CPU-less CANopen at 200°C

D. Leu, Inicore Inc.

H. Defretin, Schlumberger Oilfield Services Inc.

This paper presents a cost effective CANopen implementation of a system to be deployed in an extreme environment exceeding MIL-STD operating temperature range of -55C to 125C. Applications such as Deep Oil Field exploration tools require robust electronic systems that can operate at 200C and beyond which very often results in the miniaturization of the electronics. To operate in this severe and noisy environment, a CANopen network for inter-module communication was chosen.

Contrary to a standard implementation, a CPU-less approach was selected where the entire CANopen stack is implemented in a Field Programmable Gate Array (FPGA). The FPGA is a System-on-Chip (SOC) integration where the CANopen stack is coupled with a CAN controller and peripheral logic for analog and digital I/Os. This CPU-less approach meets the size, cost, reliability, and power consumption requirements.

Introduction

To optimize hydrocarbons production and to forecast reservoir reserves, the reservoir has to be characterized on a regular interval. For this, the oil production is stopped and a tool string is deployed into the well to perform a variety of measurements. To limit the production loss, the measurement time should be short and provide as much data as possible in a single downhole trip.

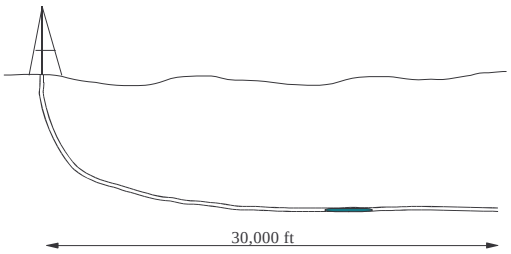

Figure 1: Oil well

Figure 1 shows a diagram of an oil/gas well with the inserted tool string.

For the reservoir characterization, some of the operations performed inside the well are:

- Temperature and pressure profiles
- Reservoir and core samples

An oil/gas well can be up to 30,000 feet (10 km) in length. Temperatures can exceed 200C. The diameter of the well varies from 6 to 14 inches, which puts tight constrains on the size of the tool string electronics.

To increase the system response time and reduce data bandwidth requirements, downhole data preprocessing is desired.

Embedded Networking

The tool string is made of several modules as described in Figure 2.

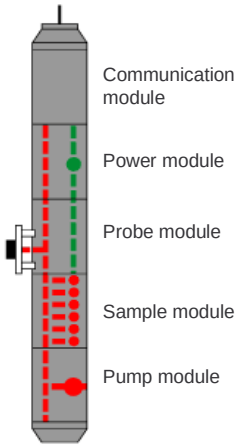

Figure 2: Tool string

The electronic boards inside each module are connected via the CAN bus. One CAN node serves as the bridge between the local CAN bus and the backbone, which provides the communication link to the surface. The other boards provide the interface to sensors and other devices such as motors, solenoids, and seal valves.

Figure 3 shows an overview of a module with its sub modules.

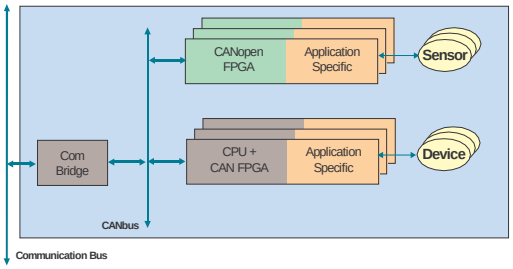

Figure 3: Module with electronic boards

CANopen was chosen as Higher Layer Protocol (HLP) due to its simplicity and flexibility. Using a well established standard has helped to coordinate the development team spread around the globe. The CANopen protocol was even embedded into the proprietary surface link providing an almost transparent communication path from the surface down to the individual sub modules.

During the development, several application specific requirements lead to some adaptation of the CANopen protocol to provide better support:

PDO Packet

Several sub modules provide big data samples which extend far beyond 8 bytes. Since the number of local CAN nodes never exceeds 32, two of the ID bits are used to group several PDO objects into a bigger PDO Packet (PDOP):

COB-ID										
10	9	8	7	6	5	4	3	2	1	0
Function Code				t	c	Node-ID				

Table 1: Identifier allocation scheme

t: Toggle bit

Used for flow control in PDOP messages. This bit toggles with each PDOP message and is always 0 for other messages.

c: Continued bit

Used for flow control in PDOP messages. Bit is one if message continues in next PDO message.

