AVR ASM INTRODUCTION

Search this site

AVR ASSEMBLER TUTOR

- 1. AVR ASM BIT MANIPULATION
- 2a. BASIC ARITHMETIC
- 2b. BASIC MATH
- 2c. LOGARITHMS
- 2z. INTEGER RATIOS for FASTER CODE
- 3a. USING THE ADC
- 3b. BUTTERFLY ADC
- 4a. USING THE EEPROM
- **4b. BUTTERFLY EEPROM**
- 5. TIMER COUNTERS & PWM
- 6. BUTTERFLY LCD & JOYSTICK
- 7. BUTTERFLY SPI & AT45 DATAFLASH

Sitemap

AVR ASSEMBLER TUTOR >

4b. BUTTERFLY EEPROM

A MORONS GUIDE TO EEPROMS v1.2

BUTTERFLY EDITION

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 but 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 are using the AVR Butterfly with 16K RAM, 512 Bytes of EEPROM and a speaker on Port B,5.

CREATING A TONE MAKER

First we create a small program that will emit a tone on the speaker connected to Port B Five (PB5). 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 "M169DEF.INC"      ;AVR ATMEGA169 DEFINITION
.DEF A       = R16          ;GENERAL PURPOSE ACCUMULA
```

Since we are not using interrupts we can start our program at the bottom of memory at zero, and we place our stack at the top of memory:

```
.ORG $0000
RESET: LDI   A,LOW(RAMEND)  ;SET UP STACK AT TOP OF R
       OUT   SPL,A          ;LOW BYTE TOP OF RAM
       LDI   A,HIGH(RAMEND) ;HIGH BYTE TOP OF RAM
       OUT   SPH,A
```

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 OUTPUT TO
```

We are going to have our program emit a tone based on the value in the Accumulator "A". 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, then wait for loops in the pause routine then we toggle the speaker port, and we do it over and over. The result is a tone from the speaker.

```
MLUPE: LDI   A,255          ;LOAD "A" WITH 255
       RCALL PAUSE          ;WAIT
       SBI   PINB,5         ;TOGGLE THE SPEAKER
        RJMP MLUPE          ;LOOP-BACK DO IT AGAIN
```

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 FROM A
        BRNE PAUSE          ;WAIT UNTIL IT REACHES ZE
         RET
```

Here is what our complete Tone Maker Program looks like:

```
.INCLUDE "M169DEF.INC"      ;AVR ATMEGA169 DEFINITION
.DEF A       = R16          ;GENERAL PURPOSE ACCUMULA

.ORG $0000
RESET: LDI   A,LOW(RAMEND)  ;SET UP STACK AT TOP OF R
       OUT   SPL,A          ;LOW BYTE TOP OF RAM
       LDI   A,HIGH(RAMEND) ;HIGH BYTE TOP OF RAM
       OUT   SPH,A
       SBI   DDRB,5         ;SET PORTB5 FOR OUTPUT TO

MLUPE: LDI   A,255          ;LOAD "A" WITH 255
       RCALL PAUSE          ;WAIT
       SBI   PINB,5         ;TOGGLE THE SPEAKER
        RJMP MLUPE          ;LOOP-BACK DO IT AGAIN

PAUSE: DEC A                ;SUBTRACT ONE FROM A
        BRNE PAUSE          ;WAIT UNTIL IT REACHES ZE
         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,200          ;LOAD TONE #1
        RCALL HOLD_TONE      ;PLAY IT
        LDI   A,250          ;LOAD TONE #2
        RCALL HOLD_TONE      ;PLAY IT
         RJMP MLUPE          ;LOOP-BACK DO IT AGAIN
```

The HOLD_TONE routine calls the FREQ routine 255 times to give us a tone at a frequency dependent 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 CLICK
           DEC R10               ;LOOP TO HOLD TONE
            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,5          ;TOGGLE SPEAKER
FLUPE: DEC   A               ;SUBTRACT ONE FROM A
        BRNE FLUPE           ;WAIT UNTIL IT REACHES Z
       POP   A               ;RESTORE "A"
        RET
```

