

Eversmith

AVR-SMS © 2003 Martin Thomas
Version 0.56a, Sept. 2003

Overview

This is a firmware for ATMEL (www.atmel.com) ATmega microcontrollers (and maybe "Classic"-AVRs) to send and receive short messages in a GSM Network (SMS) with a GSM Mobile Phone/Cell Phone/Handy connected to the AVR serial port. The firmware is completely written in AVR-Assembler Code. (You may call it: YAAS - Yet Another AVR SMS Software - there are other projects out there.)

Features:

- Sends and receives Short Message in SMS PDU-Format. PDU creation and decoding is done on the fly, so only one memory area is needed that can be shared.
- Switches output on receive of SMS.
- Send status on demand.
- Sends SMS if input pins where enabled.
- Menu-based configuration - no special software needed.
- Configuration values are kept in EEPROM storage.
- Debouncing of input keys.
- Buffered UART
- Compact code, but sometimes cryptic. (It's Assembler and my first AVR-Project, 16 years after I've left the Commodore 64 and 6510 Assembler)

Required Hardware

- ATMEL ATmega16 (1)
- A GSM Modem or Mobile/Cell-Phone/Handy (ME) that supports AT-commands on a serial bus and the SMS PDU-Mode according to standard GSM 07.05 (2)
- Connection between ME and AVR UART (3)
- Voltage Level converter for USART (like MAX232) for PC connection during configuration (3)
- 8 MHz External Quartz Oscillator (4)
- Input Connections to PORTA (active LOW = ON is grounded) (5)
- Output Connections to PORTC (active LOW = ON is pulled low, controller "sinks" source) (5)

Hardware-Remarks:

- (1) The Software should work with all ATmegas. But was only tested with an ATmega16. An AVR AT90S8515 was used for the first developments but has not been tested with later versions. If you intend to use classic AVRs keep in mind that the UART might act different than an USART and the assembler directive `call` is not supported on most (all?) of the old devices. Memory demand is not that high, the firmware should fit in "small" devices. RAM demand may be critical. Not because of the PDU handling which is done on the fly but the large UART input buffer that is needed to receive a delivered SMS-PDU from the GSM network since no handshaking is done on the serial interface.
- (2) Firmware has been tested with Siemens TC35 Terminal "cellular engine" (GSM-Modem) and Siemens S35 mobile phone (www.siemens.com). Nokia and Ericsson devices should work as

long as AT-Commands are supported. Most Ericsson MEs seem to support AT-Commands. The "cheap" Nokia MEs are equipped with a serial port but use a special protocol called F-BUS and/or M-BUS which is not supported by Eversmith (although the PDU part seems to be identical). The "professional" series Nokias usually support AT-Commands. (And yes, I know that the TC35 can handle SMS in text-mode. But who wants to go the easy way?)

- (3) The only "tricky" part in the controller-circuit is the serial-port (UART) connection. The controller should act as a Terminal (TE or DTE) when connected to the modem. So use a "Male" D-Sub plug on your controller board to avoid confusion. During set-up a connection to the PC is needed. This can be done by using a so-called Null-Modem-Cable with "female" plugs on both ends (wiring: 5-5 2-3 3-2 on D-Sub 9). To connect the AVR to the modem use a normal cable with "male" and "female" plugs and straight wiring (5-5,2-2,3-3 on D-Sub 9). If you use a mobile phone you may want to connect the AVR controllers TTL-level pins directly to the TTL-Level-Pins of the Phone without using the level-shifter. In the current version neither hardware nor software handshaking is supported. This may change in further releases.
- (4) The delays and interrupts are calibrated to this frequency. With minor changes every frequency should be possible as long as the UART works reliably.
- (5) Off course the ports and pins for in- and output can be changed in the source. But changing from "inverse" logic to "positive" logic may lead to a lot of changes

Build the Firmware ("Compilation")

The firmware was developed with AVRStudio Version 4.07 (Build 240), which you can download from ATMEL (www.atmel.com). Other Assemblers/Environments should work but have not been tested. Copy all Assembler Files together in one directory. Create a new project and import the main-file `avrsmc.asm` into it. Set debug-platform and device to AVR simulator and ATmega16 (you own better debugging hardware? – you know better than me how to use it). Create the hex-File and upload it with your favourite tool to the Controller. I used Ponyprog 2000 (available from www.LancOS.com) and a STK200 compatible programming circuit for serial SPI programming. The programming software AVRDUDE (<http://savannah.gnu.org/projects/avrdude>) has also been tested with success (much faster than PonyProg). A step-by-step instruction is given in the appendix.

