AVR ASM INTRODUCTION

Search this site

AVR ASSEMBLER TUTOR >

4a. USING THE EEPROM

A MORONS GUIDE to EEPROMS v1.3

by RetroDan@GMail.com

CONTENTS:

Electronically Erasable Programmable Read Only Memory (EEPROM) is very similar to Flash memory. Flash memory is good for 10,000 writes but is faster. EEPROMs are slower to write and erase but are good for 10 times the number read/writes. The EEPROMS in the AVRs are to hold vital data that needs to be preserved if the power goes out.

EEPROM cells can be though of as little batteries or capacitors, when erased they are all charged to one. When we program a number into a location, only the bits that need to be zero are discharged. This waiting to charge or discharge EEPROM cells takes considerable time (ms) on a hardware scale.

We will connect a speaker to one of the output pins of the ATtiny13 and create a program that produces sound. We will use this sound output to check that the EEPROM is being written & read correctly.

We are using the AVR ATtiny13 for its simplicity & small number of pins. The ATtiny13 runs at 1.2MHz (9.6MHz Oscillator divided by 8) with 1K of RAM and 64 Bytes of EEPROM. The example programs

should run on the Attiny13, ATtiny25, ATtiny45 or ATtiny85. The difference between these chips are basically the amount of Flash Ram for Program Memory. The circuits & code should be easy to adapt to most any AVR Core chips.

The Pin-outs of the ATtiny13 chip are:

```
          ATTINY13
          .-----.
PB5 --|1 A 8|-- Vcc
PB3 ->|2 T 7|-- PB2
PB4 --|3 N 6|-- PB1
GND --|4 Y 5|-->PB0
          `-----'
```

We are using Port B zero (PB0 pin #5) in the lower right-hand corner of the chip. For output connect a small speaker on pin 5 to ground. If you are using more than 3 Volts then put a 100-220 Ohm resistor between the speaker and ground. Here is the simple circuit:

```
                3VDC
  ATTINY13  |
    .-----.  |
 --|1 A 8|--'
 --|2 T 7|--
 --|3 N 6|--
.--|4 Y 5|---[]< SPEAKER
|  `-----'   |
|            |
`------------+ GROUND
```

CREATING A TONE MAKER:

We will create a small program that will emit a tone on a speaker connected to Port B Zero (PB0). Later we will build it up to test our EEPROM read/writes.

First we tell the assembler to read the TN13DEF.INC file for the definitions for the chip we are using, then we define the registers that we will use:

```
.INCLUDE "TN13DEF.INC"      ;AVR ATTINY13 DEF
```

```
.DEF A       = R16          ;GENERAL PURPOSE
```

Since we are not using interrupts we can start our program at the bottom of program flash memory at zero:

```
.ORG $0000
```

Next we tell the system that we are using Port B Zero for output by setting the the zero-bit in the Data Direction Register for Port B (DDRB):

```
RESET: SBI   DDRB,0         ;SET PORTB0 FOR O
```

We are going to have our program emit a tone based on the value in the Accumulator "A" (R16). We toggle the speaker pin then wait an amount of time depending on the value of the "A" register. If we do this over and over, the result will be a tone from the speaker, and the value stored in "A" determines its frequency.

Here we load the Accumulator with 100, then wait for 100 loops in a pause routine then we toggle the speaker port, and we do it over and over. The result is a tone will emit from the speaker.

```
MLUPE: LDI   A,100          ;LOAD "A" WITH 10
       RCALL PAUSE          ;WAIT 100 LOOPS
       SBI   PINB,0         ;TOGGLE THE SPEAK
        RJMP MLUPE          ;LOOP-BACK DO IT
```

The PAUSE routine subtracts one from "A" over and over and when it equals zero, we return. The result is a pause whose length is determined by the value in "A".

```
PAUSE: DEC A                ;SUBTRACT ONE FRO
        BRNE PAUSE          ;WAIT UNTIL IT RE
         RET
```

Here is what our complete Tone Maker Program looks like:

```
.INCLUDE "TN13DEF.INC" ;AVR ATTINY13 DEFINITI
.DEF A       = R16          ;GENERAL PURPOSE

.ORG $0000
RESET: SBI   DDRB,0         ;SET PORTB0 FOR O
MLUPE: LDI   A,100          ;LOAD "A" WITH 10
```

```
            RCALL PAUSE          ;WAIT 100 LOOPS
            SBI   PINB,0         ;TOGGLE THE SPEAK
             RJMP MLUPE          ;LOOP-BACK DO IT

PAUSE:  DEC A                    ;SUBTRACT ONE FRO
            BRNE PAUSE           ;WAIT UNTIL IT RE
             RET
```

If we connect the speaker properly and programmed our AVR, when activated it should emit a solid tone.