THE TONE PLAYER PROGRAM

This is how our complete program looks now:

```
.INCLUDE "M169DEF.INC"      ;AVR ATMEGA169 DEFINITION
.DEF A       = R16          ;GENERAL PURPOSE ACCUMULA

.ORG $0000
RESET: LDI   A,LOW(RAMEND)  ;SET UP STACK AT TOP OF R
       OUT   SPL,A          ;LOW BYTE TOP OF RAM
       LDI   A,HIGH(RAMEND) ;HIGH BYTE TOP OF RAM
       OUT   SPH,A
       SBI   DDRB,5         ;SET PORTB5 FOR OUTPUT TO

MLUPE: LDI   A,200          ;LOAD TONE #1
       RCALL HOLD_TONE      ;PLAY IT;
       LDI   A,250          ;LOAD TONE #2
       RCALL HOLD_TONE      ;PLAY IT
        RJMP MLUPE          ;LOOP-BACK DO IT AGAIN

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

FREQ:  PUSH A               ;SAVE "A" REGISTER ON STA
       SBI   PINB,5         ;TOGGLE SPEAKER
FLUPE: DEC   A              ;SUBTRACT ONE FROM A
        BRNE FLUPE          ;WAIT UNTIL IT REACHES ZE
       POP A                ;RESTORE "A" FROM STACK
         RET
```

If you connected the circuit properly and entered the program, you should hear two tones coming from the speaker that are rather annoying. To make the sound more appealing, we can slow down the Butterfly to run at 1Mhz by setting the clock pre-scaler/divider register (CLKPR):

```
RESET: LDI   A,LOW(RAMEND)  ;SET UP STACK AT TOP OF R
       OUT   SPL,A          ;LOW BYTE TOP OF RAM
       LDI   A,HIGH(RAMEND) ;HIGH BYTE TOP OF RAM
       OUT   SPH,A
       LDI A,128            ;SET SYS CLOCK SPEED
       STS CLKPR,A
       LDI A,3              ;0=8MHz 1=4MHz 2=2MHz 3=1
       STS CLKPR,A          ;BUTTERFLY @ 2MHZ AS SHIP
       SBI   DDRB,5         ;SET PORTB5 FOR OUTPUT TO
```

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 you tell it to write. "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 "A" and writes them to the EEPROM starting at byte zero (ADR = 0):

```
       CLR   ADR             ;MAKE SURE ADDRESS STARTS
MLUPE: LDI   A,100           ;LOAD TONE #1
       RCALL EE_WRITE        ;WRITE IT TO EEPROM
       LDI   A,200           ;LOAD TONE #2
       RCALL EE_WRITE        ;WRITE IT TO EEPROM
```

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

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,EEWE        ;CHECK IF EEPROM AVAILABL
        RJMP EE_WRITE        ;LOOP-BACK IF NOT AVAILAB
```

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 WRITE
```

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 EEMWE bit followed immediately by setting the EEWE bit of the EEPROM Control Register (EECR). This helps to prevent accidental writes to the EEPROM.

```
        SBI EECR,EEMWE       ;ENABLE EEPROM
        SBI EECR,EEWE        ;ENABLE WRITE
```

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

```
       INC ADR              ;INCREMENT EEPROM ADDRESS
        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 BUSY
        RJMP EE_READ        ;ITS BUSY SO WE WAIT
```

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 ADDRESS
```

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 REGISTER
```