Configuration

Hardware

GSM-Modem: According to the ME documentations I have read no extra set-up is needed on the ME. BUT: So far a PIN-code handling for the SIM-card in the GSM-device (ME) is not implemented. So disable "security" or "Ask for PIN" on the SIM. This can be done with any mobile phone (see the phone manual) or by sending the AT-Command `AT+CLCK="SC",0,1234` to the ME where 1234 is your SIM-Card-PIN. PIN handling is on the TODO-list if the software proves to be stable enough (I did not want to enter PUKs every time the PIN input failed three times). You also need to know the modems serial port parameters. Read the documentation and do some testing by sending AT-Commands (just "AT" is ok) from a PC Terminal-Program to the GSM-modem. Start with 19200 baud, 8 data-bits, 1 stop-bit, no parity, and no handshaking, if this fails try 9600,8,n,1. The Terminal by Br@y is a very useful tool for this kind of tests (<http://bray.velenje.cx/avr/terminal>). Some MEs offer baud-rate auto-detection or AT-commands to change the baud-rates, refer to the ME manual. If the phone does not support 19200,8,n,1 you have to adjust the Baudrate in the file `globals.inc` and recompile. A Siemens TC35 Terminal works "out of the box" in factory default settings.

Controller: Power down the controller. Connect a PC with your controller-board using a Null-Modem-Cable (wires 2 and 3 crossed) and start a terminal program. Hyperterm that comes with

MS-Windows will do. The default settings of the AVR-U(S)ART are 19200 baud, 8 data-bits, 1 stop-bit, no parity and no handshaking. You can change the parameters in the files `globals.inc` (Baud-Rate) and `UART.asm` (Mode)). Set them to the parameters supported by the ME, since Configuration from PC and Connection to the ME is done on the same port with the same parameters. Now apply power to the controller board and connect Pin PA7 (PDIP33) to ground to get the configuration menu. Continue with the software-configuration.

Software Configuration (Parameters)

A menu will come up on your terminal screen by grounding (pull low) Pin PA7:

```
*** AVR-SMS Main-Menue ***
(1) Enter Message 1
(2) Enter Number 1
(3) Enter Passcode
(4) Select ME
(5) Show Config
(0) Exit
```

If the menu does not appear double-check your wiring and the UART settings.

Select 1 and see the prompt:

"Message-Input :"

Enter something like:

Example: Foos Eversmith Switched :

Add a colon and a space char at the end of the input just for a nicer output. Backspace is supported for editing to delete the last character. Hit the return-key to save the Message into the controllers EEPROM. The AVR will send this text with all messages that will be send when an input has been enabled. So if PA0 (PDIP40) is grounded you will get the message: "Foos Eversmith Switched : 1". Avoid special characters; so far there is no translation between ASCII and SMS char-code in the software. For the usual characters the codes are identical (see GSM 03.38 alphabet tables).

Select 2 and get:

"Number-Input: "

Enter the number to which SMS messages should be sent. At the moment only one destination address is supported. You must enter the number in INTERNATIONAL format. Which means if a messages should be send i.e. to a German D1-Telekom (or T-D1 or T-Mobile) phone with the national number 0171-987654321. The international access code for Germany is 49 so the destination number has to be entered like this:

Example: 49171987654321

Select 3 and get

"SMS-Receive-Passcode-Input: "

This is the pass-code for incoming SMS messages (SMS-deliver). Every SMS that should be processed by the controller must start with this 4-digit-code. The default is 9999 if the EEPROM-Storage of the passcode was empty (\$FF) (i.e. after firmware-flash and/or chip erase). Everyone who knows the PIN can send commands to the AVR - so be careful. The sender-number is also decoded from the PDU during receive and could be used as identification with minor code modifications but this limits the operation of Eversmith to one sender-SIM-card.

Example: 1234

Select 4 to get the ME selection menu:

```
-- select ME Type --
(1) Siemens (AT-Commands)
(2) Ericsson (AT-Commands)
(3) Nokia (AT-Commands)

(0) Leave unchanged
```

Select your ME. The only difference between Siemens and others is the "wait for prompt" after AT+GMGS. Nokias and Ericssons do not send a prompt (according to the manuals, no real-life test). The default after chip-erase is Siemens (1). Choose your device by typing the number, the selection will be stored in EEPROM. Example: "1" (=Siemens).