CREATING A TONE PLAYER:

To test our EEPROM read & writes we need a program that will emit a tone for a brief period of time. A different tone for each value that we store in EEPROM.

The main loop of our next program simply loads the accumulator with two different values and calls a routine that will play a note based on that value for a short period of time:

```
MLUPE:  LDI   A,100          ;LOAD TONE #1
        RCALL HOLD_TONE      ;PLAY IT
        LDI   A,200          ;LOAD TONE #2
        RCALL HOLD_TONE      ;PLAY IT
         RJMP MLUPE          ;LOOP-BACK DO IT
```

The HOLD_TONE routine calls the FREQ routine 255 times to give us a tone at a frequency depending on the value of the accumulator "A". So it will emit a steady tone for a brief period of time:

```
HOLD_TONE:
        RCALL FREQ           ;PAUSE BETWEEN C
        DEC R10              ;LOOP TO HOLD TO
         BRNE HOLD_TONE
          RET                ;RETURN
```

Our frequency routine (FREQ) saves the value of the accumulator "A" on the stack each time it is called, then toggles the speaker bit on port zero with a very small pause based on the value of "A". When called 255 times it will produce a frequency that varies with the value of "A":

```
FREQ:  PUSH  A              ;SAVE "A"
       SBI   PINB,0         ;TOGGLE SPEAKER
FLUPE: DEC   A              ;SUBTRACT ONE FR
        BRNE FLUPE          ;WAIT UNTIL IT R
       POP   A              ;RESTORE "A"
        RET
```

THE TONE PLAYER PROGRAM:

This is how our complete program looks now:

```
.INCLUDE "TN13DEF.INC"      ;AVR ATTINY13 DE
.DEF A       = R16          ;GENERAL PURPOSE

.ORG $0000
RESET:  SBI   DDRB,0        ;SET PORTB0 FOR
MLUPE:  LDI   A,100         ;LOAD TONE #1
        RCALL HOLD_TONE     ;PLAY IT
        LDI   A,200         ;LOAD TONE #2
        RCALL HOLD_TONE     ;PLAY IT
         RJMP MLUPE         ;LOOP-BACK DO IT

HOLD_TONE:
        RCALL FREQ          ;PAUSE BETWEEN C
        DEC R10             ;LOOP TO HOLD TO
         BRNE HOLD_TONE
          RET               ;RETURN

FREQ:  PUSH  A              ;SAVE "A"
       SBI   PINB,0         ;TOGGLE SPEAKER
FLUPE: DEC   A              ;SUBTRACT ONE FR
        BRNE FLUPE          ;WAIT UNTIL IT R
       POP   A              ;RESTORE "A"
        RET
```

If you connected the circuit properly and entered the program, you should hear two tones coming from the speaker, like some retro space toy.

CREATING AN EEPROM ERASE-WRITE THEN

READ PROGRAM:

The basic concept behind writing to the internal EEPROM is quite simple, we load a register with the data we wish to store, then we load another one with the address within the EEPROM and we tell it to write. Register "A" will hold the data we wish to write to the EEPROM and the ADR register will hold the address (byte number) inside the EEPROM of where we want the data stored.

The start of this program takes two values for register "A" and writes them to the EEPROM starting at byte zero (ADR = 0):

```
.INCLUDE "TN13DEF.INC"      ;AVR ATTINY13 DEF
.DEF A       = R16          ;GENERAL PURPOSE
.DEF B       = R18          ;GENERAL PURPOSE
.DEF ADR     = R28          ;HOLDS EEPROM ADD

.ORG $0000
RESET: SBI   DDRB,0         ;SET PORTB0 FOR O
       CLR   ADR            ;MAKE SURE ADDRES
MLUPE: LDI   A,50           ;LOAD TONE #1
       RCALL EE_WRITE       ;WRITE IT TO EEPR
       LDI   A,100          ;LOAD TONE #2
       RCALL EE_WRITE       ;WRITE IT TO EEPR
```

