image** is present: a) If not, continue with serial boot mode for **image** download b) If valid entry is found, boot and execute **image**	Not applicable for image download or update	🔴	🔴	🔴
DPM	Verify if **external flash memory** with bootable **image** is present: a) If not, open DPM for initial download of **image** to **local memory** b) If valid entry is found, boot and execute **image**	**Host system:** CIFX API / C-Toolkit (Hilscher)	⚪	⚪	⚪
Ethernet	Verify if **external flash memory** with bootable **image** is present a) If not, start ETH for initial download of **image** to **local memory** b) If valid entry is found, boot and execute **image**	Tool: Bootwizard (Hilscher) Server: DHCP/TFTP	🟢	🟢	⚪
JTAG Boot mode independent	Establish JTAG port connection with external tooling Download or update **image** to/in **local memory** Boundary scan mode for BSDL file	Tool: Bootwizard (Hilscher) IDE: netX Studio (Hilscher) Debug/Adapter (3rd Party)	🟢	🟢	🟠

Remark:
- **Image:** executable image with defined bootheader
- **External flash memory:** parallel/serial Flash or SD/MMC
- **Local memory:** external Flash/SDRAM or internal RAM
- **Host system:** embedded Processor/Controller, PCI Card, etc.

Legend:

⚪ Requires development effort at host site
🔴 Impractical for image download or update
🟡 Manually possible for individual field devices
🟢 Out-of-the-box solution with dedicated tooling

Figure 8: Chip boot modes and tool options for software download



2nd Stage Bootloader: Software Update

Interface	Procedure	Scenario	
Software bootloader	Generate generic setup for **loadable firmware** adaptable to **application needs** Initialize system channel & handshake cell of dual ported memory for host access Verify if **loadable firmware** for **companion chip** application is present: a) If not, indicate status on system LEDs b) If found, boot and execute **firmware**	Update software bootloader and/or loadable firmware over dual ported memory through embedded host, e.g. remote access in the field via webserver application at host site	Code examples with C level routines for update procedures over dual ported memory are part of CIFX API/C-Toolkits

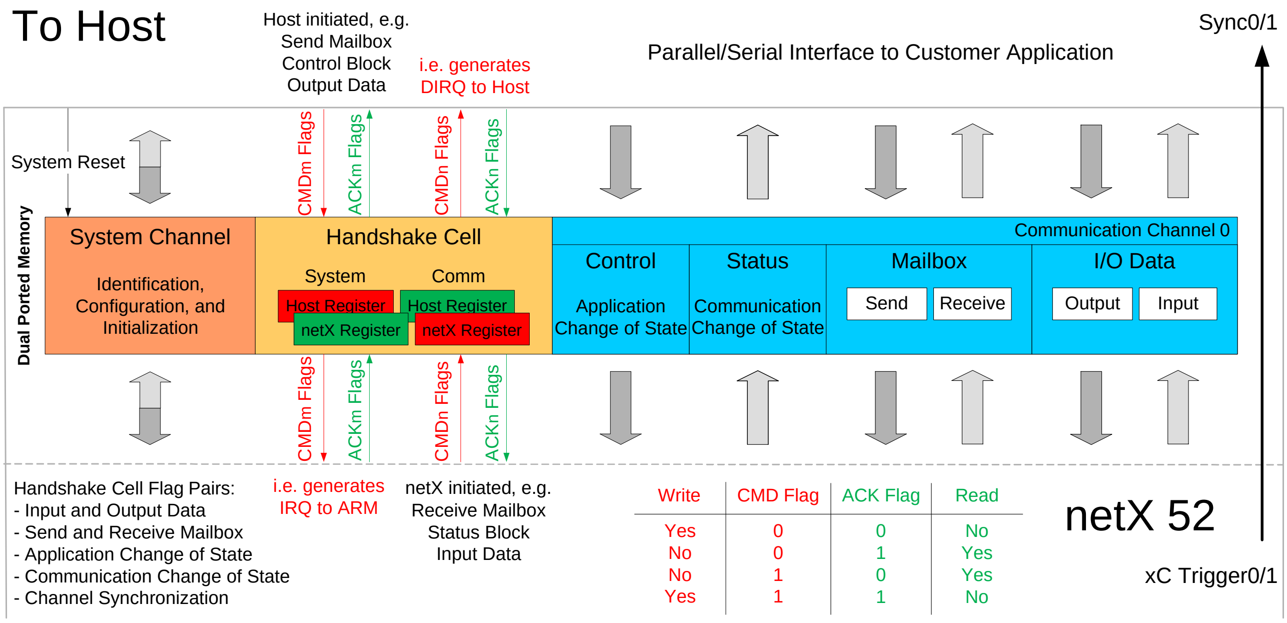
Remark:
- **Loadable firmware:** consists of protocol stack and rcX kernel
- **Companion chip**: loadable firmware without webserver implementation
- **Application needs:** individual parameters adaptable by TagListEditor tool

Figure 9: Software update procedure for field applications

3.3 Interface to Customer Application

This section builds on the previous analysis to introduce the structured layout that is being used for the dual ported memory, coupled with supported hardware and software assisted synchronization mechanisms for the host application device. Figure 10 illustrates the standard layout of the dual ported memory that is accessible from the customer application. In general, the layout starts off with the system channel and handshake cell, initialized by the bootloader after power-up. The system channel intends to provide information and options to identify, configure and initialize the system device. The handshake cell provides a set of registers for every channel with special flag pairs to mutually synchronize changes, updates, transfer cycles, etc.

If the firmware with embedded protocol stack is loaded, the communication channel appears equal to the default layout as shown below. The total layout size including the first two initial blocks is fixed by default to 16 kB. The number of bytes for cyclic data that is transferred at once depends on the application, protocol stack and underlying network in terms of quality of service and hard real-time. A few default parameters that affect the layout of the communication channel can be optionally adapted to individual needs by altering settings kept in the system channel.



Write	CMD Flag	ACK Flag	Read
Yes	0	0	No
No	0	1	Yes
No	1	0	Yes
Yes	1	1	No

Figure 10: Standard dual ported memory software layout

From the host’s point of view, the control block and status block stay in close relations. With the control block the host forces the channel to transition between communication states and with the status block the host keeps track of the actual network state. A featured watchdog handling allows the host and channel to supervise each other.

The configuration service of the channel is realized by using a mailbox mechanism as server/client system. The host sends data packets with identifier down to communication channel and receives in return its response with status information. In fact, if supported by the protocol stack, the host application device is able to open a TCP/IP socket interface based on the use of data packets for remote access. In this case, requests from network are passed on through the receive mailbox.

As illustrated by the bottom table in Figure 10, whenever one side wants to read from or write to dual ported memory, the handshake procedure as indicated by the alternating flag pair ensures a reliable operation. For industrial control, the exchange of cyclic data enforces a high accuracy and deterministic operation aligned to channel and network cycle time. Thus, Table 4 summarizes how the host application can synchronize with the underlying network device.