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

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

THE ERASE-WRITE THEN READ EEPROM PROGRAM

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

```
.INCLUDE "M169DEF.INC"      ;AVR ATTINY13 DEFINITIONS
.DEF A       = R16          ;GENERAL PURPOSE ACCUMULA
.DEF B       = R18          ;GENERAL PURPOSE REGISTER
.DEF ADR     = R24          ;HOLDS EEPROM ADDRESS

.ORG $0000
RESET: LDI   A,LOW(RAMEND)  ;SET UP STACK...
       OUT   SPL,A          ;AT TOP OF MEMORY
       LDI   A,HIGH(RAMEND)
       OUT   SPH,A
       LDI A,128            ;SET SYS CLOCK SPEED
       STS CLKPR,A
       LDI A,3              ;0=8MHz 1=4MHz 2=2MHz 3=1
       STS CLKPR,A          ;BUTTERFLY @ 2MHZ AS SHIP
       SBI   DDRB,5         ;SET FOR OUTPUT TO SPEAKE
       CLR   ADR            ;MAKE SURE ADDRESS STARTS
MLUPE: LDI   A,100          ;LOAD TONE #1
       RCALL EE_WRITE       ;WRITE IT TO EEPROM
       LDI   A,200          ;LOAD TONE #2
       RCALL EE_WRITE       ;WRITE IT TO EEPROM

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

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

FREQ:  PUSH  A             ;SAVE "A"
       SBI   PINB,5        ;TOGGLE SPEAKER
FLUPE: DEC   A             ;SUBTRACT ONE FROM A
        BRNE FLUPE         ;WAIT UNTIL IT REACHES ZE
       POP   A             ;RESTORE "A"
         RET

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

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

This time the sound 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 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: RCALL EE_ERASE      ;ERASE EEPROM BYTE
       LDI   A,50          ;LOAD TONE #1
       RCALL EE_WRITE      ;WRITE IT TO EEPROM
```

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

```
EE_ERASE:
       SBIC EECR,EEWE       ;CHECK IF EEPROM AVAILABL
        RJMP EE_ERASE       ;LOOP-BACK IF NOT AVAILAB
       LDI B,0b0000_0001    ;SET EEPM0,EEPROM ERASE M
       OUT EECR,B           ;SET MODE TO ERASE
       OUT EEARL,ADR        ;EPROM ADDRESS
       OUT EEDR,A           ;EEPROM DATA TO WRITE
       SBI EECR,EEMWE       ;ENABLE EEPROM
       SBI EECR,EEWE        ;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,EEWE       ;CHECK IF EEPROM AVAILABL
        RJMP EE_WRITE       ;LOOP-BACK IF NOT AVAILAB
       LDI B,0b0000_0010    ;SET EEPM1, EEPROM WRITE
       OUT EECR,B           ;SET MODE TO WRITE ONLY
       OUT EEARL,ADR        ;EPROM ADDRESS
       OUT EEDR,A           ;EEPROM DATA TO WRITE
       SBI EECR,EEMWE       ;ENABLE EEPROM
       SBI EECR,EEWE        ;ENABLE WRITE
       INC ADR              ;INCREMENT EEPROM ADDRESS
        RET                 ;RETURN
```