Next we call a routine called EE_READ that will fetch our values from the EEPROM and we call the HOLD_TONE routine to play a tone based on the values retrieved. If the sound emitted from the speaker is the similar as before, that tells us that the values were successfully written and read from the EEPROM:

```
PLAY_LOOP:
       CLR  ADR             ;START READS AT Z
       RCALL EE_READ        ;READ EEPROM INTO
       RCALL HOLD_TONE      ;PLAY TONE
       RCALL EE_READ        ;READ EEPROM INTO
       RCALL HOLD_TONE      ;PLAY TONE
        RJMP PLAY_LOOP      ;LOOP-BACK DO IT
```

THE EEPROM ERASE-WRITE ROUTINE:

An EEPROM write can take quite a while in terms of computer clocks, so if we are writing a block of data, we must check that the previous write has completed by checking the EEPROM Program Enable bit (EEPE) of the EEPROM Control Register (EECR). The SBIC will skip the RJMP EE_WRITE when the EEPE bit flips to zero.

When we write to the EEPROM we set the EEPE bit to one and the system clears it to zero when it is complete:

```
EE_WRITE:
        SBIC EECR,EEPE        ;CHECK IF EEPROM
         RJMP EE_WRITE        ;LOOP-BACK IF NOT
```

Next we load our address into the EEPROM Address Register (EEARL) and our data into the EEPROM Data Register (EEDR):

```
        OUT EEARL,ADR         ;EPROM ADDRESS
        OUT EEDR,A            ;EEPROM DATA TO W
```

Now that we have our data and address loaded we instruct the EEPROM to erase any old data and to write our new data. To do this we must enable the EEPROM write by setting two bit within four clock cycles. First we set the EEMPE bit followed immediately by setting the EEPE bit of the EEPROM Control Register (EECR). This helps to prevent accidental writes to the EEPROM.

```
        SBI EECR,EEMPE        ;ENABLE EEPROM
        SBI EECR,EEPE         ;ENABLE WRITE
```

At the end of our write routine we increment our address register (ADR) by one and return:

```
        INC ADR               ;INCREMENT EEPROM
         RET                  ;RETURN
```

THE EEPROM READ ROUTINE:

As we did in the write routine, we poll the EEPROM Enable Program bit (EEPE) of the EEPROM Control Register (EECR) to make sure any previous EEPROM accesses have completed:

```
EE_READ:
       SBIC EECR,EEPE       ;CHECK IF EEPROM
        RJMP EE_READ        ;ITS BUSY SO WE W
```

Now we move the address/byte of the location inside the EEPROM that we wish to read into the EEPROM Address Register (EEARL):

```
       OUT EEARL,ADR        ;SET-UP THE ADDRE
```

We now set the read mode bit of the EECR register and read the data into our "A" register:

```
       SBI EECR,EERE        ;SET-UP TO READ
       IN  A,EEDR           ;READ THE DATA RE
```

We increment our address register (ADR) by one and return:

```
       INC ADR              ;INCREMENT EEPROM
        RET                 ;RETURN
```

THE ERASE-WRITE THEN READ EEPROM PROGRAM:

After we make all the appropriate changes, this is what our complete program looks like:

```
.INCLUDE "TN13DEF.INC"      ;AVR ATTINY13 DEF
.DEF A       = R16          ;GENERAL PURPOSE
.DEF B       = R18          ;GENERAL PURPOSE
.DEF ADR     = R28          ;HOLDS EEPROM ADD

.ORG $0000
RESET: SBI   DDRB,0         ;SET PORTB0 FOR O
       CLR   ADR            ;MAKE SURE ADDRES
MLUPE: LDI   A,50           ;LOAD TONE #1
       RCALL EE_WRITE       ;WRITE IT TO EEPR
       LDI   A,100          ;LOAD TONE #2
       RCALL EE_WRITE       ;WRITE IT TO EEPR

PLAY_LOOP:
       CLR  ADR             ;START READS AT Z
       RCALL EE_READ        ;READ EEPROM INTO
```

```
        RCALL HOLD_TONE      ;PLAY TONE
        RCALL EE_READ        ;READ EEPROM INTO
        RCALL HOLD_TONE      ;PLAY TONE
         RJMP PLAY_LOOP      ;LOOP-BACK DO IT

HOLD_TONE:
        RCALL FREQ           ;PAUSE BETWEEN CL
        DEC R10              ;LOOP TO HOLD TON
         BRNE HOLD_TONE
          RET                ;RETURN

FREQ:  PUSH  A               ;SAVE "A"
       SBI   PINB,0          ;TOGGLE SPEAKER
FLUPE: DEC   A               ;SUBTRACT ONE FRO
        BRNE FLUPE           ;WAIT UNTIL IT RE
       POP   A               ;RESTORE "A"
        RET

EE_WRITE:
       SBIC EECR,EEPE        ;CHECK IF EEPROM
        RJMP EE_WRITE        ;LOOP-BACK IF NOT
       OUT EEARL,ADR         ;EPROM ADDRESS
       OUT EEDR,A            ;EEPROM DATA TO W
       SBI EECR,EEMPE        ;ENABLE EEPROM
       SBI EECR,EEPE         ;ENABLE WRITE
       INC ADR               ;INCREMENT EEPROM
        RET                  ;RETURN

EE_READ:
       SBIC EECR,EEPE        ;CHECK IF EEPROM
        RJMP EE_READ         ;ITS BUSY SO WE W
       OUT EEARL,ADR         ;SET-UP THE ADDRE
       SBI EECR,EERE         ;SET-UP TO READ
       IN  A,EEDR            ;READ THE DATA RE
       INC ADR               ;INCREMENT EEPROM
        RET                  ;RETURN
```