Figure 4 shows the message flow using the PDO Packet protocol for a receive PDO message.

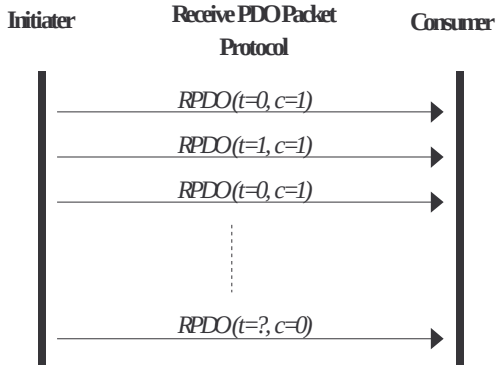

Figure 4: RPDOP Protocol

NMT dynamic node assignment

LSS would provide dynamic node assignment services, but its implementation is too complex for this system. A simplified approach using the NMT master message with a custom command was taken to set or change the Node ID of a target.

Architecture and Technology Selection

To reliably design a tool to operate at high temperature, the selection of high temperature components is very challenging. Off-the-shelf components cannot always be used and in some cases, repackaging is necessary to withstand extreme temperatures. Since no low-cost microcontroller with an embedded CAN controller has been identified for high temperature application, an FPGA implementation has been investigated as a potential alternative.

To reduce electronics size and cost, two types of CAN slave modules are currently implemented:

- CPU CAN gateway
  The CPU CAN Gateway is made of a CPU coupled with an FPGA that contains the CAN controller plus additional

application specific logic. Figure 5 shows the block diagram.

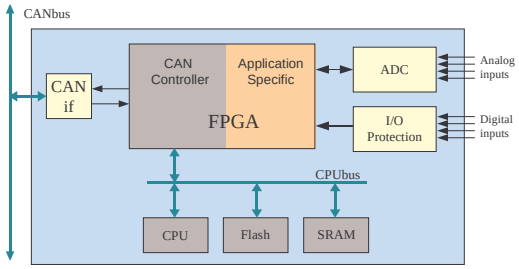

Figure 5: CPU CAN Gateway

- FPGA CAN Gateway
  The FPGA CAN gateway is a low cost application solution where local data processing is not required. Coupling the CAN controller with additional logic demonstrates the possibility of implementing the CANopen protocol stock in a single chip.

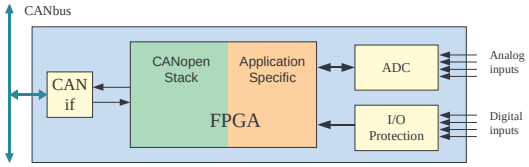

Figure 6: FPGA CAN Gateway

This CPU-less approach resulted in an implementation with fewer components and reduced board space, which leads to increased reliability and reduced total system costs.

"Standard" CANopen Implementation

In a typical CANopen implementation, a CPU complemented by a selection of peripheral interfaces is integrated onto a silicon die. Either an external or an on-chip CAN controller provides the network interface. Figure 7 shows the block diagram of a typical CAN controller. To be operational, external data and program memory have to be added.

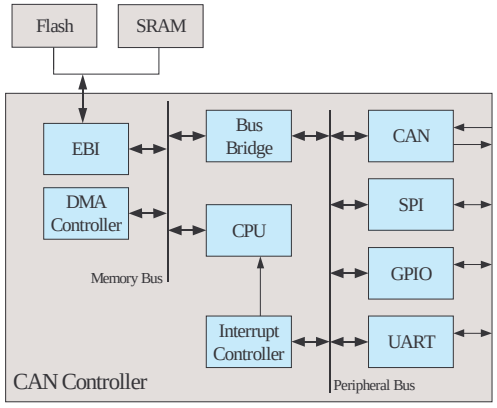

Figure 7: Typical CAN Controller

In such systems, the CPU performs the entire operation. Depending on the implementation, either a small operating system or a simple main loop controls operation of the different tasks. An example for the main loop of a simple I/O controller is shown in figure 8.