Select 5 to verify your input. From the Example Input you should get:

```
Message is :Foos Eversmith Switched:
Number is  :4917198765421
SMS-Receive-Passcode is :1234
ME : Siemens (AT-Commands)
```

Selections 6-9 are for debugging only and can change anytime. Select 0 to exit the Menu and to start the normal operation. Unplug the PC from the controller board and connect your GSM modem (remind the different wiring). If you need some time (more than 30 seconds) to change the cables you should turn off controller and modem, connect the modem, turn on the modem and finally power up the controller board. After 30 seconds of normal operation the controller will start to ask the modem if new messages have been received.

Operation

Some Port-Pins will not be handled by the Eversmith routines. So you can use them for other functionality. Take a look in the files `keys.asm` and `switches.asm`, there are masks mentioned. Set the masks to 0b11111111 and everything is under control of Eversmith.

Incoming (Outside World to AVR)

Every 30 seconds (at default clock and timer-settings) the AVR starts to "poll" the ME. This is done by issuing the command to transfer every single message in the MEs SMS memory to the ME serial port. If no GSM-Modem is connected to the AVR the system will step from one receive timeout to another which may cause delays. If a new message has been received (SMS-Deliver Unread) it gets parsed on the fly from the PDU format send by the ME to text format which is stored in SRAM.

Then first four characters of the received text will be checked against the 4-digit SMS-Receive-Passcode given during configuration. If the codes are not equal (or not given) the SMS message will not be processed by the AVR. If code has been verified the parser processed the message and looks for keywords and parameters. The internal parser knows the following "keywords":

S = send (S)tatus

The AVR will answer to this request with an SMS to the number given during configuration. The Message consist of the Text: "Eversmith Status Outp.: 0cccc7 Inp.:0aaaaa7". Where 0c..c7 is the status of the output port (default PORTC) as a bit field with zeros and ones and 0aaaaa7 is the state of the input port (default PINA).

Example: Send a Message: "1234 S" to the ME connected to the AVR. The controller reads the message, verifies the pass code and issues a process request. The controller transmits the SMS i.e. "Eversmith Status Outp.: 10001000 Inp.:11000000" which informs that the outputs PC0 and PC4 are enabled (PORTCx=0) and the inputs PA0 and PA1 are grounded (PINAx=0).

L = (L)og-Level (for sending Status)

Not completely implemented yet

N = Switch o(N)

All pins with 1 in Mask (LSB first) will be turned on. Not implemented yet

F = Switch o(F)f

All pins with 1 in Mask (LSB first) will be turned off. Not implemented yet

A = Switch (A)ll Pins on

Switches all pins of the output port on (PORTC=\$00); remind we "sink" on the port.

Example: "1234 A" and all your LEDs or relays or whatever should switch on (if "active low").

Z = Switch all Pins off

Same as "A" but off (PORTC=\$FF).

P = Switch one (P)in

Takes two parameters and switches an output pin. The first Parameter is the output-pin number starting with 1 for PIN0. The second Parameter is 1 for on (sink) or 0 for off. Example 1: sending "12345 P 2 1" turns on output-pin PC3 (PORTC=xxxx0xxx). Example 2: "12345 P 2 0" turns off output-pin PC3 (PORTC=xxxx1xxx).

Remarks: With A, Z and P only those Pins will be switched, that are not masked out in `swi tches.asm` ("managed outputs"). The polling system deletes every message in the MEs (default) Message Memory after processing. So messages that have been on a SIM-card will be lost after the card was used in an Eversmith-controlled-device when the MEs SMS message memory was set to SIM which is default on most MEs.

Outgoing (AVR to Outside World)

If you connect a Pin on PORTA that is not masked out by the key-mask in `keys.asm` the AVR will send a Message. For the pins PA0-PA3 the message informs you that the pin was grounded (switched on). The message consists of the message-text given during configuration followed by the Input PIN that has been closed. Numbering starts with 1 for PIN0. Example: switch on PINA1 (ground the pin) and the AVR will issue the modem to send a Message: "Foos Eversmith Switched : 2" to the number you gave during configuration.

If pin PA4 gets grounded a Message from RAM is sent, the default text for this message is copied from the program flash. This should give an example how to send messages that were build in RAM.