Host Synchronization	Process Data	Synchronization Mode	
Supported Protocol Types	Non-IRT / IRT-like	IRT-like	IRT-like
Handshake Procedure	Software assisted	Software assisted	Hardware assisted
Handshake Cell Flag Pair	Input Data	Channel Synchronization	n.a.
Hardware Signals netX	DIRQ	DIRQ	xC Trigger0/1
Network-related Jitter	decoupled	< 5 µs (jitter + latency)	<< 1 µs

Table 4: Handshake procedures for host synchronization

The process data handshake, which is pre-configured by a scheduling timer, synchronizes the host every time if a new set of input data can be read from the channel. The host then writes back the output data based on the inputs or any other factors. The synchronization is done by polarity changes of the input data flag pair that generates an IRQ out the host needs to respond to. If the host does not acknowledge, the channel cannot claim back its write access and keeps triple buffering input data. This ensures that the channel retains always the actual input data.

The synchronization based on the process data handshake as described before is most suitable for non-IRT applications. For IRT-like applications, where distributed clock synchronization is used, the synchronization mode is introduced as summarized in Table 4. The software assisted mode offers to use a special channel synchronization flag pair in the handshake cell assigned internally to efficient IRQ handling. On the contrary, the hardware assisted synchronization provides the least jitter as the trigger signals are generated by the xC subsystem directly.

The following use case for EtherCAT emphasizes the difference between both mode options. The software assisted synchronization derives from network events, such as frame reception, for the exchange of cyclic input or output data. The hardware assisted synchronization derives from two xC trigger signals with programmable offsets based on the distributed clock. This allows the host application device to sample input data or latch output data at a defined point in time, which reduces the jitter significantly below << 1 µs.

3.4 Scale Model for Rapid Prototyping

A valued set of complementary software toolkits and evaluation boards with add-on modules offer in turn a scale model for rapid prototyping, driven by the fact that software development costs are high and hardware costs low. This approach is used for a wide range of multiprotocol chip interface designs because embedded software developers can start in early project phases using a PCI card/USB adapter if the custom hardware is not yet ready and later move on for the integration to development toolchains inherent to host application device.

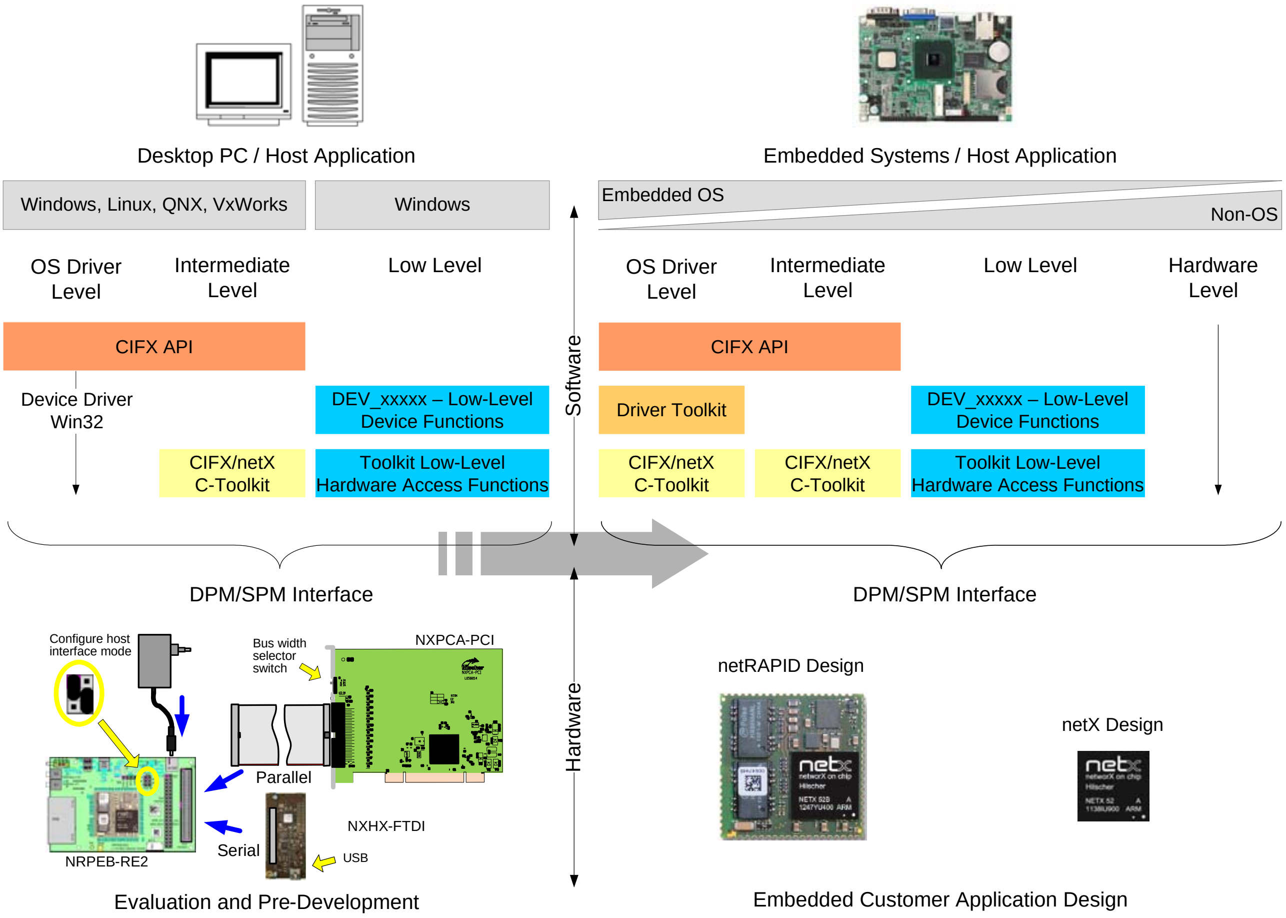

Desktop PC / Host Application
Embedded Systems / Host Application
Windows, Linux, QNX, VxWorks
Windows
Embedded OS
Non-OS
OS Driver Level
Intermediate Level
Low Level
OS Driver Level
Intermediate Level
Low Level
Hardware Level
CIFX API
CIFX API
Device Driver Win32
DEV_xxxxx – Low-Level Device Functions
Driver Toolkit
DEV_xxxxx – Low-Level Device Functions
CIFX/netX C-Toolkit
Toolkit Low-Level Hardware Access Functions
CIFX/netX C-Toolkit
CIFX/netX C-Toolkit
Toolkit Low-Level Hardware Access Functions
Software
Hardware
DPM/SPM Interface
DPM/SPM Interface
Configure host interface mode
Bus width selector switch
NXPCA-PCI
Parallel
Serial
NRPEB-RE2
NXHX-FTDI
USB
netRAPID Design
netX Design
Evaluation and Pre-Development
Embedded Customer Application Design

Figure 11: Scale model for rapid prototyping

Figure 11 outlines the scale of migration for development cycles from left to right and scope of abstraction from hardware up to software. Scale models of software typically refer to user interface, functionality and performance. The uniform API with toolkits denotes a protocol stack independent interface with full access to dual ported memory of netX based devices. The API is designed to give applications control over channel access functions for the underlying system device and its communication network. Therefore, the API has been named CIFX, an acronym for communication interface based on netX.

The potential entry level as outlined depends on the selected host application device and its embedded software needs. The spectrum typically ranges from high-end MPU devices with operating system to low-end MCU devices with restricted memory footprint. Toolkits for the CIFX API are independent from operating systems and available with C source code examples for every use case as shown to develop either a device driver or function library respectively. For MCU applications without operating system, the set of low-level device functions with access to dual ported memory builds another entry point. The offered toolkit delivers the structure and sources required to adapt special hardware access functions to MCU specific needs. For direct hardware level access, a header file with defined structures, flags and bit masks is provided.

