AVR100: Accessing the EEPROM

Features
• Random Read/Write
• Sequential Read/Write
• Runable Test/Example Program

Introduction
This application note contains routines for access of the EEPROM memory in the AVR Microcontroller. Two types of Read/Write access has been implemented:
• Random read/write: The user must set up both data and address before calling the Read or Write routine
• Sequential read/write: The user needs only to set up the data to be read/written. The current EEPROM address is automatically incremented prior to access. The address has to be set prior to writing the first byte in a sequence

The application note contains four routines which are described in detail in the following sections. This application note contains routines for accessing the EEPROM in all AVR devices.

Random Write - Subroutine “EEWrite”
Three register variables must be set up prior to calling this routine:
• EEdwr - Data to be written
• EEawr - Address low byte to write
• EEawrh - Address high byte to write
The subroutine waits until the EEPROM is ready to be programmed by polling the EEPROM Write Enable - EEWE bit in the EEPROM Control Register - EECR. When EEWE is zero, the contents of EEdwr is transferred to the EEPROM Data Register - EEDR, and the contents of EEawrh:EEawr is transferred to the EEPROM Address Register - EEARH:EEARL. First the EEPROM master write enable - EEMWE is set, followed by the EEPROM write strobe EEWE in EECR. See Figure 1.

Figure 1. “EEWrite” Flow Chart
Random Read - Subroutine “EERead”

Prior to calling this routine, two register variables must be set up:
EEard - Address of low byte to read from
EEardh - Address of high byte to read from

The subroutine waits until the EEPROM is ready to be accessed by polling the EEWE bit in the EEPROM Control Register - EECR. When EEWE is zero, the subroutine and transfers the contents of EEardh:EEard to the EEPROM Address Register - EEARH:EEARL.

It then sets the EEPROM Read Strobe - EERE.

In the next instruction the content of the EEDR register is transferred to the register variable EEdrd. See Figure 2.

Figure 2. “EERead” Flow Chart

Sequential Write - Subroutine “EEWrite_seq”

Prior to calling this routine, one register variable must be set up:
EEdwr_s - Data to write

The subroutine waits until the EEPROM is ready to be programmed by polling the EEWE bit in the EEPROM Control Register - EECR. When EEWE is zero and the contents of the EEPROM address register - EEARH:EEARL are read into the register variable EEWTMPH:EEWTMP. EEwtmp is incremented and written back to EEARH:EEARL. This increments the current EEPROM address by one. The contents of EEdwr is then transferred to the EEPROM Data Register - EEDR, before EEWE in EECR is set, and then EEMWE is set. See Figure 3.

Figure 3. “EEWrite_seq” Flow Chart
Sequential Read - Subroutine “EERead_seq”

The subroutine waits until the EEPROM is ready to be accessed by polling the EEWE bit in the EEPROM Control Register - EECR. The subroutine then increments the current EEPROM address by performing the following operation: Transfer EEAR to the register variable EERTMPH:EERTMP, increments this register and writes the new address back to EEARH:EEARL. The routine then sets the EEPROM Read Strobe - EERE twice. Finally, the EEPROM data is transferred from EEDR to the register variable EErd_s. See Figure 4.

Figure 4. “EERead_seq” Flow Chart for 8515

Optimization for different devices

Not all the instructions are necessary for all devices. If the device has an EEPROM of 256 bytes or less, the high address of the EEPROM address register doesn’t need to be changed. On the AT90S1200, the EEMWE bit in the EEGR doesn’t have to be set.

See the section EEPROM Read/Write in the datasheet for further information.

Test Program

The application note assembly file contains a complete program which calls the four subroutines as a test of operation, and also as an example of usage. The test program is suitable for running in AVR Studio.

The test programs contains comments on how to port the code to work on any AVR-part.

Note: If the code initiates a write to EEPROM shortly after reset, keep in mind the following: If EEPROM contents are programmed during the manufacturing process, the MCU might change the code shortly after programming. When the programmer then verifies the EEPROM contents, this might fail because the EEPROM contents have already been modified by the MCU. Also notice that some in-system programmers will allow the MCU to execute a short time between each step in the programming and verification process.

Table 1. CPU and Memory Usage

<table>
<thead>
<tr>
<th>Function</th>
<th>Code Size</th>
<th>Cycles</th>
<th>Example Register Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEWrite</td>
<td>10 words</td>
<td>15</td>
<td>R16, R17, R18</td>
<td>EEPROM Random Location Write</td>
</tr>
<tr>
<td>EERead</td>
<td>7 words</td>
<td>11</td>
<td>R0, R17, R18</td>
<td>EEPROM Random Location Read</td>
</tr>
<tr>
<td>EEWrite_seq</td>
<td>13 words</td>
<td>19</td>
<td>R24, R25, R18</td>
<td>EEPROM Sequential Location Write</td>
</tr>
<tr>
<td>EERead_seq</td>
<td>10 words</td>
<td>17</td>
<td>R0, R24, R25</td>
<td>EEPROM Sequential Location Read</td>
</tr>
<tr>
<td>Reset</td>
<td>8 words</td>
<td>8</td>
<td>R16</td>
<td>Example Initialisation</td>
</tr>
<tr>
<td>Main</td>
<td>39 words</td>
<td>-</td>
<td>R16, R19, R20</td>
<td>Example Program</td>
</tr>
<tr>
<td>TOTAL</td>
<td>87 words</td>
<td>-</td>
<td>R0, R16, R17, R18, R19, R20, R24, R25</td>
<td>-</td>
</tr>
</tbody>
</table>
Table 2. Peripheral Usage