Remark: In the current configuration only PINA0-PINA4 will init a message. PINA7 is reserved for entering the configuration-menu (see also section Bootloader in the Appendix)

License

© (2003) Martin Thomas, Kaiserslautern, Germany

This Software is copyrighted and distributed under the Aladdin Free Public License (AFPL). Please respect the license. It is only free for **non-commercial** use. You can play around as much as you like, change the code and give it away for free to everyone as long as you give the complete package of Eversmith (source-files, license-text and documents) together with the **source-code** of your derived work. If you want to sell software or hardware (programmed Controllers) that uses Eversmith routines, modifications of Eversmith or Eversmith itself you have to contact me (Martin Thomas) for a commercial license (e-mail-address further down in this text). I hope the Aladdin license was a good choice. Eversmith was made in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of FITNESS FOR A PARTICULAR PURPOSE.

In case the AFPL is not valid in your country: Eversmith is copyrighted, non-commercial use is free. Commercial use of Eversmith, modifications of Eversmith or parts of Eversmith on any kind of media (CD, Flash-Memory etc.) only with written permission ("commercial license") of the author.

Remarks and tips:

- Never again such a project in assembler without good debugging equipment. At least a second hard- or software UART. An ATmega162 or ATmega128 with two UARTS would be the better device for such a project; one UART for configuration end debugging and one UART for Modem connection. JTAG-based debugging hardware is on the wish-list. WINAVR (gcc et al) is already installed – back to the world of "high level" languages.
- Serial communication without handshaking is not the real thing. I had to increase the input buffer size to fit the incoming PDU length. That's a waste of SRAM. At least Software-Handshaking (XON/XOFF) is desirable. News in Version 0.56: the main problem were the delays in the receive function. After some improvements everything works fine at 19200 Baud without handshaking and a small receive buffer of 20 chars.
- rcalls do not always work as expected and the AVRStudio-Assembler does not warn about this in some settings. If you see strange behaviour in your software select Menu Project/AVR Assembler Settings and disable the option "wrap relative jumps", recompile and read the output carefully. This issue took me 4-5 hours of debugging: A reset occurred just before the call of an Eeprom write routine (the same method was used several times before). I searched a long time in and around the EEPROM-routines and thought about side-effects caused by interrupts. Finally the issue has been solved by replacing some rcalls with calls (and loosing compatibility to the "classic" AVRs). So avoid rcalls and use calls when possible.
- The PDU-format for SMS is a little strange but allows full control over the GSM-SMS handling in a compact structure. Prototyping with a "real" computer in C helped a lot.
- Serial programming is too slow - need to buy a STK500. AVRDUDE seems to be much faster than PonyProg.
- ...

Credits:

for Code-Snippets:

- Jack Tidwell (UART FIFO-code is based on his code)

for Tools:

- [Br@y](#) ([Br@y](#) Terminal)
- Claudio Lanconelli (PonyProg2000 Programmer)
- Brian S. Dean (AVRDUDE Programmer)
- Stefan Frings (smstools, used for debugging the c prototypes of the send-routine)
- ATMEL (for the free AVR Studio, and of cause for the "nice" devices they build)
- T. Kloppenburg (Delay-Loop-Generator, used for debugging, no more delays in code)

for Documents:

- SIEMENS ("SMS with the SMS PDU-mode" document)

for Support:

- Herr Lösch at ZBT Elektronik/Technical University of Kaiserslautern for rescuing my development ATmega with his parallel programmer since I locked myself out by writing the wrong fuse-bits.

Contact

Suggestions, error-reports, every kind of feedback welcome...

e-mail: eversmith@heizung-thomas.de

Change log

23. Sept. 2003	Initial Public Release to the AVRFREAKS web-site as Version 0.54
24. Sept. 2003	Started Version 0.55, Minor Corrections in the manual. Bootloader-code was included in V 0.54 – fixed text passages saying it would be included in a later release. Tested AVRDUDE programming software and added it to the manual
25. Sept. 2003	More corrections in the manual, added a step-by-step instruction, more licensing information, added some more licensing text to the source code
26. Sept. 2003	Release of Version 0.55 – changes in documentation and licensing, no new functions
27. Sept. 2003	Started Version 0.56 – code cleanup. New file globals.inc with main configuration parameters (clock, baud)
29. Sept. 2003	(a) Cleaner implementation of the timer0 interrupt, now easier to adopt to other clock-rates. (b) Improved UART Receive by handling the timeouts via timer-interrupt (no delay loops anymore). Seems to work very good with a small receive buffer (20 chars) at 19200, so implementation of handshaking may not be needed any more – needs testing. (3) Added ME selection for Siemens, Nokia and Ericsson in Menu, still only AT-Command devices (GSM07.05), only difference in code is the prompt handling after AT+CMGS.
29. Sept. 2003	Release of Version 0.56
30. Sept. 2003	Started Version 0.57
11. Oct. 2003	More text in the Manual, since there has been confusion about the licensing. Interims release 0.56a

