FPGA-based CAN Solutions

Daniel Leu

In the past, CAN-based products either used a standard CAN controller chip, a microprocessor with a built-in CAN interface, or a custom ASIC with CAN interface for high-volume applications. Now with the advances in silicon technologies, a new option is readily available: field programmable gate arrays (FPGA). An FPGA is a gate array that can be programmed during production or later when deployed. They are produced in high volume, which lowers individual device cost.

FPGA vendors advertise their biggest multi-million “system-gates” devices. Although few applications actually require them, they enable implementing other products. Large devices need the most advanced silicon process technologies. Since each FPGA family comprises several members ranging from large to small gate counts, small devices are available on the same technology. With the high gate density available in these advanced process technologies, even small devices provide sufficient resources for system integrations. Suddenly, small FPGAs offer solutions that were not available before and cost just a few dollars.

The following list shows some of the advantages of using an FPGA-based product:

- Flexibility for functionality upgrades: Most FPGA technologies nowadays support in-system programming. Instead of exchanging hardware, a simple device reconfiguration adds a new feature or fixes a bug.
- Long-term availability: FPGA devices tend to be available longer than some standard integrated chips. Furthermore, using a technology independent design methodology, an existing design can be migrated to new technologies to take advantage of additional features and reduced costs.
- Wide operating range selection: commercial, industrial, automotive, military, space hardened. Most IC vendors do not support extended operating range products. Using FPGAs, users can select the most appropriate technology.
- Product differentiation: By having a piece of custom silicon, unique product features can be added that help to make the difference in the market place.
- The upfront investment is minimal, since common FPGA design software is available from FPGA vendors at nominal cost, if any cost at all.

CAN Intellectual Property (IP) Cores

Designing an FPGA-based CAN system does not mean that everything has to be developed from scratch. The CAN interface can be implemented by using a intellectual property core. An IP core is a pre-designed and verified functional module that can be integrated together with the application logic into an FPGA. As with stand-alone and embedded CAN controllers, CAN IP cores have different features and capabilities. Differences of commercially available cores are:

Bus Interface	• No bus: the interface is message-based • 8/16/32-bit wide bus interface • Standard-compliant interface: AMBA APB, OCP, Avalon, Wishbone, etc.
Message filtering concept	• Number of available filters • Covered message bits
Message queue architecture	• FIFO • Mailboxes
Support for higher layer protocols	• Native CANopen Sync support for improved timing accuracy • Message filtering on first two bytes of data field for better DeviceNet support
Debug features	• Listen only mode • Internal message loop back: message is looped inside of the core (no bus traffic) • External message loop back: a copy of all sent messages is received
Technology neutrality	• IP core is technology-neutral • IP core is targeted (as a netlist) for a particular vendor or device

By selecting the “right” core, an application-optimized implementation is only a few steps away. In the following few paragraphs, several different applications are presented.

Simple CAN Slave Node

A lot of CAN nodes provide simple analog or digital I/O functions. These nodes do not require a complex CPU to connect the peripheral functions to the CAN bus. A simple FPGA, containing a CAN controller and dedicated peripherals, does the job. Since an FPGA has no on-chip non-volatile storage and just incorporates digital logic, some additional components may help to provide common functions:

- Device configuration such as node-ID is stored in external EEPROM.

- Analog interfaces are implemented with external A/D converter and D/A converter components.