In early project phases, a PCI card/USB adapter that hooks up the evaluation board is well suited to start off with. If the firmware is loaded, the protocol stack requires a set of configuration data which is kept in the local flash file system. With the PCI card/USB adapter, the configuration tool SYCON.net from Hilscher can be used to access the communication channel from any Windows PC, by interfacing the dual ported memory of the netX device directly. As a result, it is possible to set up a network device for initial testing without the need of writing any line of programming code. In addition, provided project examples for Visual Studio give insights for the use of channel access functions to initialize and configure network devices on software level. In this case, Visual Studio Express from Microsoft as lightweight and freeware IDE edition serves well for evaluation and pre-development purposes.

4 Conclusion

As illustrated throughout this paper, the ready-to-use approach in form of the software-defined multiprotocol chip interface as companion chip solution reveals a well-balanced design approach to benefit from the use of pre-certified embedded components and software modules that shorten development cycles, improve time to markets and lower design risks.

In conclusion, the embedded multiprotocol chip interface and aligned software enablements deliver at a glance:

- A low footprint that reduces the number of additional components to a minimum
- A complete set of software protocol stacks for all major real-time Ethernet variants
- A very standardized and flexibly configurable interface to any host application device
- A pre-certified solution that can be easily integrated and leveraged across multiple platforms

Figure 12 finally highlights the full range of Hilscher's products and services for slave-type of applications. The netX 52 as introduced with its multiprotocol chip interface capability supports all major real-time Ethernet standards. At the lower end is positioned the netX 10 with one xC channel that commercially addresses traditional fieldbus technologies. The higher end is represented by the netX 51 that offers an external memory interface with a high-performance multi-channel and multi-cache controller and is thus suitable for the development of cost-sensitive stand-alone applications.

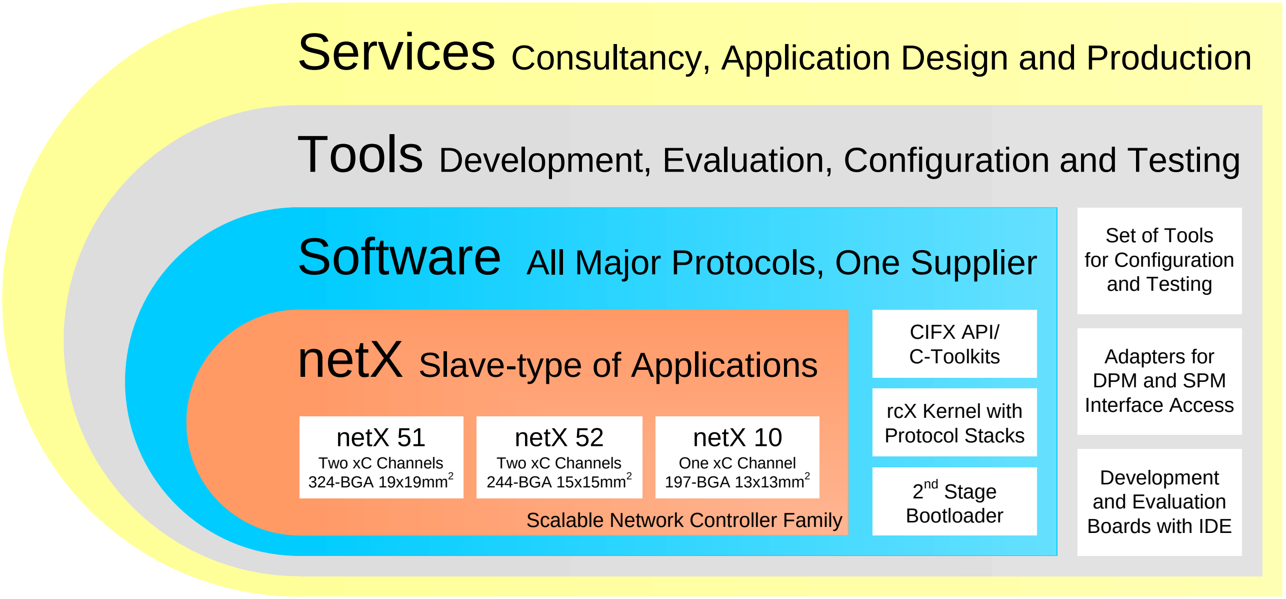

Services Consultancy, Application Design and Production
Tools Development, Evaluation, Configuration and Testing
Software All Major Protocols, One Supplier
netX Slave-type of Applications
netX 51
Two xC Channels
324-BGA 19x19mm²
netX 52
Two xC Channels
244-BGA 15x15mm²
netX 10
One xC Channel
197-BGA 13x13mm²
Scalable Network Controller Family
CIFX API/ C-Toolkits
rcX Kernel with Protocol Stacks
2nd Stage Bootloader
Set of Tools for Configuration and Testing
Adapters for DPM and SPM Interface Access
Development and Evaluation Boards with IDE

Figure 12: Single-source supplier of network controller and protocol stacks for industrial communication

In practice, the best way to get started with is the netRAPID evaluation board, since this product bundle contains three extra netRAPID modules for the build of prototypes and it also includes all configuration tools, software toolkits, project examples and documentations needed. The lineup of available NXHX software development boards with an industry standard Eclipsed based toolchain enables customers to start application designs quickly.

5 Appendix

5.1 Development/Evaluation Boards

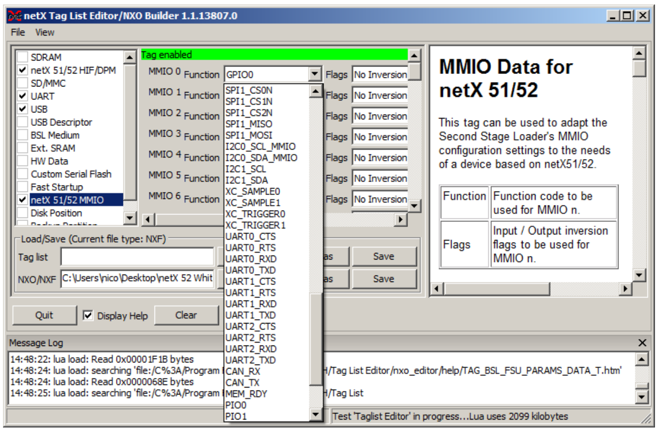
Part Name	Part Description	Part Number
NRPEB-RE2***1)**	▪ netRAPID evaluation board with accessories and product DVD ▪ Protocol stacks as loadable firmware with limited functionality ▪ Three extra netRAPID modules for the build of prototypes	7600.200
NXHX 52-JTAG***1)**	▪ netX 52 software development board with accessories and product DVD ▪ Protocol stacks as loadable firmware/object module with limited functionality ▪ Industry standard Eclipsed based toolchain for ARM9 and xPIC	7773.300
NXPCA-PCI	▪ PCI card adapter for parallel access to dual ported memory of netX devices from any desktop PC	7902.100
CAB-NXPCA-PCI	▪ PCI cable for NXPCA-PCI to interconnect development/evaluation boards	4400.000
NXHX-FTDI	▪ USB adapter with FDTI chip for serial access to dual ported memory of netX devices from any desktop PC/Laptop	7703.050 (available in Q3 2014)

Note 1: Listed product bundle includes all configuration tools, software toolkits and documentations needed to get started with.

