Concept and ASIC Solution for CAN based Field Bus in Service Automation

Authors:
Matthias Isler **INICORE AG**
Mark Hug **Ascom Autelca AG**

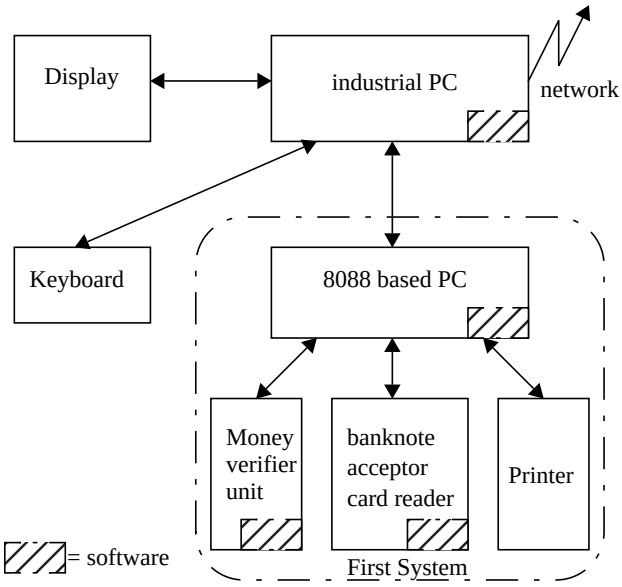
Date: 26 February 1997

Abstract
This paper describes the way from a conceptional restructuring of existing architecture with hard limitations to an open, field bus based concept, using innovative architecture, CAN, leading edge ASIC technology and advanced design methodology.

Table of contents:

Starting point

Ticket vending machines and any other service automation application require user interactive, easy to use human interfaces as well as improved service capability for tariff modifications, money verifier adaptations (false money) and general maintenance.
The internal architecture of a widely used ticket vending machine of **Ascom Autelca** reflects the whole history of feature enhancements over the years.
The following picture shows the actual implementation:



The first system, based on a 8088 PC, contained only the money verifier unit, a banknote acceptor or card reader and a printer for the tickets. The next generation provided already a large display, an enhanced keyboard and a supervisor PC with network access via modems. There where different software parts inside the system: The 8088 software, the money recognition and checking software, the card reader software and a last piece of code for the supervising PC. This architecture, a heterogeneous tree type, was chosen, because all components were standard PC IO boards, so it was cheap and easy available.
Today's needs of fast software updates, fast reaction on falsifications, advanced service facilities and flexible tariff demands could no longer be provided with the actual solution. The following problems occurred:

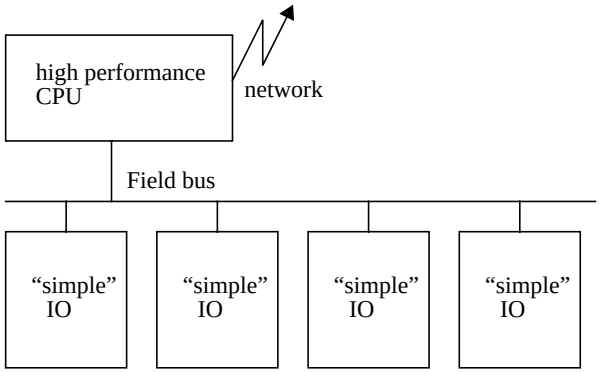
1. The whole device became too expensive
2. Not flexible in hardware and software
3. Software updates are very awkward (ROMs)
4. Expensive logistic:
   maintenance and compatibility of many pieces of software in ROMs.

The conclusion was clear: No future!

What new?

I had a dream...

Field busses are a well known vehicle to solve deadlocks like the above. The first dream about field bus solutions was on a basic structure, explained in the picture below:



Fieldbus solution, first approach

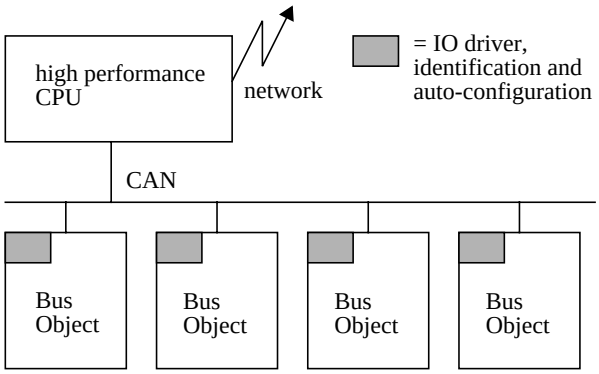
The first approach discussed was a field bus solution with "simple" input-output modules, connected to the bus and controlled by the host computer.
This has following advantages: The IOs are cheap and flexible, because the number of IO modules is not (really) limited. Adding new modules needs no system modification. The whole system becomes homogenous and much cheaper:

