A CANopen compliant bootloader for Atmel's AT90CAN128

Damien Grolleau, Atmel and Christian Keydel, Embedded Systems Academy

In-system programming (ISP) of flash memory in an embedded microcontroller is the state-of-the-art method for updating program code and application parameters

This paper gives some background information on why a standard method for reprogramming via CAN bus is desirable, and how CANopen provides the necessary framework for it.

This paper also presents an existing implementation of a CANopen bootloader. The CANopen bootloader is compatible with CiA standard DSP-302 which means that any master node or an SDO client or CANopen configuration tool running on a PC can update the firmware of such a node, using SDO write access to an Object Dictionary domain entry.

Introduction

Many microcontrollers on the market today have flash memory on-chip and offer in-system programming. Quite frequently, the vendors offer proprietary software tools that run on PCs and communicate with the chip over a serial interface, and, in some instances, also via CAN. In cases where it is not desirable to have a non-standard protocol being used on the CAN bus, or where the downloading is not being initiated by a PC, it is beneficial to have a standard-based bootloader that offers more flexibility. Sometimes these nodes are deeply embedded inside a machine so mechanisms to remotely control the bootloading process are required as well.

Summary of the CANopen Framework DSP-302

The CANopen Communication Profile (DS-301) defines the basic communication mechanisms for exchanging data via a CANopen-based network. This includes the structure of the object dictionary, the network management and boot-up as well as communication objects like PDO, SDO, SYNC and time stamp. The object dictionary provides a standard interface for accessing communication parameters as well as process data. The part of the object dictionary which describes the general device and communication parameters is common for all devices types.

Application specific functionalities which are provided by certain device types are detailed in specific device profiles. A device profile is always based upon the definitions in the communication profile.

In general the mechanisms which are specified in the communication profile are sufficient for the definition of profiles for devices which, on the application level, provide some kind of I/O functionality.

Example devices include I/O modules, drives and regulators. These devices whilst they may be complex are not termed 'intelligent' as they do not run an application level program.

For the description and operation of intelligent devices further mechanisms are necessary which are specified in DSP-302. DSP-302 has to be regarded as a framework for the definition of device profiles for intelligent or programmable devices in form of an extension to the communication profile DS-301. The additional mechanisms specified in DS-302 are useful especially for intelligent devices like PLCs, HMIs or CANopen tools.

DS-302 comprises the following mechanisms and definitions:

- The term CANopen Manager is introduced to specify the functionality of a network controlling device more clearly.
- Definition of the boot-up process and related objects.
- A possibility for configuration of unconfigured nodes during system boot-up by means of a Configuration Manager.
- The dynamic establishment of SDO connections between devices. Dynamic SDO connections are handled by the SDO Manager.

- The definition of dynamically allocated entries in an object dictionary which can be used for the representation of I/O data e.g. on programmable nodes like PLCs.
- A general mechanism for downloading program data and functions for the control of programs on a device.

Some of these new mechanisms are also useful for intelligent or programmable devices.

Overview of the Atmel AT90CAN128

The Atmel Flash + CAN micro controller is perfectly suited for a CANopen application, with ample memory and processing power to accommodate the optional entries as well as the mandatory ones.

128Kbytes of flash contains the program. It can be reprogrammed up to 100K times. 32KB is large enough to contain the CANopen protocol stack (MicroCANopen: Only 4-5Kbytes) plus a good size application.

Depending on the configuration and the protocol stack used, only between 100 (MicroCANopen small configuration) and 450 bytes of the RAM (4Kbytes) will be used for the CANopen stack, leaving a large amount of RAM for the user application; pointers, stack and other variables.

4Kbytes of E2PROM are available to maintain information such as S/N, configuration data etc.

The AT90CAN128 CAN controller supports 15 Message Object Buffers with each of them programmable as Transmitter or Receiver. Each Message Object has an 8 byte dedicated message data FIFO, a dedicated Message ID and Message Mask (for receiver).

This is ideal for CANopen slave nodes where MOBs can be dedicated to CANopen objects for both receive and transmit. An example implementation of a CANopen slave node with optional TPDOs RPDOs shall illustrate a possible allocation of 12 of the 15 total Message Object Buffers:

- NMT
- Sync
- Emergency
- Time Stamp
- 1st TPDO could be replaced by 2nd TPDO
- 3rd optional TPDO
- 1st RPDO
- 2nd optional RPDO
- 3rd optional RPDO
- TSDO
- RSDO
- NMT

The Message Object Buffer structure is given in figure 1 below.

