AVR Freaks

DESIGN NOTE #032

AUTHOR: MARIANO BARRÓN RUIZ – ISPBARUM@SB.EHU.ES

KEYWORDS: BOOT LOADER, SPM, SELF-PROGRAMMING

AVR Boot Loader

Introduction

This document presents a software written in assembly for Self-Programming all AVR microcontrollers with Boot Loader Flash Section. The program uses the USART serial bus interface to read, program, or verify either the EEPROM or the Flash memory.

Overview

The ATMEL application note “AVR109: Self-Programming” describes a Boot Loader written in C and assembly. This software needs the IAR C compiler to generate the object code. The program is size optimized to 504 bytes.

The Boot Loader presented in this document has been written in assembly, so you only need the AVR Studio® to generate the object code. The program fills 362 bytes and supports the same commands as the AVR109 Boot Loader.

The Boot loader, see the bold lines in source code, can use the EEPROM two last positions to count the reprogramming cycles of Flash memory (Flash endurance: 1,000 write/erase cycles). On each Chip Erase, the Boot Loader increments the Chip Erase Counter located at EEPROM addresses: Last (counter High Byte) and last-1 (counter Low Byte). In this case the code size increments to 392 bytes. On programming the EEPROM, user must take care not to overwrite the EEPROM two last positions.

The Boot Loader starts by checking if programming is to be done, or if the user program in the Application Flash Section is to be executed. If a selected pin (PC0) is pulled low during Reset, Programming mode is entered. If this pin is high, normal execution is done from address $0000 as if an ordinary reset had occurred. After a Reset, the I/O ports are configured as input ports with internal pull-up disabled, so the selected pin should be pulled high by an external pull-up resistor.

In Programming mode, the Boot Loader receives commands form AVRprog via the UART. AVRprog is a part of AVR Studio® software but also you can get it separately. The communication uses 19,200 bps, 8N1 (8 data bits, no parity bits and one stop bit). Each command executes an associated task Any command not recognized by the Boot Loader results in a “?” being sent back to AVRprog.

Special Considerations

The Boot Loader has been written for an ATmega163 with 7.3728 MHz clock. The first time you must program the Boot Loader code into the AVR Boot Flash Section using an external programmer. Next times, the Boot Loader enables Flash and EEPROM programming via de UART with a PC running the AVRProg programming software.

To avoid the user can unadvisedly protect the Application Flash Section from software updates, the Boot Loader supports neither Fuse bits nor Lock bits programming.

After programming a new ATmega163 with this Boot Loader you must program Fuse and Lock bits as follows:

- The ATmega163 Fuse High bits must be programmed to 0x04 in order to configure the Boot size to 256 words and to move the Reset Vector to word address 0x1f00.
- The Lock bits must be programmed to 0xEF to protect the Boot Flash Section from unintentional changes and to allow accessing the Application Flash Section.
- The Fuse bits must be programmed before programming the Lock bits.

