LIN-Based Bootloader Implementation on ATA6616/17

Tiantian, Shi, and George Gong

Designers can program a microcontroller with an external chip or they can have the microcontroller program itself using a bootloader program contained in its own memory. External programming adds cost, requires long programming times, and must have the proper interconnections to the target MCU. The chip's programming ports may not be available when it has been integrated into a system module. Alternatively, a bootloader program can update the module's firmware by loading the new code via a standard serial port.

The designer first loads the bootloader program via conventional means, perhaps before the MCU chip is soldered into the module. Then the chip can reprogram its remaining Flash program memory over its LIN, CAN, UART, or TWI interface. The chip can do this even after the system is deployed to the end user. Bootloaders implement "in-system programming" (ISP). This means that the user can program or re-program the microcontroller on-chip Flash memory without removing the device from the system and without the need of an external programmer chip or system. The scope of this article is the implementation of a LIN-based bootloader.

The Automotive LIN Bus

Electronic modules have improved the comfort, safety, and fuel economy of today's vehicles. Manufacturers have developed different bus network standards to guarantee that these modules can communicate at the required speed and safety levels. Buses such as CAN (Controller Area Network), and LIN (Local Interconnect Network) minimize cost while maximizing the performance. LIN was developed as the low-cost and low-speed complement to CAN. Engineers use LIN primarily in comfort applications. This allows them to interconnect an increasing number of comfort functions. More and more companies provide LIN-related product since the LIN bus is now accepted on all new development platforms.

Atmel® is one of the most successful LIN-related product providers. We offer a modular LIN2.0/2.1 family with products at all integration levels. The products range from simple transceiver ICs to complex system basis chips (SBC). At higher integration levels, Atmel provides complete System-in-Package (SIP) solutions. A single package includes an Atmel AVR® MCU, a LIN transceiver, a voltage regulator, and a watchdog timer (figure 1). The ATA6616/17 is a complete LIN bus node application. The part integrates an ATA6624 LIN SBC die with an AVR ATtiny87/167 MCU into a 5mm x 7mm QFN package.

Once you integrate a LIN SIP into a module to make a LIN node, it becomes problematic to upgrade the firmware. Unless you unsolder the SIP IC, there is no possibility of accessing the conventional programming ports with a hardware programmer. Since the module is part of a LIN network, it is natural to think of operating a bootloader through the LIN bus.

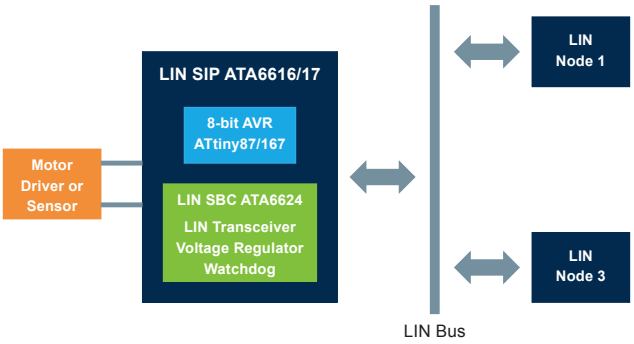

Figure 1. The ATA6616/17 LIN System-in-Package

Bootloader Implementation

The MCU integrated into the ATA6616/17 lacks a separate bootloader section in program memory. To bootload an ATA6616/17, you use the LIN interface of the LIN SBC die with its associated protocol to download the program code. The self-programming mechanism of the MCU writes the updated code into the program memory.

Designers initially connect to the LIN module with a PC and converter device (see figure 2). The PC has AVR MCU programmer software installed. The AVR Open Source Programmer (AVROSP) is a good option. The hardware converter changes the UART signals coming out the USB interface to the LIN standard. The converter device does communication with the SIP as a LIN slave, and delivers the bootloading commands as boot master. It can also deliver real-time operation commands as a LIN master.