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

```
.INCLUDE "M169DEF.INC"      ;AVR ATTINY13 DEFINITIONS
.DEF A       = R16          ;GENERAL PURPOSE ACCUMULA
.DEF B       = R18          ;GENERAL PURPOSE REGISTER
.DEF ADR     = R24          ;HOLDS EEPROM ADDRESS
```

```
.ORG $0000
RESET: LDI   A,LOW(RAMEND)  ;SET UP STACK...
       OUT   SPL,A          ;AT TOP OF MEMORY
       LDI   A,HIGH(RAMEND)
       OUT   SPH,A
       LDI A,128            ;SET SYS CLOCK SPEED
       STS CLKPR,A
       LDI A,3              ;0=8MHz 1=4MHz 2=2MHz 3=1
       STS CLKPR,A          ;BUTTERFLY @ 2MHZ AS SHIP
       SBI   DDRB,5         ;SET FOR OUTPUT TO SPEAKE
       CLR   ADR            ;MAKE SURE ADDRESS STARTS
MLUPE: RCALL EE_ERASE       ;ERASE EEPROM BYTE
       LDI   A,50
       RCALL EE_WRITE       ;WRITE IT TO EEPROM
       RCALL EE_ERASE
       LDI   A,150          ;LOAD TONE #2
       RCALL EE_WRITE       ;WRITE IT TO EEPROM

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

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

FREQ:  PUSH  A              ;SAVE "A"
       SBI   PINB,5         ;TOGGLE SPEAKER
FLUPE: DEC   A              ;SUBTRACT ONE FROM A
        BRNE FLUPE          ;WAIT UNTIL IT REACHES ZE
       POP   A              ;RESTORE "A"
        RET
```

```
EE_ERASE:
       SBIC EECR,EEWE       ;CHECK IF EEPROM AVAILABL
        RJMP EE_ERASE       ;LOOP-BACK IF NOT AVAILAB
       LDI B,0b0000_0001    ;SET EEPM0,EEPROM ERASE M
       OUT EECR,B           ;SET MODE TO ERASE
       OUT EEARL,ADR        ;EPROM ADDRESS
       OUT EEDR,A           ;EEPROM DATA TO WRITE
       SBI EECR,EEMWE       ;ENABLE EEPROM
       SBI EECR,EEWE        ;ENABLE ERASE
        RET                 ;RETURN

EE_WRITE:
       SBIC EECR,EEWE       ;CHECK IF EEPROM AVAILABL
        RJMP EE_WRITE       ;LOOP-BACK IF NOT AVAILAB
       LDI B,0b0000_0010    ;SET EEPM1, EEPROM WRITE
       OUT EECR,B           ;SET MODE TO WRITE ONLY
       OUT EEARL,ADR        ;EPROM ADDRESS
       OUT EEDR,A           ;EEPROM DATA TO WRITE
       SBI EECR,EEMWE       ;ENABLE EEPROM
       SBI EECR,EEWE        ;ENABLE WRITE
       INC ADR              ;INCREMENT EEPROM ADDRESS
        RET                 ;RETURN

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

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 OF "A"
       RCALL EE_READ        ;READ EEPROM LOCATION
       CPI  A,$FF           ;CHECK IF ITS ERASED
       MOV  A,B             ;RESTORE "A"
        BREQ EEE_XIT        ;IF ALREADY ERASED THEN E
EEE_WAIT:
       SBIC EECR,EEWE       ;CHECK IF EEPROM AVAILABL
        RJMP EEE_WAIT       ;LOOP-BACK IF NOT AVAILAB
       LDI B,0b0000_0001    ;SET EEPM0,EEPROM ERASE M
       OUT EECR,B           ;SET MODE TO ERASE
       OUT EEARL,ADR        ;EPROM ADDRESS
       OUT EEDR,A           ;EEPROM DATA TO WRITE
       SBI EECR,EEMWE       ;ENABLE EEPROM
       SBI EECR,EEWE        ;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 LOCATION
       CP   A,B             ;CHECK IF ALREADY PROGRAM
       MOV  A,B             ;RESTORE "A"
        BREQ EEW_XIT        ;ALREADY PROGRAMMED SO EX
EEW_WAIT:
       SBIC EECR,EEWE       ;CHECK IF EEPROM AVAILABL
        RJMP EEW_WAIT       ;LOOP-BACK IF NOT AVAILAB
       LDI B,0b0000_0010    ;SET EEPM1, EEPROM WRITE
       OUT EECR,B           ;SET MODE TO WRITE ONLY
       OUT EEARL,ADR        ;EPROM ADDRESS
       OUT EEDR,A           ;EEPROM DATA TO WRITE
       SBI EECR,EEMWE       ;ENABLE EEPROM
       SBI EECR,EEWE        ;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:

```
;-------------------------------------------;
; BFLY_EEPROM_ERASE, WRITE & READ SAVE TIME ;
;-------------------------------------------;

.INCLUDE "M169DEF.INC"      ;AVR ATTINY13 DEFINITIONS
.DEF A       = R16          ;GENERAL PURPOSE ACCUMULA
.DEF B       = R18          ;GENERAL PURPOSE REGISTER
.DEF ADR     = R24          ;HOLDS EEPROM ADDRESS