This time the sound we hear will be the similar as the last program, but the tones are different and are being read-in from the EEPROM.

CREATING AN ERASE, THEN WRITE, THEN READ PROGRAM:

This time we will use separate routines and commands to first erase the EEPROM memory, then we do a write. Each write call is proceeded by an EE_ERASE in the main loop of the program:

```
MLUPE: LDI   A,200          ;LOAD TONE #1
        RCALL EE_ERASE      ;ERASE EEPROM BYT
        RCALL EE_WRITE      ;WRITE IT TO EEPR
```

Erasing the EEPROM discharges its cells to produce all ones. Therefore, an unprogrammed location would read $FF (0b1111_1111). Here we put the system into EEPROM Erase mode by setting the EEPM0 bit to one:

```
EE_ERASE:
       SBIC EECR,EEPE       ;CHECK IF EEPROM
        RJMP EE_ERASE       ;LOOP-BACK IF NOT
       LDI B,0b0000_0001    ;SET EEPM0,EEPROM
       OUT EECR,B           ;SET MODE TO ERAS
       OUT EEARL,ADR        ;EPROM ADDRESS
       OUT EEDR,A           ;EEPROM DATA TO W
       SBI EECR,EEMPE       ;ENABLE EEPROM
       SBI EECR,EEPE        ;ENABLE ERASE
        RET                 ;RETURN
```

Our write routine is exactly the same a previously except the EEPM1 bit is set to tell the system we want a write-only without the erase, because we erased the location manually in our previous routine:

```
EE_WRITE:
       SBIC EECR,EEPE       ;CHECK IF EEPROM
        RJMP EE_WRITE       ;LOOP-BACK IF NOT
       LDI B,0b0000_0010    ;SET EEPM1, EEPRO
       OUT EECR,B           ;SET MODE TO WRIT
       OUT EEARL,ADR        ;EPROM ADDRESS
       OUT EEDR,A           ;EEPROM DATA TO W
       SBI EECR,EEMPE       ;ENABLE EEPROM
       SBI EECR,EEPE        ;ENABLE WRITE
       INC ADR              ;INCREMENT EEPROM
```

```
        RET                 ;RETURN
```

