AVR ASM INTRODUCTION

Search this site

AVR ASSEMBLER TUTOR >

3b. BUTTERFLY ADC

A MORON'S GUIDE TO ADC ON THE BUTTERFLY v2.3

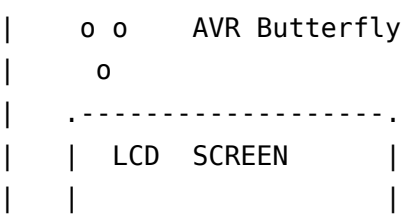
by RetroDan@GMail.com

TABLE OF CONTENTS:

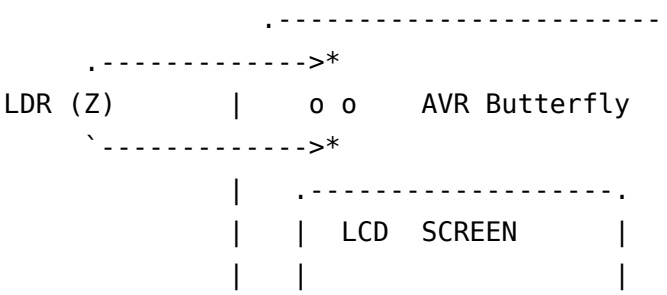
The purpose of this tutorial is to learn about Analog-to-Digital conversions by using a Light Dependant Resistor connected to the Butterfly Board. LDRs are also refered to as CDS (Cadmium DiSuflide) Cells. With this we construct my version of the Theremin Aerophone, an instrument used in classic science fiction movies of the past.

If you are lucky enough to have an older Butterfly Board with an LDR then you can skip the next section on installing one. I was sad to see that my newer batch came without an LDR attached. You could also attach a variable resister (Potentiometer) in place of the LDR.

I misplaced my collection of LDRs so I resorted to pulling them out of $1 automatic night lights I purchased at the local dollar store. To connect it to the Butterfly Board, look at the top left-hand side of the board and at the top you will see four empty holes in a circle above the LCD screen:



We are going to solder our LDR to the holes in the 6 & 12 O'Clock positions:



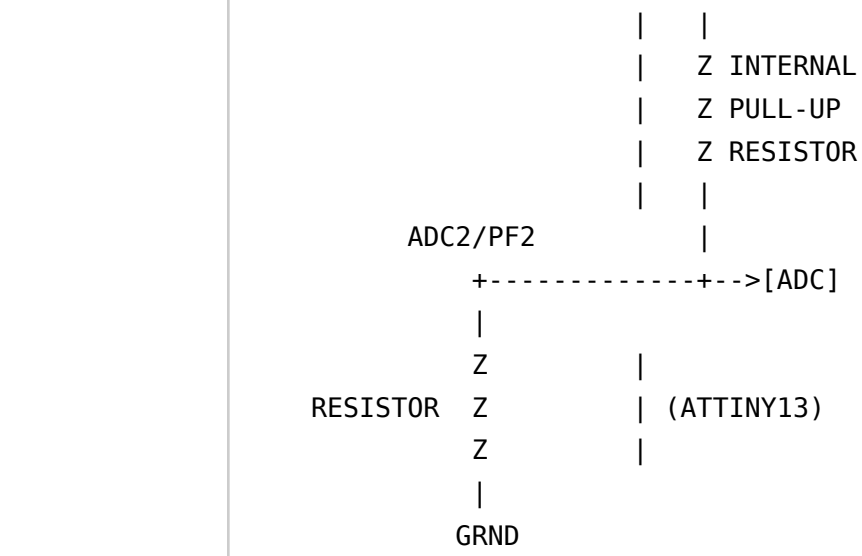
A Light Dependent Resistor (LDR) is a device that changes its resistance depending on how much light enters its active area. To read that resistance we are going to use the Analog-to-Digital (ADC) in the Butterfly's ATMega169 chip. If you can't get an LDR you can substitute a variable resistor (Potentiometer).