5.2 Configuration Tools

Tool List	Tool Description
Bootwizard	▪ Used to generate/download executable images for/to external flash memory ▪ Access from any Windows PC by UART, USB, Ethernet or JTAG (FTDI USB2JTAG)
TagListEditor	▪ Used to configure firmware and/or bootloader for applications (e.g. DPM/SPM interface, etc.) ▪ Used to load and edit tag lists in header of binary images (Hilscher file extension NXF/NXO)
netHOST	▪ Used to copy network configuration files to file system if software bootloader is present ▪ Used to download firmware to external flash memory if software bootloader is present ▪ Access from any Windows PC by UART and USB (if no loadable firmware is present)
SYCON.net	▪ Used to configure network devices in a generic way (FDT/DTM based technology) ▪ Requires adapter for DPM/SPM access from any PC (e.g. to development/evaluation board) ▪ Generate and/or export configuration files for netX network devices (e.g. for netHOST)

Screenshot: TagListEditor (left) and Bootwizard (right)



□ SDRAM
☑ netX 51/52 HIF/DPM
□ SD/MMC
☑ UART
☑ USB
□ USB Descriptor
□ BSL Medium
□ Ext. SRAM
□ HW Data
□ Custom Serial Flash
□ Fast Startup
☑ netX 51/52 MMIO
□ Disk Position

Tag enabled

MMIO 0 Function GPIO0 Flags No Inversion
MMIO 1 Function Flags No Inversion
MMIO 2 Function Flags No Inversion
MMIO 3 Function Flags No Inversion
MMIO 4 Function Flags No Inversion
MMIO 5 Function Flags No Inversion
MMIO 6 Function Flags No Inversion

SPI1_CS0N
SPI1_CS1N
SPI1_CS2N
SPI1_MISO
SPI1_MOSI
I2C0_SCL_MMIO
I2C0_SDA_MMIO
I2C1_SCL
I2C1_SDA
XC_SAMPLE0
XC_SAMPLE1
XC_TRIGGER0
XC_TRIGGER1
UART0_CTS
UART0_RTS
UART0_RXD
UART0_TXD
UART1_CTS
UART1_RTS
UART1_RXD
UART1_TXD
UART2_CTS
UART2_RTS
UART2_RXD
UART2_TXD
CAN_RX
CAN_TX
MEM_RDY
PIO0
PIO1

Load/Save (Current file type: NXF)
Tag list
NXO/NXF C:\Users\nico\Desktop\netX 52 Whit

☑ Display Help

MMIO Data for netX 51/52

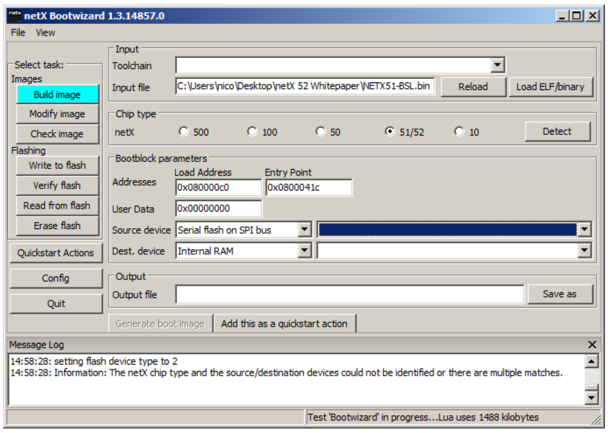
This tag can be used to adapt the Second Stage Loader's MMIO configuration settings to the needs of a device based on netX51/52.

Function	Function code to be used for MMIO n.
Flags	Input / Output inversion flags to be used for MMIO n.

Message Log

```
14:48:22: lua load: Read 0x00001F1B bytes
14:48:24: lua load: searching 'file:/C%3A/Program I /Tag List Editor/nxo_editor/help/TAG_BSL_FSU_PARAMS_DATA_T.htm'
14:48:24: lua load: Read 0x0000068E bytes
14:48:25: lua load: searching 'file:/C%3A/Program I /Tag List
```

Test 'Taglist Editor' in progress...Lua uses 2099 kilobytes



Input
Toolchain
Input file C:\Users\nico\Desktop\netX 52 Whitepaper\NETX51-BSL.bin

Chip type
netX ○ 500 ○ 100 ○ 50 ● 51/52 ○ 10

Bootblock parameters
Addresses Load Address 0x080000c0 Entry Point 0x0800041c
User Data 0x00000000
Source device Serial flash on SPI bus
Dest. device Internal RAM

Output
Output file

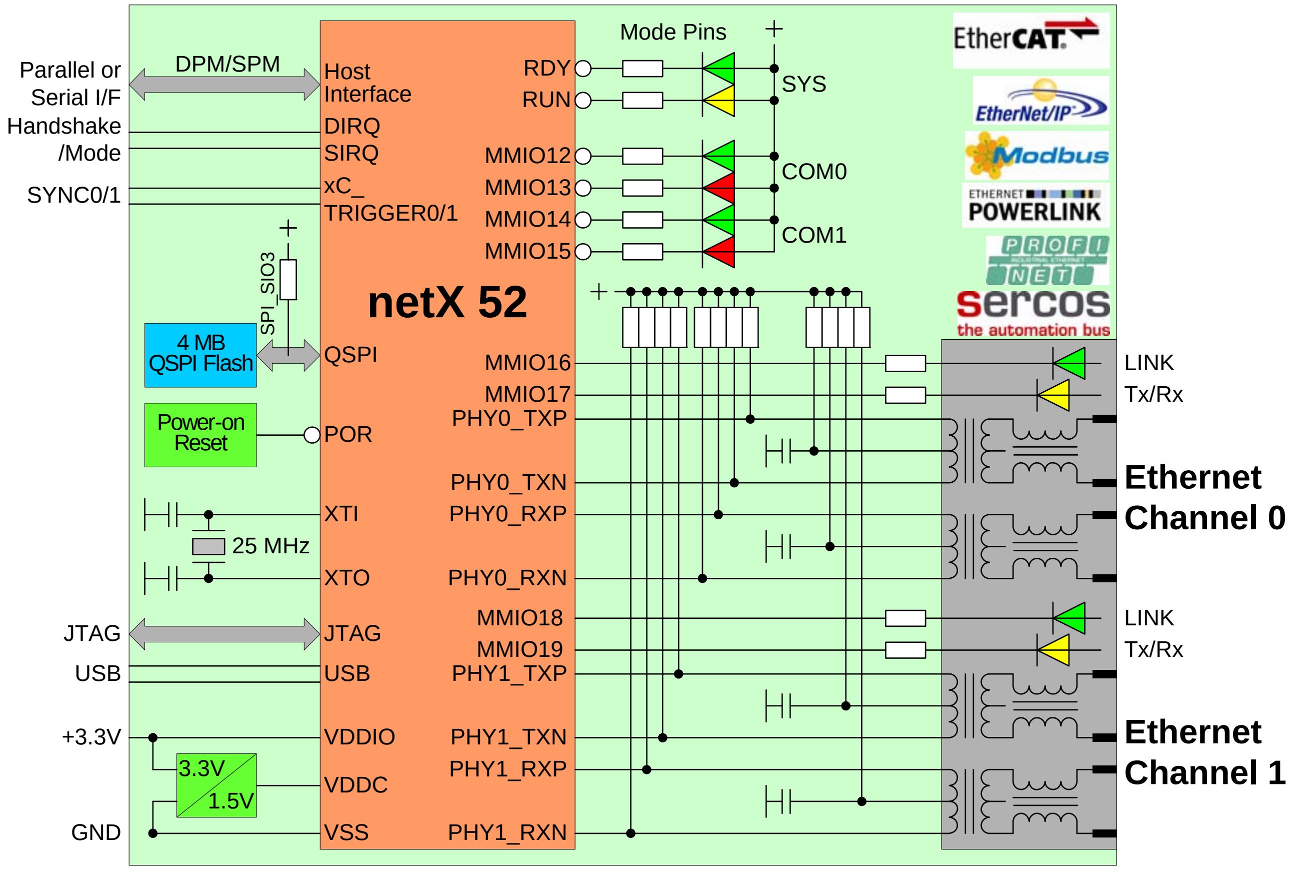
Message Log

```
14:58:28: setting flash device type to 2
14:58:28: Information: The netX chip type and the source/destination devices could not be identified or there are multiple matches.
```