The entire main loop is a sequential proc-

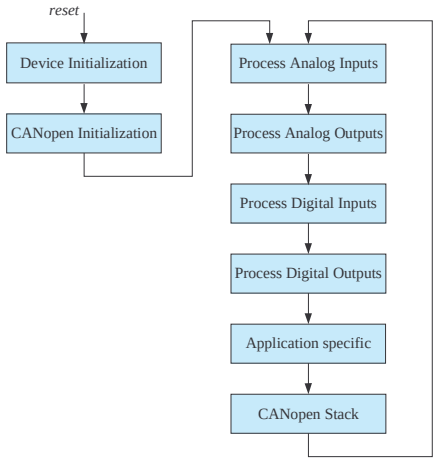

Figure 8: Application main loop

ess of several individual tasks. Some signal or data processing operations might require a lot of time and therefore delaying all other tasks. Instead of using a simple loop, a more advanced process queue with selectable process priority can help to avoid timing conflicts. Still, if certain time critical tasks have to be performed, a move to a more powerful CPU might not be avoided.

CANopen in an FPGA

As shown in previous paragraph, the regular handling of the CANopen stack, CAN controller, peripheral interfaces, and data processing requires a certain CPU performance.

Instead of having a central CPU that processes everything, a different approach has

been chosen to implement the CANopen stack with all the peripheral modules.

An FPGA can be thought of like a massive parallel architecture where several engines work in parallel. This parallel processing approach is shown in the block diagram in figure 9.

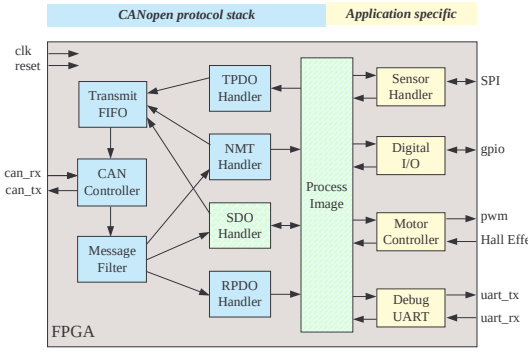

Figure 9: CANopen FPGA

Each peripheral interface is implemented as an autonomous controller or as an application specific state machine. They operate independent of the CANopen stack. The CANopen stack itself consists of several independent operating modules. The internal communication between the protocol stack and the application specific interfaces happens through the Process Image.

The different blocks shown in figure 9 implement following functions:

- CAN Controller
  This is the basic CAN protocol framer.
- Message Filter
  The message filter is used to route the received messages to the target process and discard messages that are of no interest for this node.
- RPDO Handler
  The message content is stored in the Process Image according to the fixed RPDO mapping parameter.
- TPDO handler
  Depending on the state of the PDO transmission parameter state, a new PDO message is generated using data stored in the Process Image and transferred into the transmit FIFO.
- Tx FIFO
  All outgoing messages are put into an intermediate FIFO before being transmitted.
- NMT Handler
  This controller implements the NMT state-machine and the dynamic node assignment.
- SDO Handler
  The entire Object Dictionary is hard coded inside the SDO Handler. SDO requests are received and executed and the response is put into the transmit FIFO. The SDO Handler also generates heartbeat and emergency messages.
- Process Image
  This is the central data memory where all the received and transmit PDO data is stored. The Process Image contains the application specific configuration data too.
- Sensor Handler
  Instead of having low-level peripheral interfaces such as an SPI controller, application controllers are used to perform entire data processing sequences. The Sensor Handler performs autonomous data acquisition using an external SPI based ADC. Local offset and gain correction can be enabled to compensate for ADC nonlinearity.
  The Sensor Handler is configured using an SDO object providing options such as:
  - Sampling frequency
  - Enable automatic offset and gain correction
  - Enable data over sampling
  - Start/stop control
- Digital I/O
  The digital I/O module updates its output according to the state of the Process Image. Accordingly, the input state is reflected in the Process Image too. Generation of a new TPDO message depends on the configured communication parameters.
- Motor Controller
  The motor controller provides a high-level command interface. Using RPDO messages, a motor can be controlled through parameters such as start, stop, and revolutions per second.
- Debug UART
  For system debugging, a communica-

tion channel is available to report internal event information and provide access to CANopen stack status information.

Supported CANopen features

Having the CANopen stack integrated in hardware leads directly to the question of CANopen compliance. In order to leave as much FPGA space to the application specific portion, certain CANopen features that don't provide a direct benefit to the application had to be sacrificed.