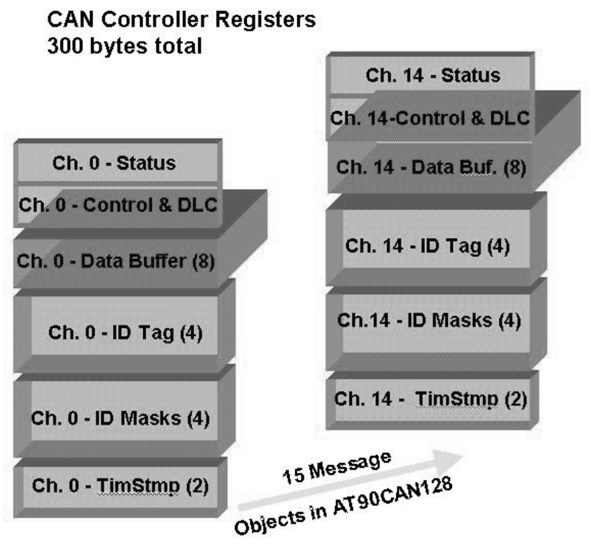

Figure 1 : Message Object Buffer structure

The block diagram of the AT90CAN128 is shown on the following figure 2.

The AT90CAN128 is manufactured using Atmel's high-density nonvolatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The boot program can use any interfaces to download the application program in the application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel AT90CAN128 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

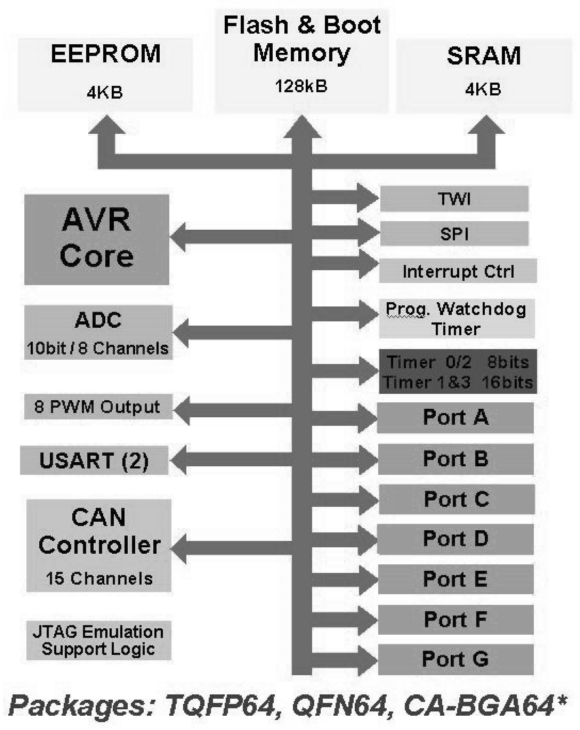

Figure 2 : AT90CAN128 Block Diagram

A CANopen Compliant Bootloader

If a CANopen device has flash memory, it makes sense to also have a CANopen compliant bootloader. This does not only free the serial communication channel from the bootloader task, it also allows using standard CANopen configuration tools as the communication partner providing the new code to be loaded into the flash memory.

Since the bootloader's task is to download a new application, including the CANopen stack into flash memory, the bootloader has to be separated from the regular application and we will have two different modes: The *boot mode* and the *application mode*.

When being in boot-mode a CANopen node’s only purpose is to accept a hex file to be loaded into a flash memory. While being in that mode, the node does not really need to be 100% CANopen compliant. It just needs to provide enough CANopen compatibility that it does not interfere with any other communication on the network and that it provides a fully functional SDO server, so that SDO clients (like Masters, Managers or Configuration Tools) can make read and write accesses to the Object Dictionary in the node.

So the only CANopen features and communication channels that need to be implemented are the SDO server and the SDO request and response channels.

Sometimes it is desirable that the bootloader can be activated without the requirement to physically touch the device (like setting a jumper, switch or button). In CANopen the straightforward method would be to use a selected write sequence to an Object Dictionary entry as an additional command to actually switch the node into the bootloader mode.

Minimal Set of Required OD Entries

OD entry [1000h,00]: Device type information, read-only

As there is no device type number standardized for a bootloader, a manufacturer specific value can be used. The Embedded Systems Academy uses 746F6F62h (ASCII representation is “boot”) in their bootloader implementations by default.

OD entry [1001h,00]: Error register, read-only

The bootloader can use this register to signal flash erase or programming failures. Setting the manufacturer specific error bit can indicate an out-of-range error – a try to program a memory location that is either protected or at which there is no flash memory.

OD entry [1018h,00-04]: Identify Object

The standard Identify Object as specified in CANopen DS-301.

OD entry [1F50h,XX]: Download program data

This Object Dictionary entry is described in DSP-302 and used to directly accept the code programmed into the target memory. Sub-index 0 is used to quantify how many different program or flash memory areas are available. The following Sub-indexes can each handle the download to one program or memory area. For many applications it is sufficient to implement one area (Sub-index 1). The download subentries are of the data type *domain*, i.e. data of unspecified length. Even though SDO block transfer mode could be used for this, SDO segmented transfer is more practical. It keeps the required RAM for buffering and the complexity and size of the code low. Also, it provides for a relaxed timing: The SDO client of the communication (=the loading program or node) will always wait for the bootloader node to send the SDO response before it sends the next packet of data. Therefore, even if some lengthy function has to be executed in between (e.g. erasing of a new flash page) no overflow or communication backlog can happen. This improves the reliability of the whole process.