1. One central computer system is less expensive than many slow ones.
2. Less wiring (bus instead of star)
3. Less software development and engineering, since centralized software is better manageable than distributed solutions.
4. The software is easy to update by a remote access via network.

This approach with centralized software requires a field bus providing both *real time capability* and *throughput*.
After studying several fieldbus protocols, heading for CAN was obvious, since both aspects (real time and throughput) are fulfilled. An open higher layer was also demanded, since IO modules should be able to work without additional software. A complex higher layer would have been a killing argument for such an application. But...

More dreams...

The above approach could fulfil today's needs of service automation, but not tomorrow's. Update and maintenance aspects were not fully satisfied. So the dream went on:



Object oriented CANbus approach

This object oriented approach requires on each IO module a defined way to access it (driver), a clear identification and a version- and production number.
This offers the following amazing features:

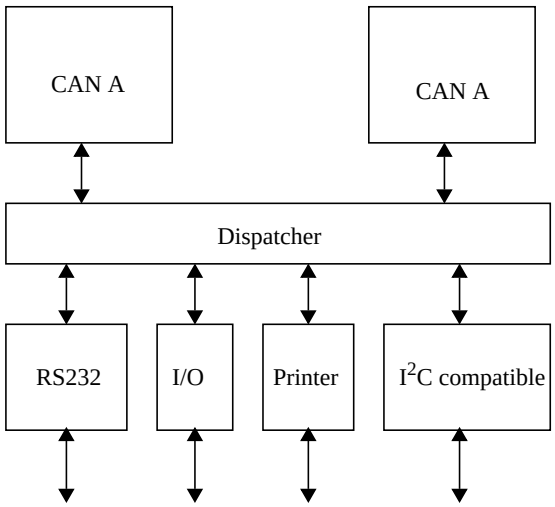
1. The device can be identified and automatically configured.
2. There are no version conflicts
3. Plug and Play
4. Service support guaranteed

Analysing this concrete solution was sufficient, but the real time aspects and data throughput were already today at its limits. To increase the real time capability, a second CAN channel was added to the bus (so every IO module would need two CAN controllers).
For additional throughput, internal communication can be pushed to a CAN bitrate of up to 4 Mbps...
Such a solution would also provide enough bandwidth for future applications like voice and compressed video over CAN.
In a few words: We need a dual CAN controller, able to overspeed at 4 Mbps, which has implemented a higher layer protocol to enable a plug and play solution with above requirements.

Nice, but... Who provides such devices?

Back to reality

The whole concept had a huge problem: We can't change the whole world (of service automation) at once. There must be a strategy for a migration path to the new, modern solution. Existing devices, which are perfectly working and reliable, must be adapted smoothly to the new concept.
The solution is the following: All nodes on the bus have two CAN controllers together with a dispatcher, which routes data from the CAN to standard interfaces. So the old subsystems like the money verifier unit etc. can be reused without any changes, when the following part is connected to it:

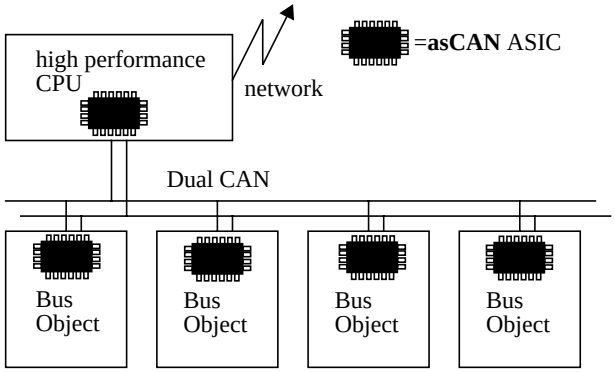


Connect existing IO modules to the CAN bus

This block diagram looks like a new subsystem - and it definitely is one! But it fulfils all needed requirements of a future oriented architecture which is able to reuse existing IO modules.