The current implementation supports following features:

- SDO (expedited access only)
- NMT
- PDO (4 RPDO, 4 TPDO)
- PDO communication parameter
- EMCY
- Heartbeat

Protocol features that were not required in this system were waved while preserving CANopen compliance:

- Segmented SDO
- Fixed PDO mapping parameters
- COB-IDs are fixed

To minimize overhead and processing inside the CANopen stack, a custom device profile was selected.

FPGA Verification

Prior to powering up the development board to check for proper operation, the CANopen FPGA was verified in VHDL simulations. The entire system was modeled, as it will be used in the real application. Figure 10 shows the block diagram of the transaction-based verification environment.

The testbench is structured into several blocks each implementing a particular function or model:

– Stimuli Generators
  Several stimuli generators are used to verify proper operation of the CANopen FPGA. A stimuli generator controls the entire operation of a test and verifies that the device under test performs as expected.

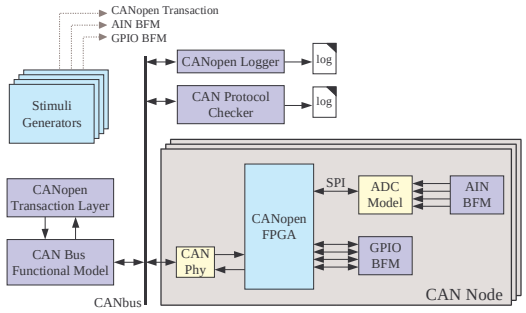

Figure 10: Transaction-based testbench

– CANopen Logger
  For error analysis all CAN transactions are reported in the context of the CANopen protocol.

– CAN Protocol Checker
  The CAN protocol checker verifies integrity of the CAN messages and reports errors.

– CAN Bus Functional Model (BFM)
  The CAN BFM generates and receives CAN messages.

– CANopen Transaction Layer
  The CANopen transaction layer translates CANopen function calls into proper CAN messages. Received CAN messages are decoded and provided as CANopen type messages.

– GPIO BFM
  Using this module, the GPIO pins of the FPGA can be controlled from the stimuli generator.

– ADC Model and AIN BFM
  Using the AIN BFM, the analog inputs of the ADC Model can be stimulated. These values are returned by the ADC Model upon receipt of the proper SPI sample command.

Transaction-based verification uses a higher level of abstraction. Instead of controlling individual signals by setting them to one or zero, signal operations are executed as entire transactions. The stimuli generator uses function calls to execute transactions. These transactions are converted into the individual bit sequences using a Bus Functional Models (BFM).

Figure 11 shows an extract of the VHDL stimuli generator where successive transactions modeled as procedure calls are use to read the device type entry of a particular CAN node.

This higher level of abstraction results increased efficiency and enables reuse of

the testbench environment and its components.

```
index := x"1000"; subindex := x"00";

tb_can_bfm_pkg.put_message(
 sdo_read( node_id, index,subindex),
 can_cmd, can_stat
);

tb_can_bfm_pkg.get_message(
 sdo_read_response( node_id, index,
   subindex, c_canopen_device_type
 ),
 can_obj, can_cmd, can_stat,
 "sdo: device_type error"
);
```

Figure 11: CANopen device type read

Using this transaction-based testbench, the entire FPGA functionality was verified prior of deploying the FPGA in the lab. This approached proved very successful where the FPGA was fully functional upon powering the development board for the first time.

Conclusion

In this paper we showed that special design constrains for a CANopen implementation require a new approach. An FPGA implementation was presented that contains a hardware implementation of the CANopen stack complemented by several peripheral interface controllers. While still being CANopen compliant, some of the protocol features had to be sacrificed. This allowed for a successful development of a high temperature, cost, and space effective solution: a CPU-less single chip implementation of the CANopen stack working at beyond 200C.

Daniel Leu
Inicore, Inc.
5600 Mowry School Rd Ste 180
Newark, CA 94560
Tel: +1 510 445 1529
Fax: +1 510 656 0995
E-mail: daniel@inicore.com
Web: www.inicore.com

Harmel Defretin
Schlumberger Oilfield Services Inc.
125 Industrial Boulevard
Sugar Land, TX 77478
Tel: +1 281285 8883
Fax: +1 281 285 7826
E-mail: defretin@slb.com
Web: www.slb.com