<table>
<thead>
<tr>
<th>Peripheral</th>
<th>Description</th>
<th>Interrupts Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 I/O Pins</td>
<td>LEDs (example only)</td>
<td>-</td>
</tr>
<tr>
<td>1 I/O Pin</td>
<td>Button (example only)</td>
<td>-</td>
</tr>
<tr>
<td>10 bytes EEPROM</td>
<td>Target EEPROM Locations (example only)</td>
<td>-</td>
</tr>
</tbody>
</table>

avr100.asm

;**** APPLICATION NOTE AVR100 ******************************
;*
;* Title: Accessing the EEPROM
;* Version: 2.0
;* Last updated: 98.10.14
;* Target: AT90S8515
;* Suitable for: Any AVR with internal EEPROM
;*
;* Support E-mail: avr@atmel.com
;*
;* DESCRIPTION
;* This Application note shows how to read data from and write data to the
;* EEPROM. Both random access and sequential access routines are listed.
;* The code is written for 8515. To modify for 90S4414,90S2313,90S2323...
;* apply the following changes:
;*-- Remove all entries to EEPROM Address Register High Byte EEARH
;*-- To modify for 90S1200, apply the changes above. In addition:
;*-- Remove all writes to EEMWE
;*
;* Change log
;*V2.098.10.14 (jboe) Bugfix, changed to support AT90S8515
;*V1.197.07.04 (gk) Created
;***************************************************************************
.include "8515def.inc"

rjmpRESET;Reset Handle

;***************************************************************************
;* EEWrite
;*
;* This subroutine waits until the EEPROM is ready to be programmed, then
;* programs the EEPROM with register variable "EEdwr" at address "EEawr:EEawr"
;*
;* Number of words : 7 + return
;* Number of cycles : 11 + return (if EEPROM is ready)
;* Low Registers used:None
;* High Registers used :3 (EEdwr,EEawr,EEawrh)
;*
;***** Subroutine register variables

.def EEdwr  =r16 ;data byte to write to EEPROM
.def EEawr  =r17 ;address low byte to write to
.def EEawrh =r18 ;address high byte to write to

;***** Code

EEPWrite:
    sbic   EECR, EEWE  ;if EEWE not clear
    rjmp   EEPWrite    ; wait more
    out    EEARH, EEawrh  ;output address high byte, remove if no high byte exist
    out    EEAL, EEawr   ;output address low byte
    out    EEDR, EEdwr  ;output data
    cli    ;disable global interrupts
    sbi    EECR, EEMWE  ;set master write enable, remove if AT90S1200 is used
    sbi    EECR, EEWE   ;set EEPROM Write strobe
    ;This instruction takes 4 clock cycles since
    ;it halts the CPU for two clock cycles
    sei    ;enable global interrupts
    ret

;*****************************************************************
;* EERead
;*****************************************************************

; This subroutine waits until the EEPROM is ready to be programmed, then
; reads the register variable "EEdrd" from address "EEard:EEard"
;*
;* Number of words : 6 + return
;* Number of cycles : 9 + return (if EEPROM is ready)
;* Low Registers used :1 (EEdrd)
;* High Registers used :2 (EEard,EEardh)
;*
;*****************************************************************

;***** Subroutine register variables

.def EEdrd  =r0 ;result data byte
.def EEard  =r17 ;address low to read from
.def EEardh =r18 ;address high to read from

;***** Code

EERead:
sbic EECR,EEWE ;if EEWE not clear
rjmp EERead ; wait more
out EEARH,EEardh ;output address high byte, remove if no high byte exist
out EEARL,EEard ;output address low byte
sbi EECR,EERE ;set EEPROM Read strobe

; output address high byte, remove if no high byte exist
;output address low byte
;
;This instruction takes 4 clock cycles since
;it halts the CPU for two clock cycles
;
;get data

ret

***************************************************************************
* EEWrite_seq
* This subroutine increments the EEPROM address by one and waits until the
* EEPROM is ready for programming. It then programs the EEPROM with
* register variable "EEdwr_s".
*
* Number of words : 12 + return
* Number of cycles : 15 + return (if EEPROM is ready)
* Low Registers used :None
* High Registers used :3 (EEdwr_s,EEwtmp,EEwtmph)
*
***************************************************************************

***** Subroutine register variables

.def EEwtmp =r24 ;temporary storage of address low byte
.def EEwtmph =r25 ;temporary storage of address high byte
.def EEdwr_s =r18 ;data to write