Appendix

Step-by-step instruction to build and program the firmware

This is a step-by-step instruction for AVRStudio 4 and Ponyprog2000 with STK200 compatible programming hardware. It's a beginner's guide that hopefully covers all needed steps (Please don't be annoyed if something is too trivial for you):

- Download AVRStudio and Ponyprog2000 and install the software on your system.
- Extract the Eversmith distribution archive (zip-File) to c:\eversmith_x_xx (replace x_xx with the version number). Rename the directory c:\eversmith_x_xx to c:\eversmith.
- The assembler source files (*.asm) should now be located in the directory c:\eversmith\src.
- Start AVRStudio4. If a dialog "Welcome to AVR Studio 4" appears: cancel it.
- In AVRStudio select Project/New Project. Select: Project Type: Atmel AVR Assembler, Project Name: eversmith, uncheck Create initial File, uncheck Create folder (both, initial file and folder are already there). Type avrsm in the Initial File text-field. At Location click the [...] button and browse to the folder c:\eversmith\src. Click on the [Next>>] button. Select Debug Platform: AVR Simulator, Device ATmega16. Click the [Finish] Button.
- Select Project/Add existing file from the main-menu. In the open dialog select avrsm.asm.
- Select File/Open file and open the file globals.inc, change the value in the line .equ clock to the value of your crystal frequency in the unit Hz (1 MHz = 1000000 Hz).
- Select Project/AVR Assembler Setup and verify that Additional Output file format is set to Intel intellec 8/MDS (Intel hex).
- Select Project/Build. The last line in the Output should be: "Assembly complete with no errors."
- Exit AVRStudio
- Start PonyProg2000. Skip the initial dialogs about set-up and calibration if any.
- Select Setup/Interface Setup: For the STK200 compatible programmer select Parallel, AVR ISP I/O, leave all polarity checkboxes unchecked and click on [OK]. Select/Calibration from the main menu and click [YES].
- Select Device/AVR micro/ATmega16 from the main menu (just to be sure, it should be auto-detected)
- Select File/Open device file and browse to c:\eversmith\src\eversmith.hex to load the firmware into Ponyprog.
- Select Command/Write All to program the firmware on the ATmega16.
- At this point the firmware is on the chip and running. If you are using the internal oscillator (factory default for new ATmegs) and the clock in UART.asm is adjusted to the internal oscillators frequency you can go on with the configuration explained in the next section.
- If you are using an external crystal or another external clock source you have to set the fuse-bits. Please read the manual for your controller available from ATMEL. You may loose access to the controller if you program wrong fuse values and have to buy a parallel programmer (e.g. STK500) or find someone who owns one. OK, enough warnings, here is what I

have done: Verify the right device is selected on PonyProg, Select Command/Security and Configuration bits. Click button [READ] to read the current fuse settings. Uncheck SUT1, SUT0, CKSEL3, CKSEL2, CKSEL1 and CKSEL0. Click the [Write] Button. Click [OK] to close the dialog. Reset the controller by selecting Command/Reset. Go on with the configuration.

Bootloader

A bootloader is included in the distribution archive (subdirectory bootloader). So you can upload the Eversmith firmware via the UART connection without extra hardware once you have the bootloader on the chip. See the original documentation from the bootloader developer Mariano Barrón Ruiz that comes with the Eversmith package. AVRProg from the AVRStudio Tools menu can interact with the bootloader to upload firmware like Eversmith.

If the bootloader is flashed and the boot-fuses are set to access the bootloader on reset (BOOTRST) then Pin PA7 enables the bootloader when the pin is connected to GND (pulled low) during power-on or another reset condition. The AVR enters bootloader-mode and the Eversmith configuration menu will not appear even if you toggle PA7. In this case release pin PA7, reset the controller and ground PA7 afterwards to enter the menu.

The bootloader is *not covered* by the AFPL license since it has been developed by Mariano Barrón Ruiz who did not mention any kind of license in his document. I only implemented some minor modifications. See the bootloader as a goodie so you don't have to search the net. *It's not a part of Eversmith*, just a "convenience" tool.