CHAPTER 1: MEASURING RESISTANCE WITH ADC:

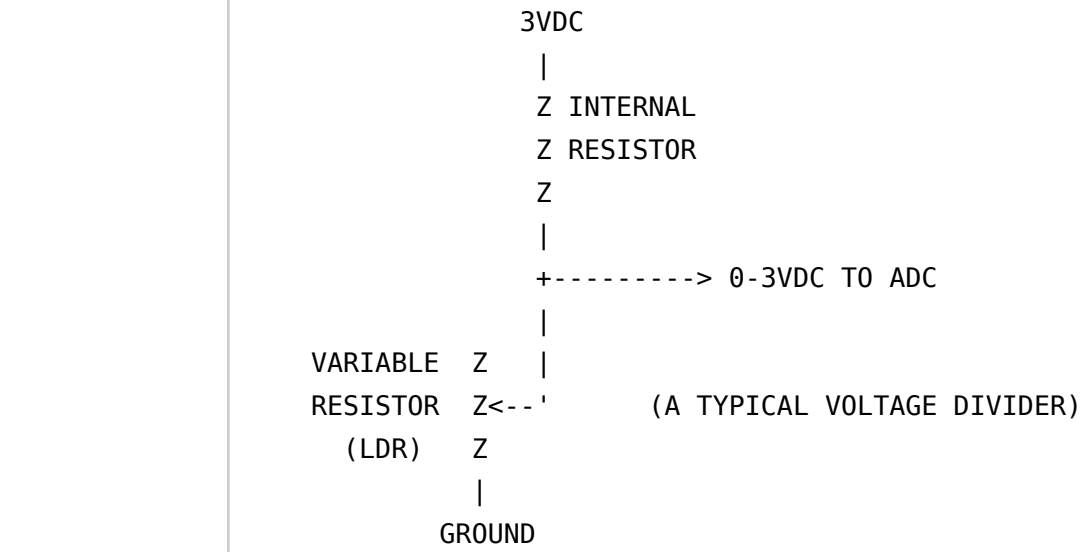
An Analog to Digital Converter (ADC) can turn an analog signal into a digital one. It does this by measuring voltage on its input pin. Not only can they convert an analog signal like music into a digital format, but they can also be used to measure resistance.

ADCs are not mysterious, they often work by measuring how long it takes to charge a capacitor, keeping track of time with a counter. The result is a digital number that correlates to the applied voltage at the input.

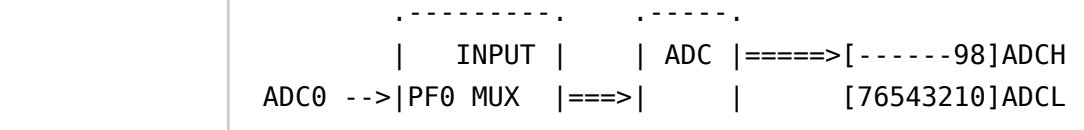
If we connect a resistor between an ADC input pin to ground and if the internal resistor is active we get a cicuit as follows:



The two resistors form a voltage divider and the voltage at the input pin will be dependant on the value of our variable resistor:



Below is a block diagram of the ADC in the Butterly. A multiplexer (MUX) selects one of eight inputs to be fed into the ADC. A prescaler determines the speed of the conversion. The output is ten bits wide, so the lowest eight bits go into the ADCL register and the ADCH register holds the remaining high bits in its low end:



```
ADC1 -->|PF1      |    |     |
ADC2 -->|PF2      |    |     |<-- TIMER-PRESCALER
ADC3 -->|PF3      |    `-----'
ADC4 -->|PF4      |
ADC5 -->|PF5      |<------------- INPUT-SELCTOR
ADC6 -->|PF6      |
ADC7 -->|PF7      |
        |         |
        `---------'
```

We will use the ADC2 input that is connected from PORTF2 (PF2) to our LDR.

Let us first get the Butterfly to make a noise, then build upon that. Below is a program listing that will produce a tone on the Butterfly internal speaker (A full explanation follows the listing):

```
.INCLUDE "M169DEF.INC"

.DEF A       = R16
.DEF I       = R21

.ORG $0000

RESET: LDI   A,HIGH(RAMEND) ;SETUP THE STACK POINTER
       OUT   SPH,A          ;AT TOP OF MEMORY AND
       LDI   A,LOW(RAMEND)  ;GROW DOWNWARDS
       OUT   SPL,A
       SBI   DDRB,5         ;CONFIG SPEAKER PORT
MLUPE:
       RCALL BPAUSE         ;WAIT
       SBI   PINB,5         ;CLICK THE SPEAKER
        RJMP MLUPE          ;DO IT AGAIN

BPAUSE:                     ;PAUSE ROUTINE
BLOOP:  LDI I,20            ;TIME DEPENDS ON "A"
BPLUPE: DEC I
         BRNE BPLUPE
        DEC  A
         BRNE BLOOP
          RET
```