After we make those changes, this is how our entire program looks:

```
.INCLUDE "TN13DEF.INC"      ;AVR ATTINY13 DEF
.DEF A      = R16           ;GENERAL PURPOSE
.DEF B      = R18           ;GENERAL PURPOSE
.DEF ADR    = R28           ;HOLDS EEPROM ADD

.ORG $0000
RESET: SBI   DDRB,0         ;SET PORTB0 FOR O
       CLR   ADR            ;MAKE SURE ADDRES
MLUPE: LDI   A,50           ;LOAD TONE #1
       RCALL EE_ERASE       ;ERASE EEPROM BYT
       RCALL EE_WRITE       ;WRITE IT TO EEPR
       LDI   A,150          ;LOAD TONE #2
       RCALL EE_ERASE       ;ERASE EEPROM BYT
       RCALL EE_WRITE       ;WRITE IT TO EEPR

PLAY_LOOP:
       CLR  ADR             ;START READS AT Z
       RCALL EE_READ        ;READ EEPROM INTO
       RCALL HOLD_TONE      ;PLAY TONE
       RCALL EE_READ        ;READ EEPROM INTO
       RCALL HOLD_TONE      ;PLAY TONE
        RJMP PLAY_LOOP      ;LOOP-BACK DO IT

HOLD_TONE:
       RCALL FREQ           ;PAUSE BETWEEN CL
       DEC R10              ;LOOP TO HOLD TON
        BRNE HOLD_TONE
         RET                ;RETURN

FREQ:  PUSH  A              ;SAVE "A"
       SBI   PINB,0         ;TOGGLE SPEAKER
FLUPE: DEC   A              ;SUBTRACT ONE FRO
        BRNE FLUPE          ;WAIT UNTIL IT RE
       POP   A              ;RESTORE "A"
        RET

EE_ERASE:
```

```
        SBIC EECR,EEPE       ;CHECK IF EEPROM
         RJMP EE_ERASE       ;LOOP-BACK IF NOT
        LDI B,0b0000_0001    ;SET EEPM0,EEPROM
        OUT EECR,B           ;SET MODE TO ERAS
        OUT EEARL,ADR        ;EPROM ADDRESS
        OUT EEDR,A           ;EEPROM DATA TO W
        SBI EECR,EEMPE       ;ENABLE EEPROM
        SBI EECR,EEPE        ;ENABLE ERASE
         RET                 ;RETURN

EE_WRITE:
        SBIC EECR,EEPE       ;CHECK IF EEPROM
         RJMP EE_WRITE       ;LOOP-BACK IF NOT
        LDI B,0b0000_0010    ;SET EEPM1, EEPRO
        OUT EECR,B           ;SET MODE TO WRIT
        OUT EEARL,ADR        ;EPROM ADDRESS
        OUT EEDR,A           ;EEPROM DATA TO W
        SBI EECR,EEMPE       ;ENABLE EEPROM
        SBI EECR,EEPE        ;ENABLE WRITE
        INC ADR              ;INCREMENT EEPROM
         RET                 ;RETURN

EE_READ:
        SBIC EECR,EEPE       ;CHECK IF EEPROM
         RJMP EE_READ        ;ITS BUSY SO WE W
        OUT EEARL,ADR        ;SET-UP THE ADDRE
        SBI EECR,EERE        ;SET-UP TO READ
        IN  A,EEDR           ;READ THE DATA RE
        INC ADR              ;INCREMENT EEPROM
         RET
```

SAVING TIME:

Since EEPROM erase can take a long time (1.8 ms on the ATtiny13) if speed is an issue, we could test the location to see if it is already erased. We would compare it to $FF since only the zeros are programmed, a blank location would be all ones:

```
EE_ERASE:
        MOV  B,A             ;PRESERVE VALUE O
```

```
        RCALL EE_READ        ;READ EEPROM LOCA
        CPI  A,$FF           ;CHECK IF ITS ERA
        MOV  A,B             ;RESTORE "A"
         BREQ EEE_XIT        ;IF ALREADY ERASE
EEE_WAIT:
        SBIC EECR,EEPE       ;CHECK IF EEPROM
         RJMP EEE_WAIT       ;LOOP-BACK IF NOT
        LDI B,0b0000_0001    ;SET EEPM0,EEPROM
        OUT EECR,B           ;SET MODE TO ERAS
        OUT EEARL,ADR        ;EPROM ADDRESS
        OUT EEDR,A           ;EEPROM DATA TO W
        SBI EECR,EEMPE       ;ENABLE EEPROM
        SBI EECR,EEPE        ;ENABLE ERASE
EEE_XIT: RET                 ;RETURN
```

We can do something similar with the write routine, check if the location in the EEPROM is already programmed. We could read it first and compare to what we are about to write. Since an EEPROM write can take a while (1.8 ms on the ATtiny13):

```
EE_WRITE:
    MOV  B,A             ;PRESERVE "A"
    RCALL EE_READ        ;READ EEPROM LOCATIO
    CP   A,B             ;CHECK IF ALREADY PR
    MOV  A,B             ;RESTORE "A"
     BREQ EEW_XIT        ;ALREADY PROGRAMMED
EEW_WAIT:
    SBIC EECR,EEPE       ;CHECK IF EEPROM AVA
     RJMP EEW_WAIT       ;LOOP-BACK IF NOT AV
    LDI B,0b0000_0010    ;SET EEPM1, EEPROM W
    OUT EECR,B           ;SET MODE TO WRITE O
    OUT EEARL,ADR        ;EPROM ADDRESS
    OUT EEDR,A           ;EEPROM DATA TO WRIT
    SBI EECR,EEMPE       ;ENABLE EEPROM
    SBI EECR,EEPE        ;ENABLE WRITE
EEW_XIT: RET             ;RETURN
```