The solution

ASIC solutions like this require leading edge technology for providing both performance and resources of such demanding applications at a reasonable price.
INICORE has access to such technologies (LSI Logic), the cores to support a fast and sure development and the design methodology to manage such projects.
The final solution looked like this:



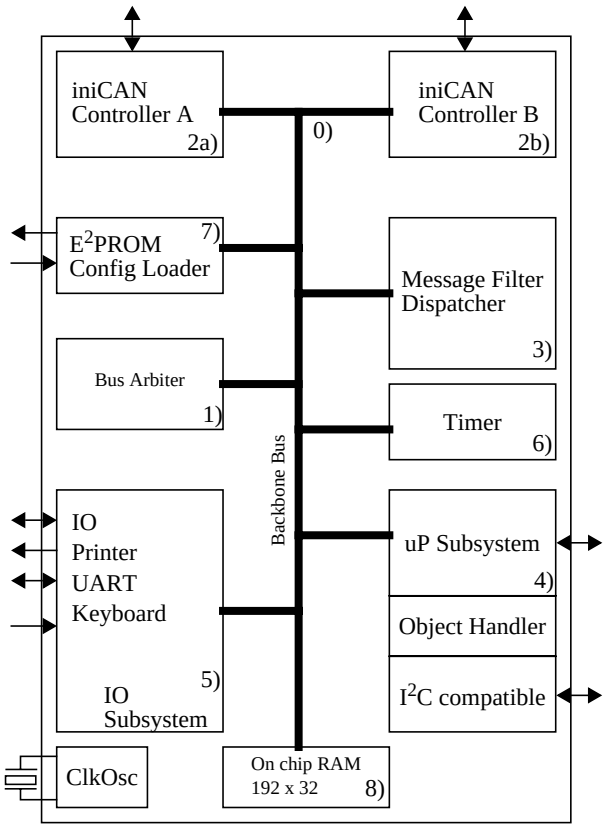
The final solution

In a first step, the **asCAN** ASIC would be used for the master node and as a bride for connecting existing devices to the dual CAN bus. In a later phase, the same chip will be directly used by subsystem manufacturers to provide access to the bus called "VendoBus". This strategy enables a straight migration path from today's solutions to tomorrows applications.

The major advantage of this solution is the fact, that the **asCAN** chip will be used on every node and guarantees the integrity of the network, since the whole networking aspects are already solved in hardware. So no version mismatch and compatibility problems will occur.

The asCAN ASIC

The **asCAN** implementation will be discussed in this chapter by means of the following block diagram:



asCAN ASIC Block Diagram

The **asCAN** chip, integrated on LSI 500k cell based technology is packaged in PQFP120. There are about 60k gates developed in a 10 months period by an inter-company team.
The **asCAN** architecture is based on a 32bit backbone bus, item 0), enabling a transfer rate of 340 Mbits per second, which connects all modules together. Data is all transferred via this bus. Exceptions are interrupts from the subblocks to the dispatcher and events from the **iniCAN** controllers. The Bus Arbiter 1) is handling bus traffic and arbitrates the different requests from the modules.

The **iniCAN controllers** 2a) and 2b) are based on silicon proved VHDL cores from **INICORE**. For this application, basic CAN2.0B was needed. Both controllers are identical and they have a customized interface for being connected by means of the backbone bus. All status (error counters, bus status, error flags etc. are readable via backbone access. Also, it is possible to shut down an **iniCAN** controller and to restart it without generating errors on the CANbus itself.
Further, it is possible to enable a self restart after a bus off state (regarding the specification!), or to handle it with the application software.
For protocol security, the **iniCAN** controller is protected against overruns and communicates such events to the system.
Since all functionality is described in VHDL and follows strictly synchronous design, verification of the customized **iniCAN** controller was successful on Actel FPGAs. This fact gave a very high confidence for the high complexity system integration.

Block 3) is the **message filter and dispatcher**, which analyzes the incoming CAN messages (for both controllers) and routes them to the blocks (defined by the identifier[1]). As well, block 3) routes CAN messages generated in the system (interrupts) to the CAN controller A or B, defined by the identifier.

Block 4) is the **microprocessor subsystem** containing the following blocks:

uP interface: Motorola 68360 type as well as Intel 8051 type processors may be connected to the **asCAN**. Further, it is able to reproduce a Motorola type bus in CPU less applications, where memory, peripheral devices etc. can be connected.