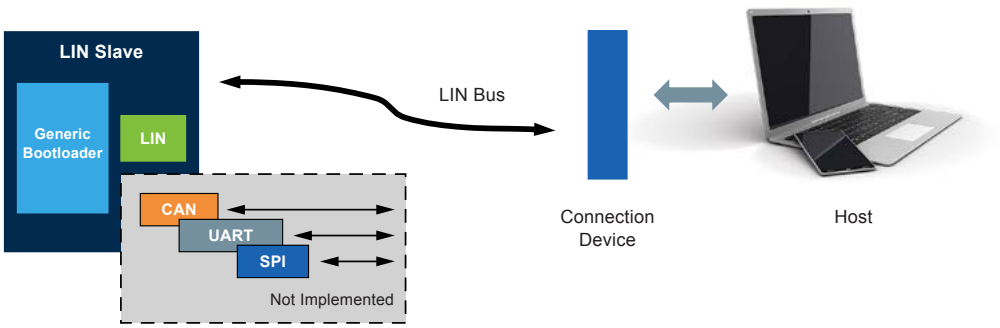

Figure 2. LIN Bootloading Physical Environment

You use the ISP programming interface to load the bootloader code into Flash memory. Then the bootloader can be used to update the application Flash section in the future without using ISP. The bootloader has the ability to read, erase, and write the flash memory. Additionally, it supports a command jump from the bootloader to the application code.

The bootloader uses the SPM (Store Program Memory) instruction of the ATA6616/17. The bootloader updates the program memory in a page-by-page fashion. The Store Program Memory Control and Status Register (SPMCSR) manages the SPM operations (figure 3).

Bit	7	6	5	4	3	2	1	0	
	-	RWWSB	SIGRD	CTPB	RFLB	PGWRT	PGERS	SPMEN	SPMCSR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 3. The Store Program Memory Control and Status Register

Only the combinations "10 0001 $_b$", "01 0001 $_b$", "00 1001 $_b$", "00 0101 $_b$", "00 0011 $_b$", or "00 0001 $_b$" in the lower six bits will have an effect. You employ the Z register to address the operation targets of the SPM commands, such as page erase, page write, or instruction write (figure 4). Here the MSBs (the Most Significant Bits) are used for addressing the pages. The LSBs (the Least Significant Bits) address the words within the page. A more detailed description of the SPM instructions and applications is available in Atmel application notes.

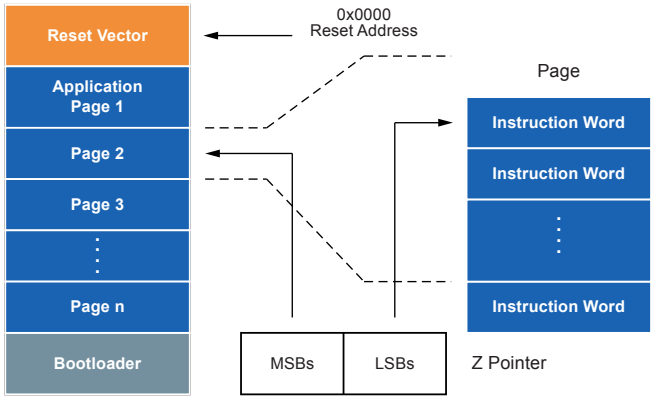

Figure 4. Flash Organization and Z Pointer Addressing

The bootloader works with the help of SPM (figure 5). There is only one entry point to the bootloader. Unlike some MCUs, this integrated MCU has no special boot Flash section. It is not possible to enter the bootloader by setting the fuses. For this reason the initial programming should load and organize the program memory according to the scheme in figure 4. You define the reset vector to always initiate the bootloader program. Figure 5 shows this as the "boot process", which occurs after a reset event. The boot process checks and decides whether the configured boot section starts executing or the program in the application section starts executing. The bootloader has separate commands to write to the application section and to execute a jump into the section. If the jump to the application section is not performed, the application will never execute, even if the application section is programmed. The "protocol identification" of the bootloader selects what protocol to use. This could be LIN, CAN, a UART, or other protocols. The first confirmed communication on the network starts the initialization of the corresponding peripheral.