With these changes made this is how our erase, then write, then read program looks. Notice that we increment the address pointer from outside the read/write routines this time since we will be calling the EE_READ routine from more than one place:

```
.INCLUDE "TN13DEF.INC" ;AVR ATTINY13 DEFINITI
.DEF A       = R16          ;GENERAL PURPOSE
.DEF B       = R18          ;GENERAL PURPOSE
.DEF ADR     = R28          ;HOLDS EEPROM ADD

.ORG $0000
RESET: SBI   DDRB,0         ;SET PORTB0 FOR O
       CLR   ADR            ;MAKE SURE ADDRES
MLUPE: LDI   A,50           ;LOAD TONE #1
       RCALL EE_ERASE       ;ERASE EEPROM BYT
       RCALL EE_WRITE       ;WRITE IT TO EEPR
       INC   ADR            ;INCREMENT OUR AD
       LDI   A,150          ;LOAD TONE #2
       RCALL EE_ERASE       ;ERASE EEPROM BYT
       RCALL EE_WRITE       ;WRITE IT TO EEPR

PLAY_LOOP:
       CLR  ADR             ;START READS AT Z
       RCALL EE_READ        ;READ EEPROM INTO
       INC  ADR             ;INCREMENT OUR AD
       RCALL HOLD_TONE      ;PLAY TONE
       RCALL EE_READ        ;READ EEPROM INTO
       RCALL HOLD_TONE      ;PLAY TONE
        RJMP PLAY_LOOP      ;LOOP-BACK DO IT

HOLD_TONE:
       RCALL FREQ           ;PAUSE BETWEEN CL
       DEC R10              ;LOOP TO HOLD TON
            BRNE HOLD_TONE
         RET                ;RETURN

FREQ:  PUSH  A              ;SAVE "A"
       SBI   PINB,0         ;TOGGLE SPEAKER
FLUPE: DEC   A              ;SUBTRACT ONE FRO
        BRNE FLUPE          ;WAIT UNTIL IT RE
       POP   A              ;RESTORE "A"
            RET

EE_ERASE:
       MOV  B,A             ;PRESERVE VALUE O
```

```
        RCALL EE_READ        ;READ EEPROM LOCA
        CPI  A,$FF           ;CHECK IF ITS ERA
        MOV  A,B             ;RESTORE "A"
         BREQ EEE_XIT        ;IF ALREADY ERASE

EEE_WAIT:
        SBIC EECR,EEPE       ;CHECK IF EEPROM
         RJMP EEE_WAIT       ;LOOP-BACK IF NOT
        LDI B,0b0000_0001    ;SET EEPM0,EEPROM
        OUT EECR,B           ;SET MODE TO ERAS
        OUT EEARL,ADR        ;EPROM ADDRESS
        OUT EEDR,A           ;EEPROM DATA TO W
        SBI EECR,EEMPE       ;ENABLE EEPROM
        SBI EECR,EEPE        ;ENABLE ERASE
EEE_XIT: RET                 ;RETURN

EE_WRITE:
        MOV  B,A             ;PRESERVE "A"
        RCALL EE_READ        ;READ EEPROM LOCA
        CP   A,B             ;CHECK IF ALREADY
        MOV  A,B             ;RESTORE "A"
         BREQ EEW_XIT        ;ALREADY PROGRAMM

EEW_WAIT:
        SBIC EECR,EEPE       ;CHECK IF EEPROM
         RJMP EEW_WAIT       ;LOOP-BACK IF NOT
        LDI B,0b0000_0010    ;SET EEPM1, EEPRO
        OUT EECR,B           ;SET MODE TO WRIT
        OUT EEARL,ADR        ;EPROM ADDRESS
        OUT EEDR,A           ;EEPROM DATA TO W
        SBI EECR,EEMPE       ;ENABLE EEPROM
        SBI EECR,EEPE        ;ENABLE WRITE
EEW_XIT: RET                 ;RETURN

EE_READ:
        SBIC EECR,EEPE       ;CHECK IF EEPROM
         RJMP EE_READ        ;ITS BUSY SO WE W
        OUT EEARL,ADR        ;SET-UP THE ADDRE
        SBI EECR,EERE        ;SET-UP TO READ
        IN  A,EEDR           ;READ THE DATA RE
```

```
        RET                 ;RETURN
```

THE EEPROM INTERRUPT METHOD:

For this program we will program the EEPROM then read the EEPROM and emit tones based on their values sixteen times, then we activate the EEPROM-Ready Interrupt and erase the EEPROM from inside the interrupt. At the speaker the noise emitted will change once it is erased.