.ORG $0000
RESET: LDI   A,LOW(RAMEND)  ;SET UP STACK...
       OUT   SPL,A          ;AT TOP OF MEMORY
       LDI   A,HIGH(RAMEND)
       OUT   SPH,A
       LDI   A,128          ;SET SYS CLOCK SPEED
       STS   CLKPR,A
       LDI   A,3            ;0=8MHz 1=4MHz 2=2MHz 3=1
       STS   CLKPR,A        ;BUTTERFLY @ 2MHZ AS SHIP
       SBI   DDRB,5         ;SET FOR OUTPUT TO SPEAKE
       CLR   ADR            ;MAKE SURE ADDRESS STARTS
MLUPE: RCALL EE_ERASE       ;ERASE EEPROM BYTE
       LDI   A,100          ;LOAD
       RCALL EE_WRITE       ;WRITE IT TO EEPROM
       INC   ADR
         RCALL EE_ERASE
       LDI   A,250          ;LOAD TONE #2
       RCALL EE_WRITE       ;WRITE IT TO EEPROM

PLAY_LOOP:
       CLR  ADR             ;START READS AT ZERO
       RCALL EE_READ        ;READ EEPROM INTO "A"
       RCALL HOLD_TONE      ;PLAY TONE
       INC   ADR
       RCALL EE_READ        ;READ EEPROM INTO "A"
       RCALL HOLD_TONE      ;PLAY TONE
         RJMP PLAY_LOOP     ;LOOP-BACK DO IT AGAIN

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

FREQ:  PUSH  A              ;SAVE "A"
       SBI   PINB,5         ;TOGGLE SPEAKER
FLUPE: DEC   A              ;SUBTRACT ONE FROM A
        BRNE FLUPE          ;WAIT UNTIL IT REACHES ZE
       POP   A              ;RESTORE "A"
        RET

EE_ERASE:
       MOV B,A              ;PRESERVE VALUE OF "A"
       RCALL EE_READ        ;READ EEPROM LOCATION
       CPI A,$FF            ;CHECK IF ITS ERASED
       MOV A,B              ;RESTORE "A"
        BREQ EEE_XIT        ;ALREADY ERASEED SO EXIT
EEE_WAIT:
       SBIC EECR,EEWE       ;CHECK IF EEPROM AVAILABL
        RJMP EEE_WAIT       ;LOOP-BACK IF NOT AVAILAB
       LDI B,0b0000_0001    ;SET EEPM0,EEPROM ERASE M
       OUT EECR,B           ;SET MODE TO ERASE
       OUT EEARL,ADR        ;EPROM ADDRESS
       OUT EEDR,A           ;EEPROM DATA TO WRITE
       SBI EECR,EEMWE       ;ENABLE EEPROM
       SBI EECR,EEWE        ;ENABLE ERASE
EEE_XIT: RET                ;RETURN

EE_WRITE:
       MOV B,A              ;PRESERVE "A"
       RCALL EE_READ        ;READ EPROM LOCATION
       CP  A,B              ;ALREADY PROGRAMMED?
       MOV A,B              ;RESTORE "A"
        BREQ EEW_XIT        ;ALREADY PROGRAMMED SO EX
EEW_WAIT:
       SBIC EECR,EEWE       ;CHECK IF EEPROM AVAILABL
        RJMP EEW_WAIT       ;LOOP-BACK IF NOT AVAILAB
       LDI B,0b0000_0010    ;SET EEPM1, EEPROM WRITE
       OUT EECR,B           ;SET MODE TO WRITE ONLY
       OUT EEARL,ADR        ;EPROM ADDRESS
       OUT EEDR,A           ;EEPROM DATA TO WRITE
       SBI EECR,EEMWE       ;ENABLE EEPROM
       SBI EECR,EEWE        ;ENABLE WRITE
EEW_XIT: RET                ;RETURN