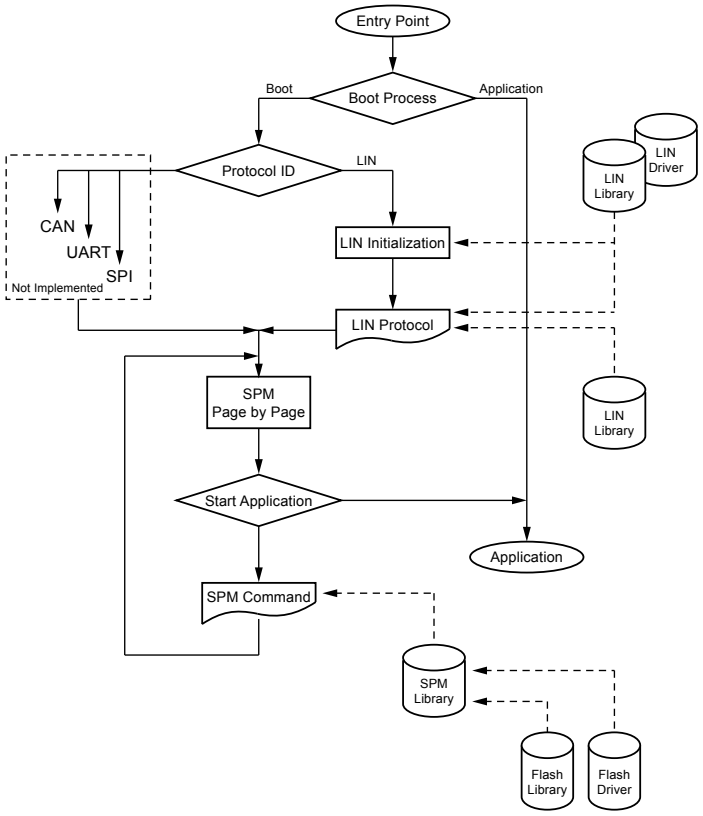

Figure 5. Bootloader Flow Chart

The LIN bootloader application performs an initialization process after each device reset. The LIN protocol is a high-level protocol, which is described in the LIN standard documents. The host initiates the communication by sending 0x55 as a synchronization character to help the slave LIN SIP to find the baud rate. At the end of this character the bootloader should have its LIN initialization done. The LIN protocol decodes the incoming commands. If the command received is to update the Flash, the received data is written into the program memory as if it was written by a programmer device. The bootloader performs the SPM instructions to update the Flash on a page-by-page fashion.

Once the entire incoming data stream has been written into program memory you send a command to execute the application code. The output from the bootloader jumps to the first instruction of the application program. The bootloader erases a page and then writes the incoming data into the page (figure 6). There is also a Read – Modify – Write operation. This is suitable for updating small parts of the Flash, such as a constant string.

Conclusion

This article highlights the bootloader implementation on an Atmel ATA6616/17 LIN chip. The bootloader application lets you update firmware in the field through the LIN bus. This is done without using the traditional programming ports. More detailed information is available at the Atmel website and corresponding AVR application notes.

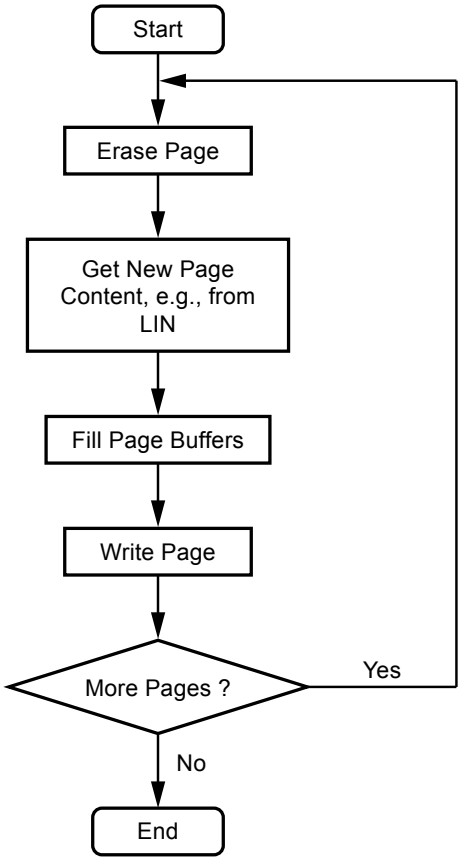

Figure 6. Typical Update Flowchart