Source Code

```
;********** B O O T  L O A D E R  F O R  A T m e g a 1 6 3 **********
;* File                              : AVRBoot2.asm (Include chip erase counter)
;* Version                           : 1.2
;* Compiler                          : AVR Studio
;* Target                            : ATmega163
;* Output size                       : 392 bytes

 .equ DT      = 0x66                  ; Device Type = 0x66 (ATmega163)
 .equ SB1     = 0x02                  ; Signature byte 1
 .equ SB2     = 0x94                  ; Signature byte 2
 .equ SB3     = 0x1e                  ; Signature byte 3
 .equ UBR     = 23                    ; Baud rate = 19.200 bps with fCK = 7.3728 MHz

 .INCLUDE "m163def.inc"               ;Include Register/Bit Definitions for the mega163


.org  SECONDBOOTSTART                 ; ($1F00) second boot. Block size is 512B

        sbic PINC,PINC0               ; Skip next instruction if PINC0 cleared
        rjmp FLASHEND+1               ; else normal execution from Reset (FLASHEND+1 = Address 0000)

; Programming mode
        ldi R24,low(RAMEND)
        ldi R25,high(RAMEND)
        out SPL,R24
        out SPH,R25                   ; SP = RAMEND

        ldi R24,UBR                   ; Baud rate = 19.200 bps
        out UBRR,R24
        ldi R24,(1<<RXEN)|(1<<TXEN)
        out UCSRB,R24                 ; Enable receiver & transmitter, 8-bit mode
```

```
L10:    rcall uartGet                   ; repeat (R16 = uartGet)
        cpi R16,27                      ; while  (R16 == ESCAPE)
        breq L10

        cpi R16,'a'                     ; if(R16=='a') 'a' = Autoincrement?
        brne L12
        ldi R16,'Y'                     ; Yes, autoincrement is quicker
        rjmp L70                        ; uartSend(R16)

L12:    cpi R16,'A'                     ; else if(R16=='A') write address
        brne L14
        rcall uartGet
        mov R27,R16                     ; R27 = address high byte
        rcall uartGet
        mov R26,R16                     ; R26 = address low byte
        lsl R26                         ; address=address<<1
        rol R27                         ; convert from byte address to word address
        rjmp L68                        ; uartSend('\r')

L14:    cpi R16,'c'                     ;else if(R16=='c') write program memory, low byte
        brne L16
        rcall uartGet
        mov R22,R16                     ; R22 = data low byte
        rjmp L68                        ; uartSend('\r')

L16:    cpi R16,'C'                     ;else if(R16=='C') write program memory,high byte
        brne L18
        rcall uartGet
        mov R23,R16                     ; R23 = data high byte
        movw R30,R26                    ; Z pointer = address
        movw R0,R22                     ; R0&R1 = data
        ldi R24,1                       ; SPMCR = 0x01
        out SPMCR,R24                   ; page load (fill temporary buffer)
        spm                             ; Store program memory
        adiw R26,2                      ; address=address+2
        rjmp L68                        ; uartSend('\r')

L18:    cpi R16,'e'                     ; else if(R16=='e') Chip erase
        brne L28
; for(address=0; address < (2*SECONDBOOTSTART); address += (2*PAGESIZE))
        clr R26                         ; page_erase();
        clr R27
        rjmp L24
L20:    movw R30,R26                    ; Z-pointer = address
        ldi R24,3                       ; SPMCR = 0x03
        out SPMCR,R24                   ; page_erase
        spm                             ; Store program memory
        nop
```

```
        subi R26,low(-2*PAGESIZE)       ; address += (2*PAGESIZE)
        sbci R27,high(-2*PAGESIZE)
L24:    ldi R24,low(2*SECONDBOOTSTART)
        ldi R25,high(2*SECONDBOOTSTART)
        cp R26,R24                      ;address < Boot Flash address(byte address) 0x3E00 ?
        cpc R27,R25
        brlo L20

        ldi R26,low(E2END-1)                ; increment Chip Erase Counter located
        ldi R27,high(E2END-1)               ; at address E2END-1
        movw R22,R26                        ; Save Chip Erase Counter Address in R22
        ldi R17,1                           ; read EEPROM
        rcall EepromTalk
        mov R24,R16                         ; R24 = Chip Erase Counter low byte
        rcall EepromTalk
        mov R25,R16                         ; R25 = Chip Erase Counter high byte
        adiw R24,1                          ; counter ++
        out EEDR,R24                        ; EEDR = R24 Chip Erase Counter low byte
        movw R26,R22                        ; R26 = Chip Erase Counter Address
        ldi R17,6                           ; write EEPROM
        rcall EepromTalk
        out EEDR,R25                        ; EEDR = R25 Chip Erase Counter high byte
        rcall EepromTalk

        rjmp L68                            ; uartSend('\r')

L28:    cpi R16,'m'                         ; else if(R16== 'm')  Write page
        brne L34
        movw R30,R26                        ; Z-pointer = address
        ldi R24,5                           ; SPMCR = 0x05 Write page
        out SPMCR,R24
        spm                                 ; Store program memory
        nop
L32:    rjmp L68                            ; uartSend('\r')

L34:    cpi R16,'P'                         ; else if(R16=='P') Enter programming mode
        breq L32                            ; uartSend('\r')
        cpi R16,'L'                         ; else if(R16=='L') Leave programming mode
        breq L32                            ; uartSend('\r')

        cpi R16,'p'                         ; else if (R16=='p') Return programmer type