CAN Object Handler: In this block, there are 8 queues (using the internal memory) and 512! queues for SARCAN messages (see below), which are placed either in the internal memory (short queues) or in the processor memory by means of DMA transfers. The CAN Object Handler routes the incoming messages to the correct queue type and number. Also, it recognises I^2C compatible SARCAN frames and triggers the

I^2C compatible Interface: This interface is triggered when a complete frame is available in a queue. It is requesting data access via the backbone bus and transfers the same over the serial interface. Incoming messages are stored in a predefined queue. After a complete message, the interface generates an interrupt to the system. Frame lengths up to 256 bytes are supported.

Block 5) is the **input/output subsystem**. It is partitioned in following blocks:
Digital IO: There are 16bits of IO, which can be configured in groups of 4bits, either input or output. In case of input, it can be defined as polling input, short debounce (3us) or long debounce (30ms). Further, it is able to generate interrupts on level changes, which can be shaped to limit the

1. This mechanism is discussed later in the protocol description.

number of interrupts (which generate CAN messages!) in a given time. There is a central configurable shape timer (1ms to 256ms). In case of output, pins can be either totem pole or open drain types.

UART: The UART is used for standard RS232 communication, but also for a special 9bit data format.
The **iniUART**, designed in the same manner as the previous mentioned **iniCAN**, is available in **INICORE**'s library. The UART is supported by a 16 byte deep queue in receive and transmit direction. Receivers use a combined time-out/fill level algorithm to trigger CAN messages, while transmitters start sending as soon as data is available in the queue. The baudrate is generated by means of a frequency synthesizer, enabling baudrates from 1200bps to 64kbps (and higher) from the system clock.

Printer: A Centronics compatible printer can be connected to the IO subsystem, using the same queue and features as the UART. So printer and UART are exclusive.

Keyboard: A 4 by 4 matrix keyboard can be directly connected to the **asCAN**, providing interrupts on key press and release events.

All peripherals which communicate with off chip components have been verified in Actel FPGAs.

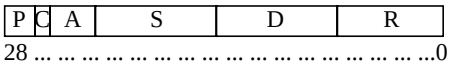
Block 6) is a **timer** which can be used to synchronize system wide. There are specially marked CAN messages for timer synchronisation.

To set up the system after reset, there is a **config loader** circuit (Block 7) reading the actual configuration out of an external serial E^2PROM and configurates the whole chip. It is also responsible for saving the actual configuration status of the running chip in the E^2PROM.
Further, the serial number, production number and node identification are stored and available. This is mainly used after boot up for a system wide configuration (plug and play).

Last but not least an **internal RAM** provides memory space for queues, temporary storage and other functions. It is organized in 192 x 32bit words and may be accessed via the backbone bus.

The VendoBus Protocol

The VendoBus protocol used in this application is adapted to the needs for service automation. It provides powerful support to transmit long frames, message routing and system auto-configuration.
The extended CAN identifier is used to determine the type of the message by:

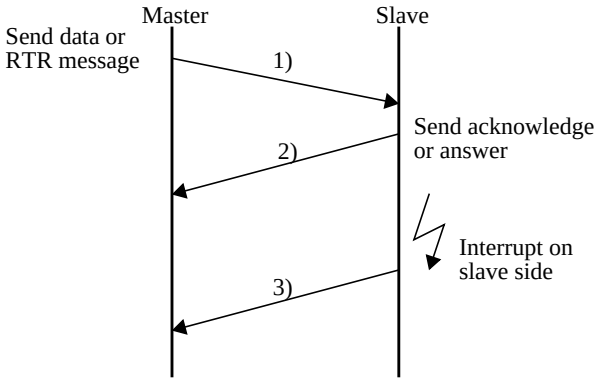


The P-field is used to define the priority of a CAN message. It also defines, which CAN controller is chosen for the actual transmission. Further, the P-field defines, if the other CAN controller may be used when the default one is bus off.
The C-field serves as 'confirm' status. Request, data and interrupt messages are marked with C=0, while answering and acknowledge messages being initiated by RTR messages (remote transmission request) are marked as confirm messages (C=1). This mechanism breaks down infinite message transfer loops between nodes, e.g. involved by transmission errors.
The A-field is the access field. It defines the general access type of a message. Legal types are boot, configuration, IO bit addressable, IO byte/word addressable, register map, SARCAN and reserved/unused (for foreign traffic). This field is used to route the incoming messages to the corresponding sub systems.
The S-field and the D-field are source and destination of a message. The source can be the master, which is sending to slaves or it may be the slave id, when interrupts and answers from the slave are sent to the master. The destination is either one defined slave (or the master for answers) or it is marked as broadcast (D=0). Such broadcast messages are accepted by all receivers.
The R-field is more universal. Its definition depends on the access type and may used as address inside the subsystem or it is used for signalling complex protocol types.