Although not specified by the standard, the Embedded Systems Academy recommends using standard ASCII hex files as the files containing the program or data. Using a hex file has two benefits: the file contains the target address on where the data needs to be programmed to and the file also contains checksums making the downloading process more secure. To reduce the amount of data and improve download speed, a binary hex format can be also supported where ASCII hex bytes are replaced with binary bytes. This reduces the file sizes by about 50%. To further improve download speed, the SDO response for the individual segments can be shortened from 8 to 1 byte since it doesn't contain any payload. This is no longer 100% standard-conform, so the user has to make sure that the tool or code library and CAN controller that acts as the SDO client for the communication tolerates the short SDO response. The speed gain depends on the baud rate, processor speed, and other factors, but can be typically around 30%.

Although the AT90CAN128 flash memory doesn't need to be erased first before being reprogrammed, Embedded System Academy has implemented a specific erase command. For example, an erase could be initiated by sending the value 6D726C63h (ASCII representation is “clrm”) to the Object Dictionary entry [1F50h,01] (or other Sub-indexes to differentiate between different blocks or segments of flash or other non-volatile memory).

To signal erase or programming errors the SDO abort option is used. Hereby the programmed node responds to the SDO download (write) request with a suitable SDO abort code that can be either a standard or manufacturer-specific code, for example:

20000008h: "Data cannot be transferred or stored to the application"

Using SDO abort codes, the SDO client device doesn't need to poll object 1000.

OD entry [1F51h,XX]: Program Control Object

This Object Dictionary entry is described in DSP-302 and used to control the program(s) downloaded to [1F50h,XX]. The essential command to implement is “Start program” which requires writing a 01h to the Object Dictionary entry. For example, if a program was downloaded to [1F50h,01] it can be started by writing a 01h to [1F51h,01].

This Object Dictionary entry could also be used to implement the activation of the bootloader itself. So if the regular CANopen application running on this node supports this entry, it should activate the bootloader upon receiving a 00h (zero =Stop Program).

Security checksum and protection

Two challenges are inherent to using a bootloader:

- Assuring the integrity of the loaded application
- Protecting the bootloader code

In order to assure the integrity of the loaded application the checksum that is included in each data record of the hex file is not sufficient: While it can be used to verify the integrity of each record during programming, these checksums are not stored in the device and a later alteration of the code ("flipping bit")_cannot be detected. So an additional checksum for the whole application is needed. To accomplish this, an external tool provided along with the bootloader code calculates a checksum (e.g. 16-bit "TCP/IP checksum") and adds it to the load file so that it gets programmed at a well-defined address into the flash memory along with the application. On startup, the bootloader always executes first, calculates the application checksum using the same algorithm and compares it with the stored one. If they match, the application is started. If they don't, the bootloader starts in bootloader mode, i.e. sends its CANopen bootup message and waits for SDO requests.

The bootloader itself needs to make sure that it can't get erased or partially overwritten by accident. Therefore, it must have a write protection built into it that guarantees that in no case it will ever erase or program a byte in it's own code space area.

CANopen Bootloader for AT90CAN128

The Embedded Systems Academy has implemented a CANopen bootloader for the Atmel AT90CAN128 microcontroller, written in C. HYPERLINK

Since only the SDO server needs to be implemented for bootloader mode, only two out of the 15 Message Object Buffers are used.

The bootloader flash area of the AT90CAN128 can be configured from 0 up to 8Kbytes. The bootloader fits comfortably into this space, using only little more than half of this memory and leaving enough room for a secondary bootloader or functional extensions. The bootloader flash area can be hardware-protected using an AVR fuse, further improving overwrite protection beyond the software mechanism outlined above. Also, the movable interrupt vector base implemented in the AT90CAN128 is supported to guarantee safe and reliable operation at all times.

The fact that the AVR allows a program in the boot flash area to run while the application flash memory is written greatly simplifies the programming process.

The bootloader and checksumming tool understand and process extended hex files (HEX86, type 2) needed for the larger address space of the AVR architecture with up to 128kBytes and typically generated by all compilers for the AVR.

It should be noted that with 128KB flash, one can have a DSP-302 bootloader of 8KB flash and 2 flash memory areas of 60KB each able to contain 2 different version of the user software. One can download an new update and save the previous one if needed.

Conclusion

In conclusion, CANopen DSP-302 based reflashing is possible in embedded applications with flash microcontrollers. The boot program can use the CAN interface to download the application program in the application flash memory. The CANopen standard provides the framework to make this happen with a wide variety of tools, or through an embedded CANopen Master/Manager node with an off-the-shelf CANopen stack with SDO client. The AT90CAN128, with its powerful bootloader features and strong CAN support, is an ideal candidate for CANopen nodes that need to be updated in the field, with maximum reliability. Embedded Systems Academy's implementation demonstrates how this can be achieved efficiently and with minimal resource requirements.

Damien Grolleau
Atmel
BP70602 Nantes 44306 France
Phone +33 2 40 18 16 11
Fax +33 2 40 18 19 60
damien.grolleau@atmel.com
www.atmel.com

Christian Keydel
Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
(408) 910-8418
ckeydel@esacademy.com
www.esacademy.com
www.microcanopen.com/