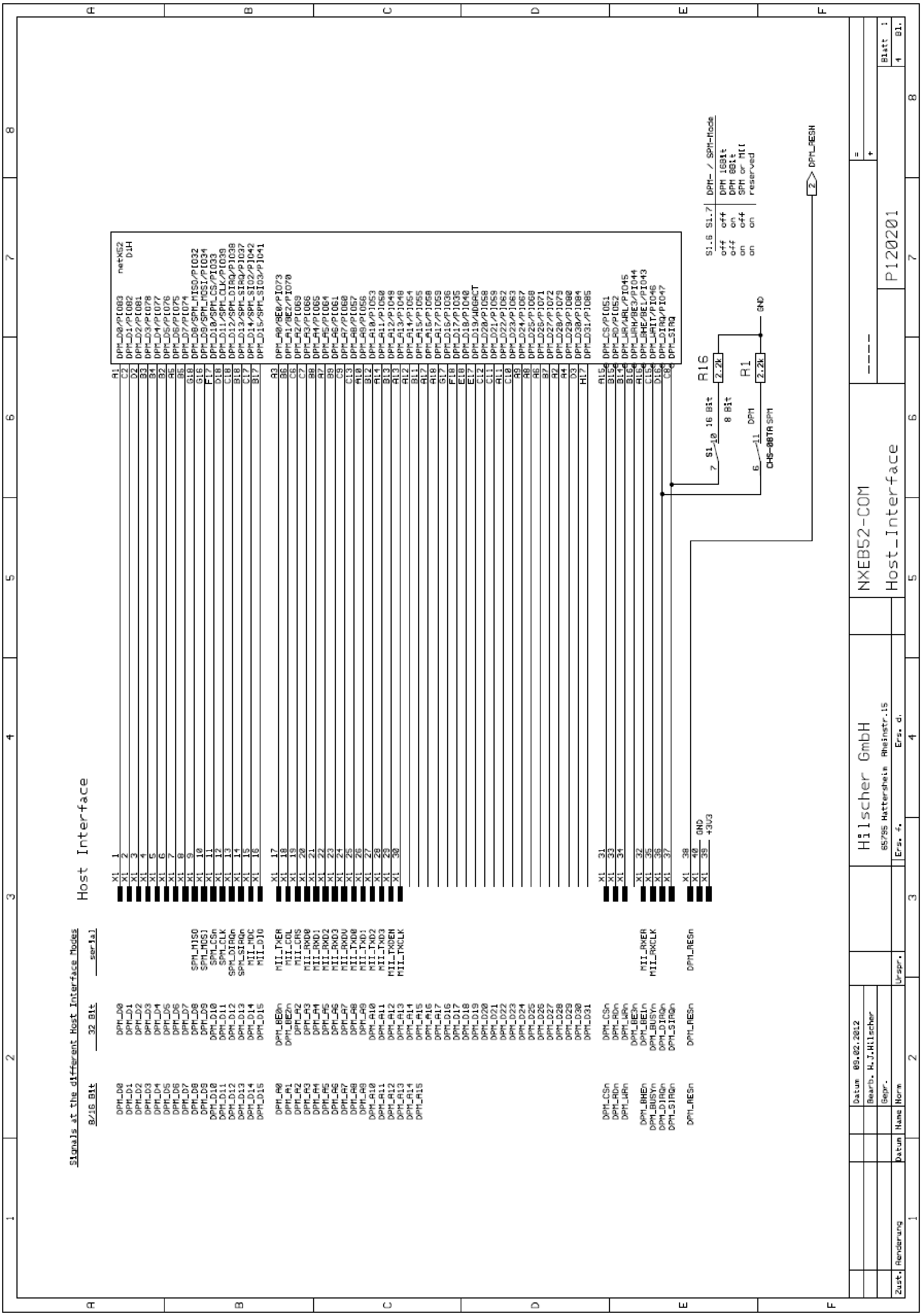
Test 'Bootwizard' in progress...Lua uses 1488 kilobytes