The communication between master and slaves can be either general purpose CAN or SARCAN.

General purpose CAN communication

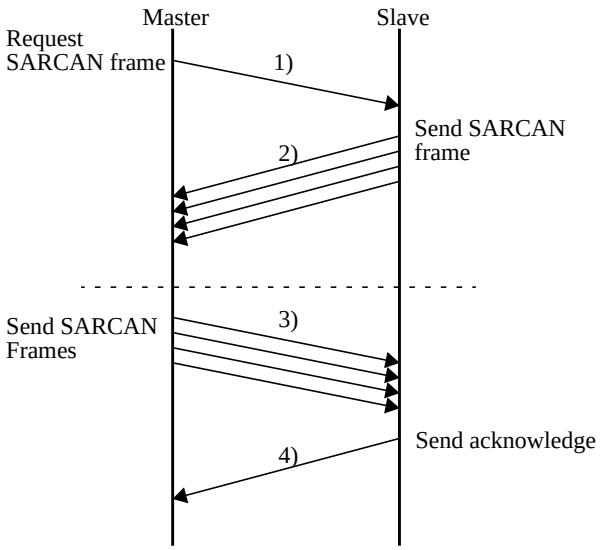
General purpose CAN communication (GPCAN) is used for transmitting and requesting data, interrupts and acknowledge. The acknowledge feature may be switched off to limit traffic on the bus.



General purpose CAN communication

SARCAN communication

SARCAN communication is used to transmit long frames, much longer than the CAN payload of 8 bytes. SAR is derived from ATM[1] telecom solutions and means 'segmentation and reassembly'.



SARCAN communication

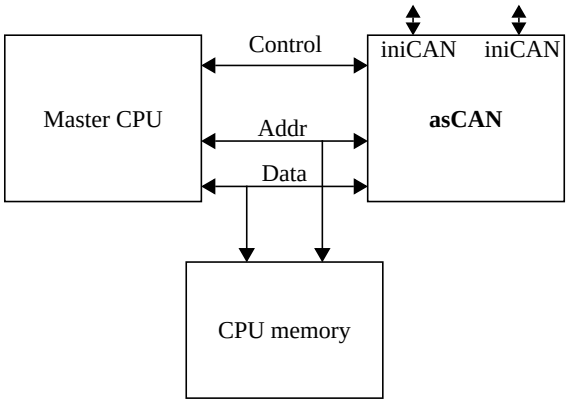
The goal is having a hardware support for long data like I^2C compatible frames and software download.

The **asCAN** provides a transparent channel for such frames. The frame is prepared in the uP's memory. By means of a descriptor, the **asCAN** will fetch data and transfer it via CAN bus to the destination. Numbering of the elements and a CRC check is inserted automatically (segmentation). At the receiver side, the elements are checked for the right order and data consistency and then reassembled to the initial frame structure. There is only one CPU interaction respectively one interrupt per transmitted frame and not per transmitted CAN message! This way, interrupt load can be limited to a minimum.

The asCAN application

The **asCAN** chip has been designed for being used in both master and slave nodes. Therefore, it has some hardware configuration codes, which define its functionality.

The master node

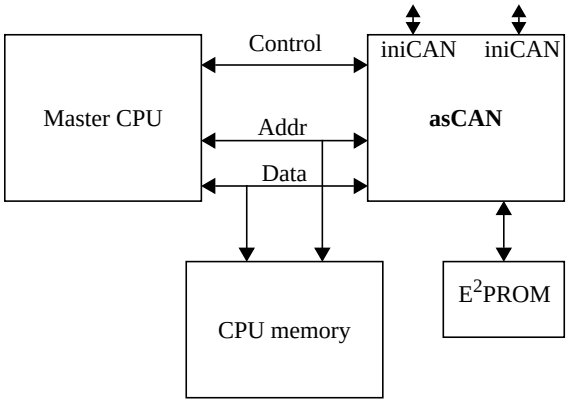


Master configuration

This combination serves as master node, where a powerful CPU and a large memory works together with the **asCAN** chip. The CPU memory is shared with the **asCAN** by means of powerful DMA[2].