EE_READ:
       SBIC EECR,EEWE       ;CHECK IF EEPROM BUSY
        RJMP EE_READ        ;ITS BUSY SO WE WAIT
       OUT EEARL,ADR        ;SET-UP THE ADDRESS
       SBI EECR,EERE        ;SET-UP TO READ
       IN  A,EEDR           ;READ THE DATA REGISTER
        RET

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

EE_READ:
       SBIC EECR,EEPE       ;CHECK IF EEPROM BUSY
        RJMP EE_READ        ;ITS BUSY SO WE WAIT
       OUT EEARL,ADR        ;SET-UP THE ADDRESS
       SBI EECR,EERE        ;SET-UP TO READ
       IN  A,EEDR           ;READ THE DATA REGISTER
        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 ATmEGA169 Data Sheet tells us that the EEPROM Ready Interrupt is at $0028:

```
.ORG $0000
       RJMP RESET           ;RESET START VECTOR
.ORG $0028
       RJMP EE_RDY          ;EEPROM READY INTERRUPT
```

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 COUNTER
       CPI N,16             ;TEN LOOPS YET?
        BRNE PLAY_LOOP      ;NO, SKIP
       SEI                  ;ACTIVATE INTERRUPTS GLOB
```

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 STACK
        PUSH B               ;SAVE "B" ON STACK
        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 "M169DEF.INC"      ;AVR ATTINY13 DEFINITIONS
.DEF A       = R16          ;GENERAL PURPOSE ACCUMULA
.DEF B       = R18          ;GENERAL PURPOSE REGISTER
.DEF N       = R20          ;COUNTER
```

```
.DEF ADR     = R28          ;HOLDS EEPROM ADDRESS

.ORG $0000
       RJMP RESET           ;RESET START VECTOR
.ORG $0028
       RJMP EE_RDY          ;EEPROM READY INTERRUPT

RESET: LDI   A,LOW(RAMEND)  ;SET UP STACK...
       OUT   SPL,A          ;AT TOP OF MEMORY
       LDI   A,HIGH(RAMEND)
       OUT   SPH,A
       LDI   A,128          ;SET SYS CLOCK SPEED
       STS   CLKPR,A
       LDI   A,3            ;0=8MHz 1=4MHz 2=2MHz 3=1
       STS   CLKPR,A        ;BUTTERFLY @ 2MHZ AS SHIP
       SBI   DDRB,5         ;SET FOR OUTPUT TO SPEAKE
MLUPE: CLI                  ;SHUT DOWN ANY INTERRUPTS
       CLR   ADR            ;MAKE SURE ADDRESS STARTS
       CLR   N              ;COUNTER FOR LOOP
       LDI   A,50           ;LOAD TONE #1
       RCALL EE_ERASE       ;ERASE EEPROM BYTE
       RCALL EE_WRITE       ;WRITE IT TO EEPROM
       INC ADR              ;INCREMENT OUR ADDRESS
       LDI   A,250          ;LOAD TONE #2
       RCALL EE_ERASE       ;ERASE EEPROM BYTE
       RCALL EE_WRITE       ;WRITE IT TO EEPROM

PLAY_LOOP:
       CLR  ADR             ;START READS AT ZERO
       RCALL EE_READ        ;READ EEPROM INTO "A"
       INC  ADR             ;INCREMENT OUR ADDRESS
       RCALL HOLD_TONE      ;PLAY TONE
       RCALL EE_READ        ;READ EEPROM INTO "A"
       INC  ADR             ;INCREMENT OUR ADDRESS
       RCALL HOLD_TONE      ;PLAY TONE
       INC N                ;INCREMENT LOOP COUNTER
       CPI N,16             ;TEN LOOPS YET?
        BRNE PLAY_LOOP      ;NO, SKIP
       SEI                  ;ACTIVATE INTERRUPTS GLOB
        RJMP PLAY_LOOP      ;LOOP-BACK DO IT AGAIN
```