        brne L38
        ldi R16,'S'                         ; uartSend('S')  Serial
        rjmp L70                            ; uartSend(R16)

L38:    cpi R16,'R'                         ; else if(R16=='R') Read program memory
        brne L40
        movw R30,R26                        ; Z-pointer <= address
```

```
        lpm R24,Z+                      ; read program memory LSB; store LSB in R24 and Z pointer ++
        lpm R16,Z+                      ; read program memory MSB; store MSB in R16 and Z pointer ++
        rcall uartSend                  ; uartSend(R16) MSB
        movw R26,R30                    ; address += 2
        mov R16,R24                     ; LSB stored in R16
        rjmp L70                        ; uartSend(R16) LSB

L40:    cpi R16,'D'                     ; else if (R16=='D') Write data to EEPROM
        brne L42
        rcall uartGet
        out EEDR,R16                    ; EEDR = uartGet()
        ldi R17,6                       ; write EEPROM
        rcall EepromTalk
        rjmp L68                        ; uartSend('\r')

L42:    cpi R16,'d'                     ; else if (R16=='d') Read data from EEPROM
        brne L44
        ldi R17,1                       ; read EEPROM
        rcall EepromTalk                ; R16 = EEPROM data
        rjmp L70                        ; uartSend(R16)

L44:    cpi R16,'F'                     ; else if(R16=='F') Read fuse bits
        brne L46
        clr R30                         ; Z-pointer = 0000
        rjmp L50                        ; rcall readFuseAndLock

L46:    cpi R16,'r'                     ; else if(R16=='r')  Read lock bits
        brne L48
        ldi R30,1                       ; Z pointer = 0001
        rjmp L50                        ; rcall readFuseAndLock

L48:    cpi R16,'N'                     ; else if(R16=='N') Read high fuse bits
        brne L52
        ldi R30,3                       ; Z-pointer = 0003
L50:    rcall readFuseAndLock
        rjmp L70                        ; uartSend(R16)

L52:    cpi R16,'t'                     ; else if(R16=='t') Return supported devices code
        brne L54
        ldi R16,DT                      ; Device Type
        rcall uartSend                  ; uartSend(DT) send Device Type
        clr R16
        rjmp L70                        ; uartSend(0)

L54:                                    ; else if ((R16=='l')||(R16=='x')||(R16=='y')||(R16=='T'))
        cpi R16,'l'             ; 'l' = Write Boot Loader lockbits
        breq L56
        cpi R16,'x'                     ; 'x' = Set LED
```

```
        breq L56
        cpi R16,'y'                     ; 'y' = Clear LED
        breq L56
        cpi R16,'T'                     ; 'T' = Select device type
        brne L60
L56:    rcall uartGet                   ; R16 = uartGet()
                                        ; YOU CAN INSERT LEDs CODE  HERE
        rjmp L68                        ; uartSend('\r')

L60:    cpi R16,'S'                     ; else if (R16=='S') Return software identifier
        brne L62
        ldi R30,low(2*Soft_Id)
        ldi R31,high(2*Soft_Id)
L61:    lpm R16,Z+
        tst R16
        breq L72                        ; branch is end of string  ((Z) == 0)
        rcall uartSend                  ; else send char
        rjmp L61

L62:    cpi R16,'V'                     ; else if (R16=='V') Return Software Version
        brne L64
        ldi R16,'1'                     ; uartSend('1')
        rcall uartSend
        ldi R16,'2'                     ; uartSend('2')
        rjmp L70                        ; uartSend(R16)

L64:    cpi R16,'s'                     ; else if (R16=='s') Return Signature Byte
        brne L66
        ldi R16,SB1                     ; uartSend(SB1) Signature Byte 1
        rcall uartSend
        ldi R16,SB2                     ; uartSend(SB2) Signature Byte 2
        rcall uartSend
        ldi R16,SB3                     ; uartSend(SB3) Signature Byte 3
        rjmp L70                        ; uartSend(R16)

L66:    ldi R16,'?'                     ; else uartSend('?')
        rjmp L70                        ; uartSend(R16)
L68:    ldi R16,13                      ; uartSend('\r')
L70:    rcall uartSend                  ; uartSend(R16)
L72:    rjmp L10

readFuseAndLock:
        clr R31                         ; Z pointer high byte = 0
        ldi R24,9                       ; SPMCR = 0x09
        out SPMCR,R24                   ; read fuse and lock
        lpm R16,Z                       ; read program memory
        ret
```

```
EepromTalk:                             ; if R17 == 6 then Write, if R17 == 1 then Read
        out EEARL,R26                   ; EEARL = address low
        out EEARH,R27                   ; EEARH = address high
        adiw R26,1                      ; address++
        sbrc R17,1                      ; skip if R17 == 1 (read Eeprom)
        sbi EECR,EEMWE                  ; EEMWE = 1  (write Eeprom)
        out EECR,R17                    ; EECR = R17 (6 write, 1 read)
L90:    sbic EECR,EEWE                  ; wait until EEWE == 0
        rjmp L90
        in R16,EEDR                     ; R16 = EEDR
        ret

uartSend:                               ; send R16
        sbis UCSRA,UDRE                 ; wait for empty transmit buffer (until UDRE==1)
        rjmp uartSend
        out UDR,R16                     ; UDR = R16, start transmission
        ret

uartGet:
        sbis UCSRA,RXC                  ; wait for incoming data (until RXC==1)
        rjmp uartGet
        in R16,UDR                      ; return received data in R16
        ret

Soft_Id: .DB "AVRB163", 0
```

; END of BOOT LOADER PROGRAM