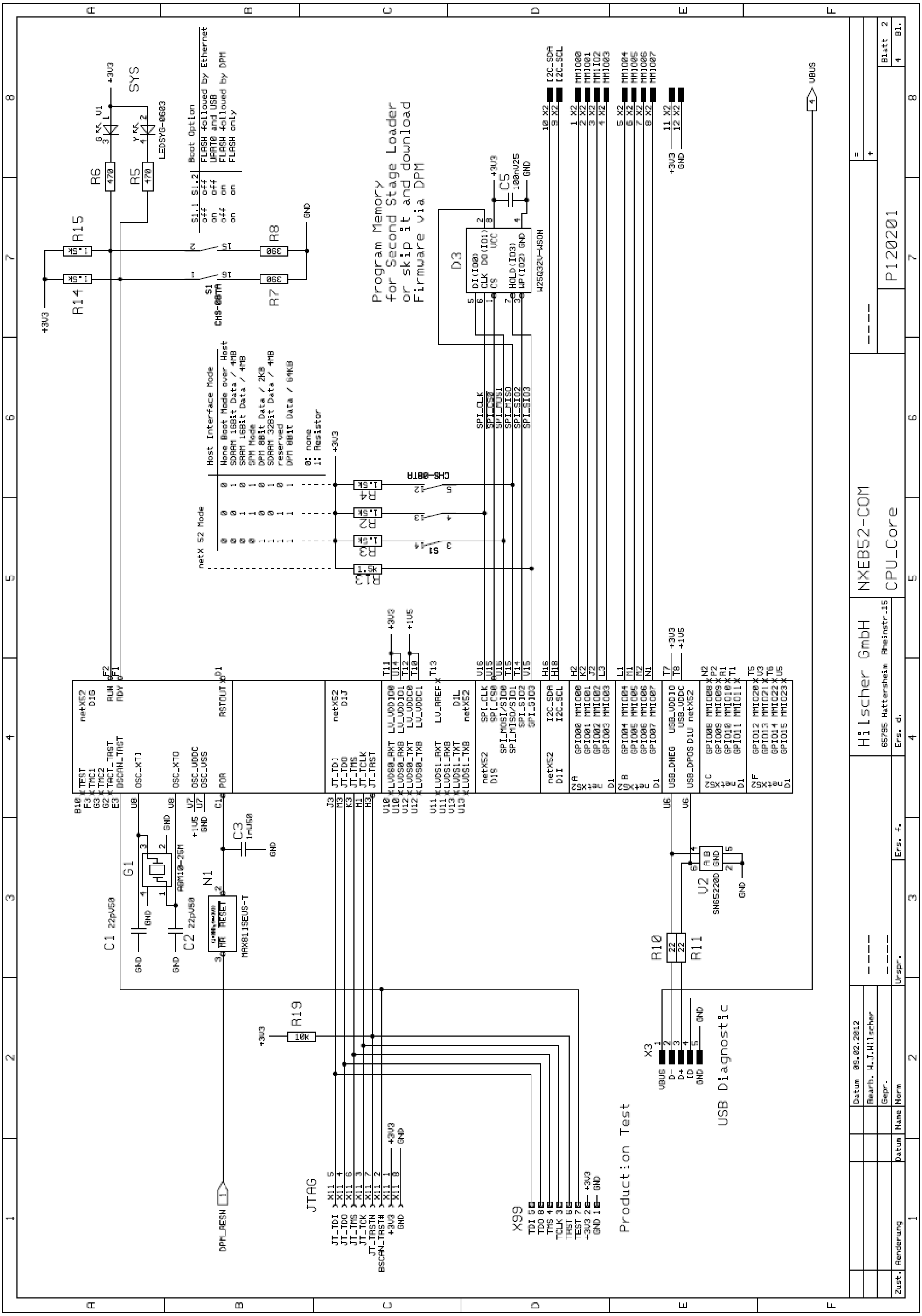
5.3 Reference Design

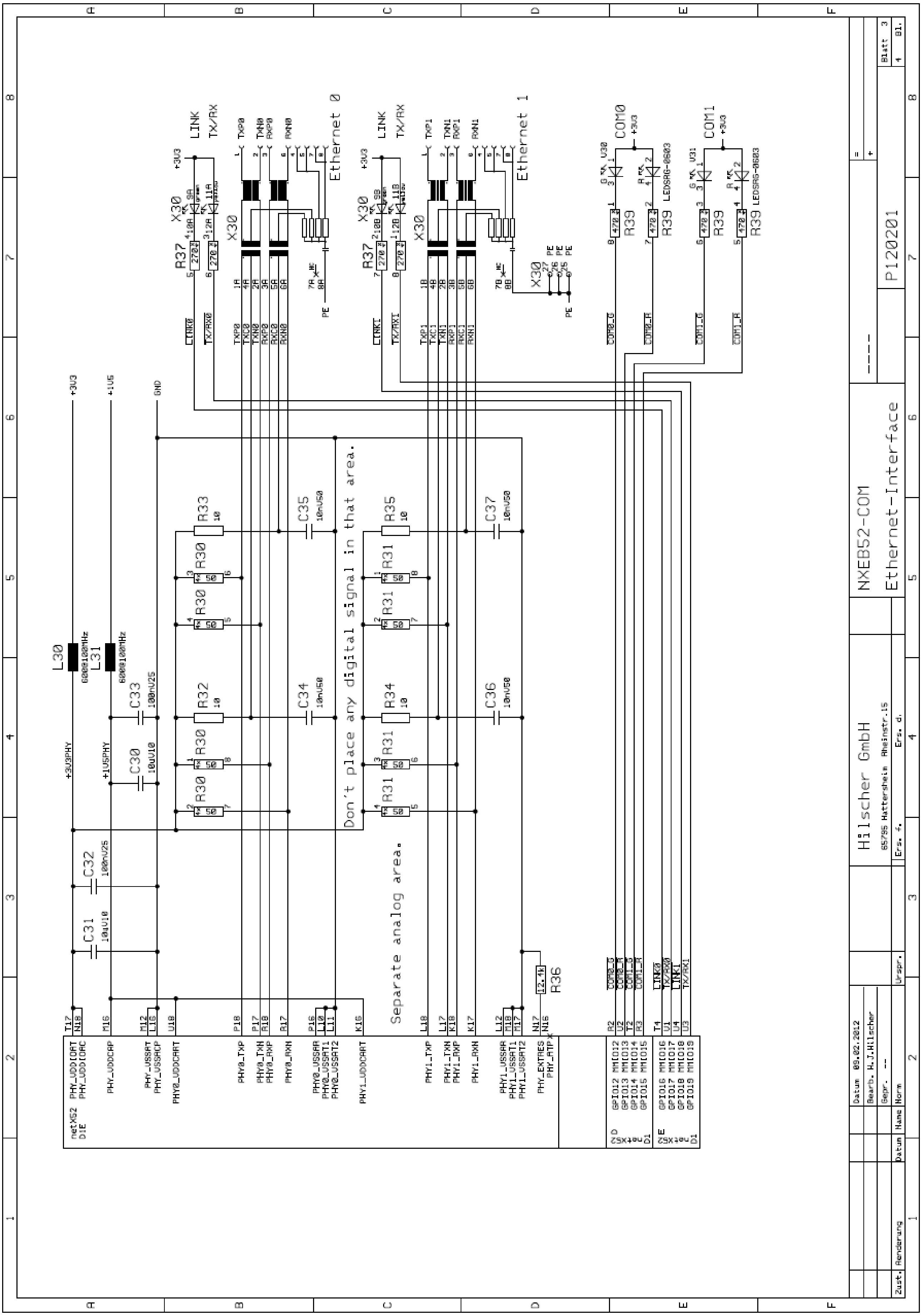
5.3.1 Example NXEB52-COM

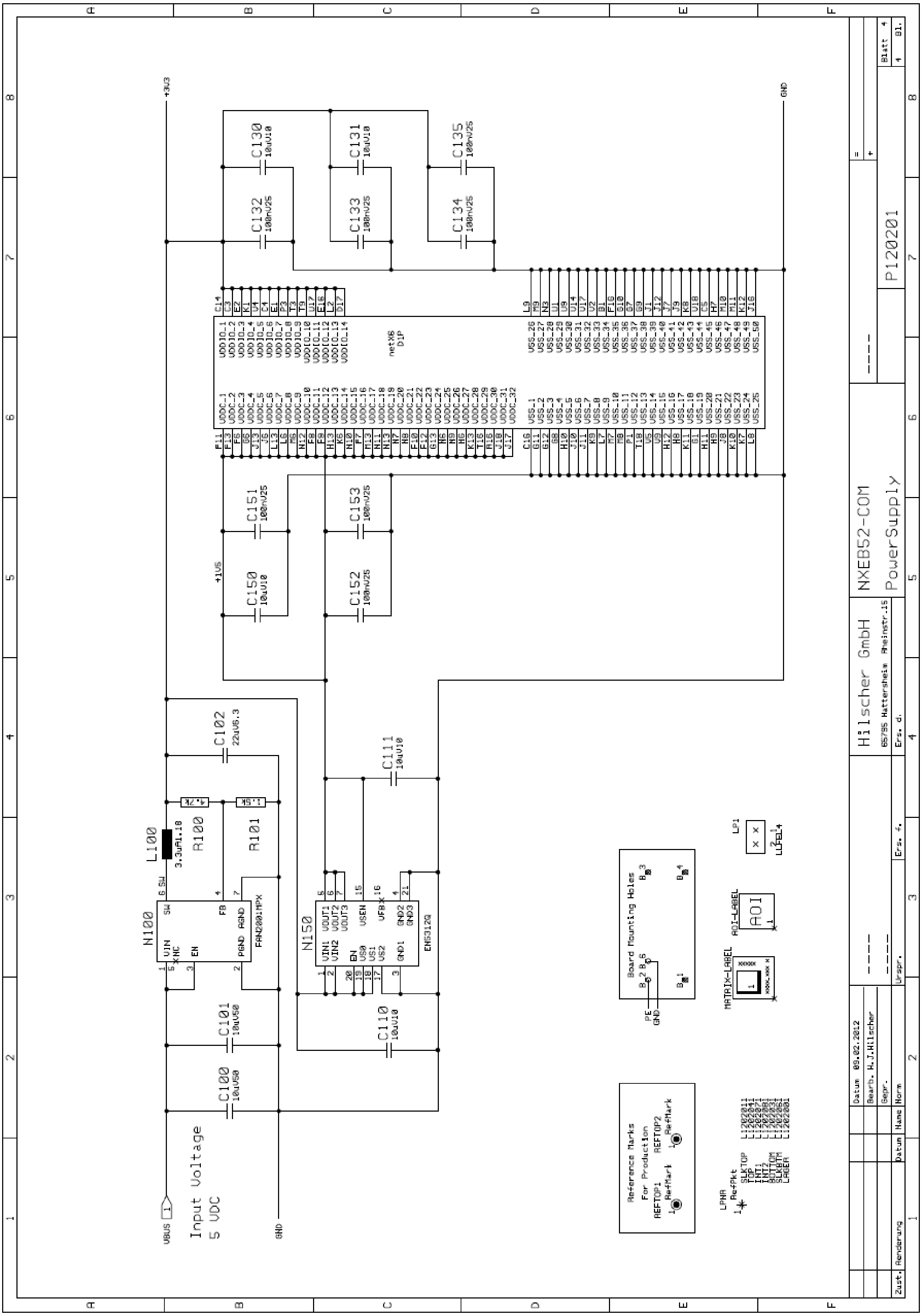


5.3.2 Schematic NXEB52-COM




Hilscher GmbH
65795 Hattersheim Rheinstr.15
NXEB52-COM
CPU_Core
P120201
Datum 09.02.2012
Bearb. H.J.Hilscher
Program Memory for Second Stage Loader or skip it and download Firmware via DPM
D3
W25Q32V-WSON
Boot Option
S1.1 S1.2
off off FLASH followed by Ethernet
on off UART0 and USB
off on FLASH followed by DPM
on on FLASH only
Host Interface Mode
netX 52 Mode
None Boot Mode over Host
SDRAM 16Bit Data / 4MB
SRAM 16Bit Data / 4MB
SPM Mode
DPM 8Bit Data / 2KB
SDRAM 32Bit Data / 4MB
reserved
DPM 8Bit Data / 64KB
0: none
1: Resistor
S1
CHS-08TA
R14
R15
R8
R7
R6
R5
R4
R2
R3
R13
SYS
LEDSYG-0603
V1
netX52
D1G
D1J
D1S
D1I
D1L
G1
ABM10-25M
C1 22pV50
C2 22pV50
C3 1nV50
C5 100nV25
N1
MAX811SEUS-T
R19 10k
JTAG
X99
Production Test
X3
USB Diagnostic
U2
SN65220D
R10
R11
VBUS
DPM_RESN
+3V3
+1V5
GND


Ethernet 0
Ethernet 1
LINK
TX/RX
COM0
COM1
+3V3
+1V5
GND
X30
R30
R31
R32 10
R33 10
R34 10
R35 10
R36 12.4k
R37 270
R39 470
C30 10uV10
C31 10uV10
C32 100nV25
C33 100nV25
C34 10nV50
C35 10nV50
C36 10nV50
C37 10nV50
L30 600@100MHz
L31 600@100MHz
V30
V31
LEDSRG-0603
PE
LINK0
TX/RX0
LINK1
TX/RX1
TXP0
TXN0
RXP0
RXN0
TXP1
TXN1
RXP1
RXN1
COM0_G
COM0_R
COM1_G
COM1_R
+3V3PHY
+1V5PHY
Don't place any digital signal in that area.
Separate analog area.
netX52 DIE