When interrupts are enabled the ATtiny13 the system looks to the bottom of RAM ($0000) for an interrupt jump table to service any interrupts. The Start-Up or Reset vector is located at $0000 so we put a jump to our program there. The ATtiny13 Data Sheet tells us that the EEPROM Ready Interrupt is at $0004:

```
.ORG $0000
       RJMP RESET           ;RESET START VECT
.ORG $0004
       RJMP EE_RDY          ;EEPROM READY INT
```

We program the EEPROM as before, but with different values and we read them back from EEPROM and play them as tones sixteen times. Then we activate interrupts with the SEI command:

```
        INC N               ;INCREMENT LOOP C
        CPI N,16            ;TEN LOOPS YET?
         BRNE PLAY_LOOP     ;NO, SKIP
        SEI                 ;ACTIVATE INTERRU
```

When doing an interrupt we should save off the system status and contents of any registers we use because we might have interrupted something important. First we save the "A" & "B" registers, then the contents of the system status register (SREG):

```
EE_RDY: PUSH A               ;SAVE "A" ON STA
        PUSH B               ;SAVE "B" ON STA
        IN  A,SREG           ;SAVE STATUS...
        PUSH A               ;ON STACK
```

Inside the main part of our interrupt service routine (ISR) we increment our address pointer, then erase the contents if they need it, so it will

eventually erase the entire EEPROM:

```
            INC ADR
            RCALL EE_ERASE      ;ERASE LOCATION
```

This is what the entire EEPROM Interrupt Program looks like:

```
        .INCLUDE "TN13DEF.INC"       ;AVR ATTINY13 DE
        .DEF A       = R16           ;GENERAL PURPOSE
        .DEF B       = R18           ;GENERAL PURPOSE
        .DEF N       = R20           ;COUNTER
        .DEF ADR     = R28           ;HOLDS EEPROM AD

        .ORG $0000
           RJMP RESET                ;RESET START VEC
        .ORG $0004
           RJMP EE_RDY               ;EEPROM READY IN

        RESET: SBI   DDRB,0          ;SET PORTB0 FOR
        MLUPE: CLI                   ;SHUT DOWN ANY I
               CLR   ADR             ;MAKE SURE ADDRE
               CLR   N               ;COUNTER FOR LOO
               LDI   A,100           ;LOAD TONE #1
               RCALL EE_ERASE        ;ERASE EEPROM BY
               RCALL EE_WRITE        ;WRITE IT TO EEP
               INC ADR               ;INCREMENT OUR A
               LDI   A,250           ;LOAD TONE #2
               RCALL EE_ERASE        ;ERASE EEPROM BY
               RCALL EE_WRITE        ;WRITE IT TO EEP

        PLAY_LOOP:
               CLR  ADR              ;START READS AT
               RCALL EE_READ         ;READ EEPROM INT
               INC  ADR              ;INCREMENT OUR A
               RCALL HOLD_TONE       ;PLAY TONE
               RCALL EE_READ         ;READ EEPROM INT
               INC  ADR              ;INCREMENT OUR A
               RCALL HOLD_TONE       ;PLAY TONE
               INC N                 ;INCREMENT LOOP
               CPI N,16              ;TEN LOOPS YET?
                BRNE PLAY_LOOP       ;NO, SKIP
```

```
        SEI                     ;ACTIVATE INTERR
         RJMP PLAY_LOOP         ;LOOP-BACK DO IT

HOLD_TONE:
        RCALL FREQ              ;PAUSE BETWEEN C
        DEC R10                 ;LOOP TO HOLD TO
         BRNE HOLD_TONE
           RET                  ;RETURN

FREQ:  PUSH  A                  ;SAVE "A"
        SBI   PINB,0            ;TOGGLE SPEAKER
FLUPE: DEC   A                  ;SUBTRACT ONE FR
         BRNE FLUPE             ;WAIT UNTIL IT R
        POP   A                 ;RESTORE "A"
            RET

EE_RDY: PUSH A                  ;SAVE "A" ON STA
         PUSH B                 ;SAVE "B" ON STA
         IN  A,SREG             ;SAVE STATUS...
         PUSH A                 ;ON STACK
         INC ADR
         RCALL EE_ERASE         ;ERASE LOCATION
         POP A                  ;RESTORE STATUS.
         OUT SREG,A             ;TO STATUS REGIS
         POP B                  ;RESTORE "B"
         POP A                  ;RESTORE "A"
          RETI

EE_ERASE:
        MOV  B,A                ;PRESERVE VALUE
        RCALL EE_READ           ;READ EEPROM LOC
        CPI  A,$FF              ;CHECK IF ITS ER
        MOV  A,B                ;RESTORE "A"
         BREQ EEE_XIT           ;IF ALREADY ERAS
        SBIC EECR,EEPE          ;CHECK IF EEPROM
         RJMP EE_WRITE          ;LOOP-BACK IF NO
        LDI B,0b0000_1001       ;SET EEPM0,EEPRO
        OUT EECR,B              ;SET MODE TO ERA
        OUT EEARL,ADR           ;EPROM ADDRESS
        OUT EEDR,A              ;EEPROM DATA TO
```

```
        SBI EECR,EEMPE          ;ENABLE EEPROM
        SBI EECR,EEPE           ;ENABLE ERASE
EEE_XIT: RET                    ;RETURN

EE_WRITE:
        MOV  B,A                ;PRESERVE "A"
        RCALL EE_READ           ;READ EEPROM LOC
        CP   A,B                ;CHECK IF ALREAD
        MOV  A,B                ;RESTORE "A"
         BREQ EEW_XIT           ;ALREADY PROGRAM
        SBIC EECR,EEPE          ;CHECK IF EEPROM
         RJMP EE_WRITE          ;LOOP-BACK IF NO
        LDI B,0b0000_1010       ;SET EEPM1, EEPR
        OUT EECR,B              ;SET MODE TO WRI
        OUT EEARL,ADR           ;EPROM ADDRESS
        OUT EEDR,A              ;EEPROM DATA TO
        SBI EECR,EEMPE          ;ENABLE EEPROM
        SBI EECR,EEPE           ;ENABLE WRITE
EEW_XIT: RET                    ;RETURN

EE_READ:
        SBIC EECR,EEPE          ;CHECK IF EEPROM
         RJMP EE_READ           ;ITS BUSY SO WE
        OUT EEARL,ADR           ;SET-UP THE ADDR
        SBI EECR,EERE           ;SET-UP TO READ
        IN  A,EEDR              ;READ THE DATA R
         RET
```