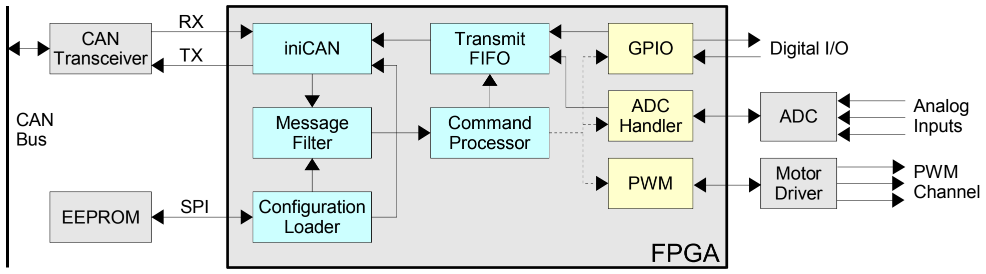

Figure 1: Simple CAN Slave Node

Few blocks are required to design a simple slave controller:

- iniCAN: This is the low-level CAN controller IP that provides a message level bus interface.
- Message filter: Only messages for this node are forwarded to the command processor.
- Transmit FIFO: Local events such as I/O change of state may result in a new CAN message. All outgoing messages are stored in a FIFO and transmitted by the CAN controller once the CAN bus is idle.
- Command processor: Incoming messages are decoded and the proper action is taken.
- Configuration loader: Most of the CAN nodes have to be preconfigured. At power-up, the configuration is loaded from external EEPROM.
- Peripheral interfaces such as general purpose I/O (GPIO), pulse width modulator (PWM), or interface towards an external A/D converter (SPI- or parallel bus based)

The features implemented in the peripheral section are only limited by the available resources. Even with small packages such as a TQ64, more than 40 I/Os can be used for general purpose input/output. More advanced peripherals can be added to support applications such as a motor controller using a PID algorithm supported by PWM and A/D converter peripherals.

CANopen-based Slave Controller

Most CAN networks use a standardized higher-layer protocol (HLP), such as CANopen. Using an HLP makes it easier to put a network together and get everything up and running in a short time. Furthermore, a wide range of tools support the development.

Due to the simplicity of the CANopen protocol, the FPGA implementation of a CANopen slave controller does not require a microprocessor. The entire CANopen stack can directly be implemented in hardware. A common process image is used as interface between this stack and the application section where the peripheral interfaces reside.

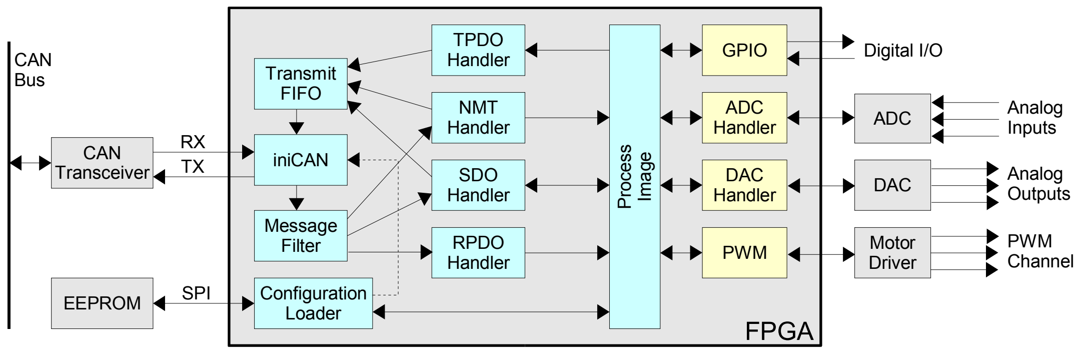

Figure 2: CANopen-based slave controller

Inside the CANopen stack, the data and control paths are separated right at the message filter boundary. Depending on the type of CANopen message, dedicated modules decode the content of the message and respond or update the process image.

An FPGA external non-volatile memory is used to store the node configuration. These parameters can be updated via SDO commands. The nodes initial configuration is loaded at power-on.