PHY_VDDIOAT
PHY_VDDIOAC
PHY_VDDCAP
PHY_VSSAT
PHY_VSSACP
PHY0_VDDCART
PHY0_TXP
PHY0_TXN
PHY0_RXP
PHY0_RXN
PHY0_VSSAR
PHY0_VSSAT1
PHY0_VSSAT2
PHY1_VDDCART
PHY1_TXP
PHY1_TXN
PHY1_RXP
PHY1_RXN
PHY1_VSSAR
PHY1_VSSAT1
PHY1_VSSAT2
PHY_EXTRES
PHY_ATP
GPIO12 MMIO12
GPIO13 MMIO13
GPIO14 MMIO14
GPIO15 MMIO15
GPIO16 MMIO16
GPIO17 MMIO17
GPIO18 MMIO18
GPIO19 MMIO19
Hilscher GmbH
65795 Hattersheim Rheinstr.15
NXEB52-COM
Ethernet-Interface
P120201
Datum 09.02.2012
Bearb. H.J.Hilscher
Gepr. --
Norm
Blatt 3
4 Bl.
Zust.
Aenderung
Datum
Name
Urspr.
Ers. f.
Ers. d.


+3U3
GND
C130 10uU10
C131 10uU10
C135 100nU25
C132 100nU25
C133 100nU25
C134 100nU25
netX6 D1P
VDDIO_1 … VDDIO_14
VDDC_1 … VDDC_32
VSS_1 … VSS_50
C151 100nU25
C153 100nU25
C150 10uU10
C152 100nU25
+1U5
C102 22uU6.3
C111 10uU10
L100 3.3uH1.18
R100 4.7K
R101 1.5K
N100 FAN2001MPX
VIN NC EN SW FB PGND AGND
N150 EN5312Q
VIN1 VIN2 EN VS0 VS1 VS2 GND1 VOUT1 VOUT2 VOUT3 VSEN VFB GND2 GND3
C110 10uU10
C101 10uU50
C100 10uU50
VBUS
Input Voltage
5 VDC
GND
Board Mounting Holes
PE
GND
Reference Marks For Production
REFTOP1 RefMark
REFTOP2 RefMark
LPNR RefPkt
SLKTOP TOP INT1 INT2 BOTTOM SLKBTM LAGER
L1202011 L1202041 L1202071 L1202031 L1202051 L1202061 L1202081
MATRIX-LABEL
AOI-LABEL AOI
LP1 LLFEL4
P120201
Hilscher GmbH
65795 Hattersheim Rheinstr.15
NXEB52-COM
PowerSupply
Datum 09.02.2012
Bearb. H.J.Hilscher
Gepr.
Norm
Zust. Aenderung Datum Name
Urspr. Ers. f. Ers. d.
Blatt 4 4 Bl.
A B C D E F
1 2 3 4 5 6 7 8