SOME PRECAUTIONS:

The Atmel application notes warn that location zero of the EEPROMs have the potential of being corrupted, so for important project avoid the use of the first location, zero.

If we are using Store Program Memory (SPM), we must make sure any SPM command is completed before attempting any EEPROM commands:

```
SPM_BUSY:
        IN   B,SPMCSR          ;CHECK IF AN SPM
```

```
ANDI B,0b0000_0001   ;WAIT SPM ENABLE
 BRNE SPM_BUSY
```

If we are using other interrupts be sure to shut them off before you write to the EEPROM Control Register (EECR):

```
CLI                  ;SHUT-DOWN INTERR
SBI EECR,EEMPE       ;ENABLE EEPROM
SBI EECR,EEPE        ;ENABLE WRITE
 SEI                 ;RE-ENABLE INTERR
```

A sample of a write routine that takes into account SPM command and other interrupts:

```
EE_WRITE:
SPM_BUSY:
       IN   B,SPMCSR        ;CHECK IF AN SPM
       ANDI B,0b0000_0001   ;WAIT SPM ENABLE
        BRNE SPM_BUSY
EE_BUSY:
       SBIC EECR,EEPE       ;CHECK IF EEPROM
        RJMP EE_WRITE       ;LOOP-BACK IF NOT
       LDI B,0b0000_0000    ;SET EEPM0,EEPM1
       OUT EECR,B           ;SET MODE TO ERAS
       OUT EEARL,ADR        ;EPROM ADDRESS
       OUT EEDR,A           ;EEPROM DATA TO W
       CLI                  ;SHUT-DOWN INTERR
       SBI EECR,EEMPE       ;ENABLE EEPROM
       SBI EECR,EEPE        ;ENABLE WRITE
       SEI                  ;RE-ENABLE INTERR
       INC ADR              ;INCREMENT EEPROM
        RET                 ;RETURN
```

Comments

You do not have permission to add comments.