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

FREQ:  PUSH  A             ;SAVE "A"
       SBI   PINB,5        ;TOGGLE SPEAKER
FLUPE: DEC   A             ;SUBTRACT ONE FROM A
        BRNE FLUPE         ;WAIT UNTIL IT REACHES ZE
       POP   A             ;RESTORE "A"
            RET

EE_RDY: PUSH A             ;SAVE "A" ON STACK
        PUSH B             ;SAVE "B" ON STACK
        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 REGISTER
        POP B              ;RESTORE "B"
        POP A              ;RESTORE "A"
         RETI

EE_ERASE:
       MOV  B,A            ;PRESERVE VALUE OF "A"
       RCALL EE_READ       ;READ EEPROM LOCATION
       CPI  A,$FF          ;CHECK IF ITS ERASED
       MOV  A,B            ;RESTORE "A"
        BREQ EEE_XIT       ;IF ALREADY ERASED THEN E
       SBIC EECR,EEWE      ;CHECK IF EEPROM AVAILABL
        RJMP EE_WRITE      ;LOOP-BACK IF NOT AVAILAB
       LDI B,0b0000_1001   ;SET EEPM0,EEPROM ERASE M
       OUT EECR,B          ;SET MODE TO ERASE
       OUT EEARL,ADR       ;EPROM ADDRESS
       OUT EEDR,A          ;EEPROM DATA TO WRITE
       SBI EECR,EEMWE      ;ENABLE EEPROM
       SBI EECR,EEWE       ;ENABLE ERASE
```

```
EEE_XIT: RET                ;RETURN

EE_WRITE:
       MOV  B,A             ;PRESERVE "A"
       RCALL EE_READ        ;READ EEPROM LOCATION
       CP   A,B             ;CHECK IF ALREADY PROGRAM
       MOV  A,B             ;RESTORE "A"
        BREQ EEW_XIT        ;ALREADY PROGRAMMED SO EX
       SBIC EECR,EEWE       ;CHECK IF EEPROM AVAILABL
        RJMP EE_WRITE       ;LOOP-BACK IF NOT AVAILAB
       LDI B,0b0000_1010    ;SET EEPM1, EEPROM WRITE
       OUT EECR,B           ;SET MODE TO WRITE ONLY
       OUT EEARL,ADR        ;EPROM ADDRESS
       OUT EEDR,A           ;EEPROM DATA TO WRITE
       SBI EECR,EEMWE       ;ENABLE EEPROM
       SBI EECR,EEWE        ;ENABLE WRITE
EEW_XIT: RET                ;RETURN

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

SOME PRECAUTIONS

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

```
SPM_BUSY:
       IN   B,SPMCSR        ;CHECK IF AN SPM COMMAND
       ANDI B,0b0000_0001   ;WAIT SPM ENABLE (SPMEN)
        BRNE SPM_BUSY
```

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

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

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 COMMAND
       ANDI B,0b0000_0001   ;WAIT SPM ENABLE (SPMEN)
        BRNE SPM_BUSY
EE_BUSY:
       SBIC EECR,EEWE       ;CHECK IF EEPROM AVAILABL
        RJMP EE_WRITE       ;LOOP-BACK IF NOT AVAILAB
       LDI B,0b0000_0000    ;SET EEPM0,EEPM1
       OUT EECR,B           ;SET MODE TO ERASE & WRIT
       OUT EEARL,ADR        ;EPROM ADDRESS
       OUT EEDR,A           ;EEPROM DATA TO WRITE
       CLI                  ;SHUT-DOWN INTERRUPTS
       SBI EECR,EEMWE       ;ENABLE EEPROM
       SBI EECR,EEWE        ;ENABLE WRITE
       SEI                  ;RE-ENABLE INTERRUPTS
       INC ADR              ;INCREMENT EEPROM ADDRESS
        RET                 ;RETURN
```

Comments

You do not have permission to add comments.

Sign in | Recent Site Activity | Report Abuse | Print Page | Powered By **Google Sites**