***** Code

EEWrite_seq:

sbic EECR,EEWE ;if EEWE not clear
rjmp EEWrite_seq ;wait more
in EEwtmp,EEARL ;get address low byte
in EEwtmph,EEARH ;get address high byte, remove if no high byte exists
adiw EEwtmp,0x01 ;increment address
out EEARL,EEwtmp ;output address low byte
out EEARH,EEwtmph ;output address byte, remove if no high byte exists
out EEDR,EEdwr_s ;output data
cli ;disable global interrupts
sbi EECR,EEMWE ;set master write enable, remove if 90S1200 is used
sbi EECR,EEWE ;set EEPROM Write strobe
; This instruction takes 4 clock cycles since
; it halts the CPU for two clock cycles
sei ; enable global interrupts
ret

;***************************************************************************
; *
; * EERead_seq
; *
; * This subroutine increments the address stored in EEAR and reads the
; * EEPROM into the register variable "EEdrd_s".
; *
; * Number of words : 9 + return
; * Number of cycles : 13 + return (if EEPROM is ready)
; * Low Registers used : 1 (EEdrd_s)
; * High Registers used: 2 (EEtmp, EEtmpH)
; *
;***************************************************************************

;***** Subroutine register variables
.def EEtmp = r24 ; temporary storage of low address
.def EEtmpH = r25 ; temporary storage of high address
.def EEdrd_s = r0 ; result data byte

;***** Code
EERead_seq:
    sbic EECR, EEWE ; if EEWE not clear
    rjmp EERead_seq ; wait more
; The above sequence for EEWE = 0 can be skipped if no write is initiated.

; Read sequence
    in EEtmp, EEARL ; get address low byte
    in EEtmpH, EEARH ; get address high byte, remove if no high byte exists
    adiw EEtmp, 0x01 ; increment address
    out EEARL, EEtmp ; output address low byte
    out EEARH, EEtmpH ; output address high byte, remove if no high byte exists
    sbi EECR, EERE ; set EEPROM Read strobe
    ; This instruction takes 4 clock cycles since
    ; it halts the CPU for two clock cycles
    in EEdrd_s, EEDR ; get data
    ret
;****************************************************************************
;* Test/Example Program
;*
;****************************************************************************

;***** Main Program Register variables
.def counter=r19
.def temp =r20

;***** Code

RESET:
;***** Initialize stack pointer
;* Initialize stack pointer to highest address in internal SRAM
;* Comment out for devices without SRAM

ldi r16,high(RAMEND) ;High byte only required if
out SPH,r16           ;RAM is bigger than 256 Bytes
ldi r16,low(RAMEND)  
out SPL,r16

;***** Initialize portB
;* Port B is used to verify the operation of the EEPROM read
;* and write routines.

ldi r16,0xff ; DDRB=0xff ->PortB=output
out DDRB,r16

;***** Initialize portD
; bit0 of PortD  is used to start the test program

ldi r16,0xff ; Enable all PortD pull-ups
out PORTD,r16

;***** Program start
;
main: in r16,PIND ; Wait for user to push button on PD0
sbrc r16,0
rjmp main

;***** Program a random location

ldi EEdwr,$aa
ldi EEawrh,$00
ldi EEawr,$10
rcall EEWrite ;store $aa in EEPROM location $0010

;**** Read from a random location

ldi EEardh,$00
ldi EEard,$10
rcall EERead ;read address $10
out PORTB,EEedrd ;output value to Port B

;**** Fill the EEPROM address 1..64 with bit pattern $55,$aa,$55,$aa,...

EEWrite_wait:
sbic EECR,EEWE ;if EEWE not clear
rjmp EEWrite_wait ; wait more

; The above sequence for EEWE = 0 can be skipped if it is guaranteed that no write is
; running when now changing the EEARL and EEARH registers.

ldi counter,63 ;init loop counter
clr temp
out EEARH,temp ;EEARH <- $00
clr temp
out EEARL,temp ;EEARL <- $00 (start address - 1)

loop1: ldi EEedwr_s,$55
rcall EEWrite_seq ;program EEPROM with $55
ldi EEedwr_s,$aa
rcall EEWrite_seq ;program EEPROM with $aa
dec counter ;decrement counter
brne loop1 ;and loop more if not done

;**** Copy 10 first bytes of EEPROM to r1-r11

EERead_wait:
sbic EECR,EEWE ;if EEWE not clear
rjmp EERead_wait ; wait more

; The above sequence for EEWE = 0 can be skipped if it is guaranteed that no write is
; running when we later change the EEARL and EEARH registers.

clr temp
out EEARH,temp ;EEARH <- $00
ldi temp,$00
out EEARL,temp ;EEARL <- $00 (start address - 1)
clr ZH
ldi ZL,1 ;Z-pointer points to r1

loop2: rcall EERead_seq ;get EEPROM data
st Z,EEedrd_s ;store to SRAM
inc ZL
cpi  ZL,12 ; reached the end?
brne  loop2 ; if not, loop more

forever:
rjmp  forever ; This is the end. On completion, the program ends up here