The slave nodes

For slave nodes, there are several configurations possible. First, the same structure as the master application can be used, but an E^2PROM will be connected for auto-configuration and identification:

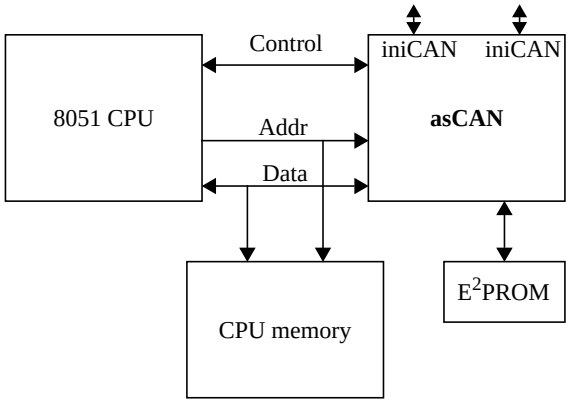


16bit uP slave node configuration

For slave applications that need a small micro controller, it is possible to directly connect a 8051 type micro controller. Since there is no DMA available on the 8051 device, the **asCAN** internal memory is used for SARCAN queues etc.

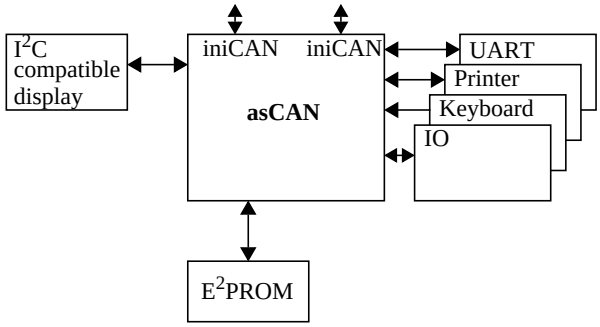
1. Asynchronous Transfer Mode

2. Direct Memory Access



8051 micro controller based slave configuration

A CPU less slave IO adapter is shown in the next picture, where the full range of IO is available. This configuration can be characterized as a high performance serial linked IO (SLIO). This is the general IO type and will be used to connect existing modules (keyboard, money verifier unit via RS232, displays via I^2C compatible etc.) It is remarkable, that all higher layer activity for protocol conversion, including queue manager, acknowledge, interrupts etc. from the different existing peripherals are managed in hardware by the **asCAN** chip.



Slave IO adapter

A somewhat special slave node is the register mapped IO. The **asCAN** is used to reproduce a 683xx type compatible bus interface with address, data and control signals which are remote controlled by the master via the CAN bus. A special message type enables register map access, where the logical placement of an IO register is located in the memory map of the master while the physical IO is located on a slave node. The VendoBus protocol provides transparent access to this remote IO, respecting the delay time of CAN message transfer. This feature enables a backplane application, where the dual CAN bus is used as backplane bus (at highest speed). In this case, many masters and many slaves are possible. By the way, this network type is greatly supported by the CAN bus architecture itself and enhanced by the VendoBus protocol implemented in hardware.

Conclusions

The **asCAN** solution opens a platform for new applications and is an available concept which can be directly used in similar projects.
It is a tremendous challenge to think over the controlling philosophy of today's machines and automates. For successful projects, the common '*standard chip thinking*' must be replaced by a consequent *system thinking*:

- System approach: The whole system must be reviewed. Optimizing isolated boards to reduce cost makes no sense
- Strong project management to guarantee working silicon in time
- High level ASIC design methodology to enable the above
- Availability of 'ready to use cores', together with the according application know how for using them
- Access to mainstream ASIC technology

Ascom Autelca will offer this concept, completed with chipset and/or modules, software and full support via **INICORE AG**.

INICORE AG
Mattenstrasse 6a
CH-2555 Brügg/Biel, Switzerland

Tel. ++41 32 374 32 00
Fax ++41 32 374 32 01
email: ask_us@sme.ch
http://www.inicore.com