5.3.3 BOM List NXEB52-COM

Part Number	Part Description	Quantity	Symbol	Reference	Supplier
2.232.000	netX 52 Network Controller	1	D1	netx52 UPD800515F1-211-FND-SSA-A	Hilscher
LSFS3.3UA1.18	Inductor SMD	1	L100	273189059032 CR32NP-3R3MC	Sumida
RKRS1.5KW63M-0402	Resistor SMD0402	7	R2 R3 R4 R13 R14 R15 R101	32.19.35 1K5, 1%, 63mW, SMD0402	Samsung
RKRS10W63M-0402	Resistor SMD0402	4	R32 R33 R34 R35	32.39.67 10R, 1%, 63mW, SMD0402	Samsung
RKRS12.4KW63M-0402	Resistor SMD0402	1	R36	35.84.69 12K4, 1%, 63mW, SMD0402	Samsung
RKRS22W63M-0402	Resistor SMD0402	2	R10 R11	32.40.01 22R, 1%, 63mW, SMD0402	Samsung
RKRS470W63M-0402	Resistor SMD0402	2	R5 R6	076327 470R, 1%, 63mW, SMD0402	Yageo
RKRS390W63M-0402	Resistor SMD0402	2	R7 R8	32.19.55 390R, 1%, 63mW, SMD0402	Samsung
RKRS2.2KW63M-0402	Resistor SMD0402	2	R1 R16	32.19.48 2.2K, 1%, 63mW SMD0402	Samsung
RKRS4.7KW63M-0402	Resistor SMD0402	1	R100	32.19.68 4K7, 63mW, 1%, SMD0402	Samsung
RKRS10KW63M-0402	Resistor SMD0402	1	R19	32.19.22 10K0, 1%, 63mW, SMD0402	Samsung
CKSS100NV16-0402	Ceramic Capacitor SMD0402	10	C5 C32 C33 C132 C133 C134 C135 C151 C152 C153	33.88.34 100nF 16V SMD 0402 X7R 10%	Murata Elektronik
CKSS10NV50-0402	Ceramic Capacitor SMD0402	4	C34 C35 C36 C37	36.62.10 10nF 50V SMD0402, X7R	Samsung
CKSS1NV50-0402	Ceramic Capacitor SMD0402	1	C3	35.00.48 1nF 50V SMD0402 5%, C0G	Murata Elektronik
CKSS22PV50-0402	Ceramic Capacitor SMD0402	2	C1 C2	33.47.55 22pF 50V SMD 0402 COG 10%	Murata Elektronik
FAN2001MPX	Step Down DC/DC Conv. 0,5...5,5V	1	N100	FAN2001MPX_NL	Fairchild
CKSS10UV10-0805	Ceramic Capacitor SMD0805	7	C30 C31 C110 C111 C130	35.70.05 10uF 10V SMD0805 X5R	Samsung

			C131 C150	15%	
CKSS22UV6.3-0805	Ceramic Capacitor SMD0805	1	C102	0805W226M6R3NT 22uF 6.3V X5R SMD0805	Novacap
CKSS10UV50-1210	Ceramic Capacitor SMD1210	2	C100 C101	UMK325C7106KM-L 10uF 50V X7S SMD1210	Taiyo Yuden
LEFS600RA1-1206	Ferrit Core SMD1206	2	L30 L31	742 792 18R 600 Ohm, 1A, SMD1206, RoHS	Würth
RSES270X4-1206	Resistor Network SMD1206	1	R37	39.06.11 4*270 Ohm, SMD1206 5% Konkav	NIC Components Europe
RSES50X4-1206	Resistor Network SMD1206	2	R30 R31	CAY16-51R0F4LF YC16-4 SMD 1% 51R	Bourns
RSES470X4-1206	Resistor Network SMD1206	1	R39	80654 YC16-4 SMD 5% 470R	Yageo
ERNI-203313	Dual-RJ45 Jack with Magnetics	1	X30	TRJ26204BNL magnetic module, LED gr/y	Trxcom Technology
XUSBS5BW-MINI	USB Mini-B Receptacle	1	X3	USB MINI-B SMT MOLEX 67503-1020	Molex
SN65220DBVT	Transient Suppressor USB-Port	1	V2	229011 SN65220DBVT	Texas Instruments
LEDSRG-0603	LED Red/Green	2	V30 V31	HSMF-C165	Avago
W25Q32VZPIG	Quad Flash Serial 32MBit	1	D3	W25Q32FVZPIG-T+R W25Q32FVZPIG-T+R	Winbond
CHS-08	Slide Switch 8pins	1	S1	CHS-08MA1	Copal
EN5312Q	DC/DC Converter	1	N150	EN5312QI-T ENPEN5312QIT	Enpirion
LEDSYG-0603	LED Yellow/Green	1	V1	629885 HSMF-C166	Avago
XWLZ40SG	Multi-pin Connector 2x20pins	1	X1	102.126.040.26	FJH
MAX811SEUS-T	Voltage Monitoring	1	N1	MAX811SEUS+T MAX811SEUS+T	Maxim
XWLE12SG	Multi-pin Connector 1x12pins	1	X2	101.126.012.26 12.6mm	FJH

ABM10-25M	Crystal 25MHz SMD	1	G1	228854 Q25,0-JXS22-12-10/20-T1-FU-LF	Jauch
08FLZ-RSM1-TB	1. FFC/FPC Connector	1	X11	08FLZ-RSM2-TB(LF)(SN)	JST

5.4 List of References

[1] Berhard, H. and Mottok, J. (2012). ‘Real-time behavior of Ethernet on the example of PROFINET’, HS-Regensburg, [Internet]. Available from: http://www.hs-regensburg.de/fileadmin/media/fakultaeten/ei/forschung_projekte/MAPR_Ver%C3%B6ffentlichungen/ARC_Heitzer.pdf

[2] Rothhöft, M., (2013). ‘Marktstudie Industrielle Kommunikation: Feldbus – Ethernet - Wireless’, Marktstudien der Automatisierungstechnik: http://www.marktstudien.org

[3] Prytz, G., (2008). ‘A performance analysis of EtherCAT and PROFINET IRT’, IEEE, [Internet]. Available from: http://www.ethercat.org/pdf/english/ETFA_2008_EtherCAT_vs_PROFINET_IRT.pdf

5.5 Contacts

Headquarters

Germany
Hilscher Gesellschaft für
Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com

Support
Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

China
Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-Mail: info@hilscher.cn

Support
Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

France
Hilscher France S.a.r.l.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr

Support
Phone: +33 (0) 4 72 37 98 40
E-Mail: fr.support@hilscher.com

India
Hilscher India Pvt. Ltd.
New Delhi - 110 065
Phone: +91 11 26915430
E-Mail: info@hilscher.in

Italy
Hilscher Italia S.r.l.
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-Mail: info@hilscher.it

Support
Phone: +39 02 25007068
E-Mail: it.support@hilscher.com

Japan
Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-Mail: info@hilscher.jp

Support
Phone: +81 (0) 3-5362-0521
E-Mail: jp.support@hilscher.com

Korea
Hilscher Korea Inc.
Seongnam, Gyeonggi, 463-400
Phone: +82 (0) 31-789-3715
E-Mail: info@hilscher.kr

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch

Support
Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@hilscher.com

USA
Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-Mail: info@hilscher.us

Support
Phone: +1 630-505-5301
E-Mail: us.support@hilscher.com