The main blocks of this FPGA device are:

- iniCAN: This is the basic CAN controller.
- Message filter: The message filter is used to route the incoming messages to the target module and discard messages that do not address this node.
- RPDO handler: The message content is stored in the process image according to the RPDO mapping parameters.
- TPDO handler: Depending on the transmit PDO state, a new PDO message is generated using data stored in the process image. This message is loaded into the transmit FIFO for transmission.
- Tx FIFO: All outgoing messages are put into an intermediate FIFO before being transmitted.
- NMT handler: This is the NMT state-machine.
- SDO handler: The entire Object Dictionary can be accessed and modified using SDO messages.
- Process image: This is the central memory unit where all process variables and states

are stored. It is used as the common interface between the CANopen stack and the application logic.

- Application: Similar to the previous application, external components such as non-volatile memory, A/D converter, and D/A converter complement the application.

PCI-CAN Bridge with DMA

The system interface is an important aspect of a design. This is where the data exchanges happen between the system and the peripheral or application ports. System performance benefits from a smart definition and implementation of the system interface.

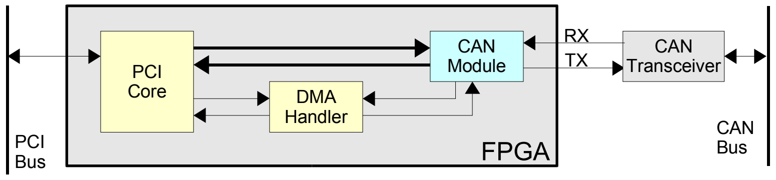

Figure 3: CAN-PCI bridge with DMA

A simple PCI implementation maps the CAN controller into the system memory space. All data transfers are interrupt-driven. An interrupt is issued when the receive FIFO is full, another interrupt is issued when the transmit FIFO is empty. Depending on the amount of CAN data traffic and the size of the FIFOs, this can have quite a negative performance impact. Such an implementation typically uses a standard PCI device controller, a CAN controller and some glue-logic to connect the two components together.

All FPGA vendors provide PCI cores for their FPGA devices. Combining such a PCI core, and a CAN core, and wiring them together in an FPGA results in a single chip PCI-CAN controller. Obviously, applications that require several CAN interfaces would have several CAN controllers on the same FPGA.

To address the system interface performance bottleneck, a local DMA controller can be added. Now, incoming CAN messages are filtered and transferred to the system memory without CPU interaction; outgoing CAN messages are directly fetched from the system memory based transmit queue.

A DMA based data transfer not only reduces the number of interrupts, but also allows for a much more efficient transfer, since CAN messages are transmitted as 32-bit wide data bursts.

This example PCI-CAN bridge shows an application using a PCI local bus. In an alternate design, the system could be VME based or could have a simple memory bus with DMA support.

Intelligent PCI-CAN Interface

In a real-time system, the performance limitations of going through the PCI bus to get to the system processor might still be too restrictive. Expanding the PCI CAN bridge application with a local embedded CPU provides an immediate performance boost. Now the host application can run on the embedded CPU and locally handle all CAN messages. The PCI bus is only used for system status updates or system commands.

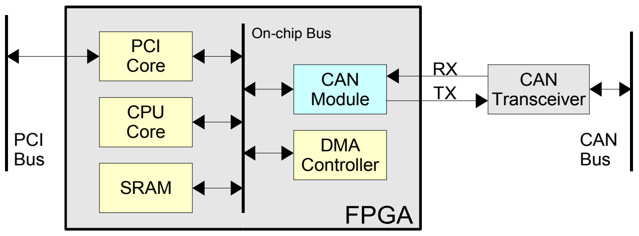

Figure 4: PCI-CAN interface with computing power

Conclusion

As shown in this article, using a state-of-the-art FPGA together with an embedded CAN core provides a new option for product design. A developer is not constrained by the available devices, but can select the proper architecture to support his application. This results in new products that differentiate themselves on the market. This makes FPGA-based CAN implementation very attractive from a technical, as well as an economical point of view.

Contact Information

Daniel Leu
INICORE INC.
5600 Mowry School Road, Suite 180
Newark, CA 94560
t: 510 445 1529 f: 510 656 0995 e: daniel@inicore.com
www.inicore.com