The .INCLUDE command tell the assembler to load in the definitions for the ATMega169 MCU chip on the Butterfly. The following .DEF statements define the registers that we use:

```
.INCLUDE "M169DEF.INC"
.DEF A       = R16
.DEF I       = R21
```

The .ORG command tell the assembler where in memory we wish to place oour program. Since we are not using any interrupts we can start at the bottom of memory:

```
.ORG $0000
```

Next we tell the system to set up a stack at the top of memory, so we can use subroutines:

```
RESET: LDI   A,HIGH(RAMEND) ;SETUP THE STACK POINTER
       OUT   SPH,A          ;AT TOP OF MEMORY AND
       LDI   A,LOW(RAMEND)  ;GROW DOWNWARDS
       OUT   SPL,A
```

Before we can use the speaker on PORTB5, we have to configure it for output by writing a one to its Data Direction Register (DDRB):

```
       SBI   DDRB,5         ;CONFIG SPEAKER PORT
```

The main loop of the program calls a pause routine, then it toggles the ouput on PORTB5 which is connected to the Butterfly speaker:

```
MLUPE:
       RCALL BPAUSE         ;WAIT
       SBI   PINB,5         ;CLICK THE SPEAKER
        RJMP MLUPE          ;DO IT AGAIN
```

The pause routine is a loop within a loop that slows things down enough to a frequency that we can hear through the speaker:

```
BPAUSE:                     ;PAUSE ROUTINE
BLOOP:  LDI I,20            ;TIME DEPENDS ON "A"
BPLUPE: DEC I
         BRNE BPLUPE
        DEC  A
         BRNE BLOOP
```

```
        RET
```

CHAPTER 2: WAIT FOR CONVERSION METHOD:

Our strategy is to start an Analog-to-Digital Conversion on ADC2, then poll a flag that tells us when the conversion is complete. Then we produce a sound on the speaker and pause for a length of time that is determined by the value of voltage/resistance that was measured. The result is a changing frequency of sound based on our input from the LDR:

The first addition to our program is to tell the ADC Multiplxer that we wish to read the LDR. The data-sheets tell us that to select ADC3 we need to set the MUX to two:

```
       LDI   A,0b0000_0010 ;SELECT INPUT 0=TEMP,2=LIG
       STS   ADMUX,A
```

Here activate the pull-up resister by writing a one to PORTF3:

```
       SBI   PORTF,PORTF3   ;INTIALIZE PORT F
```

The data-sheets tell us that the ADC works best at a frequency between 50Khz and 200 Khz. So we select a pre-scaler/divider of sixteen because 2Mhz divided by sixteen give us a frequency of 125Khz. At the same time we set the ADEN bit to enable the ADC and we also set the ADSC bit to start the conversion process:

```
MLUPE:
       LDI   A,0b1100_0100   ;ENABLE, START & SET PRE
       STS   ADCSRA,A        ;START ANALOG TO DIGITAL
```

When the conversion is complete the ADIF flag of ASCRA is set to one, so we wait for it with:

```
WAIT4:
       LDS   A,ADCSRA        ;WAIT FOR ADC CONVERSION
       ANDI  A,0b001_0000    ;(ADIF)
        BREQ WAIT4
```

Now that the conversion process is complete we read the results from the ADCL & ADCH registers. The result is a ten bit number with its

lowest eight bits in ADCL and the remaiing two bits in ADCH, which we are going to ignore. ADCL must be read first followed by the ADCH to work properly. We still have to read ADCH even though we don't use the result:

```
       LDS   A, ADCL          ;MUST READ ADCL BEFORE A
       LDS   AH, ADCH
```

After we make these additions to our program it becomes:

```
.INCLUDE "M169DEF.INC"

.DEF A       = R16
.DEF AH      = R17
.DEF I       = R21

.ORG $0000

RESET: LDI   A,HIGH(RAMEND)  ;SETUP THE STACK POINTER
       OUT   SPH,A           ;AT TOP OF MEMORY AND
       LDI   A,LOW(RAMEND)   ;GROW DOWNWARDS
       OUT   SPL,A
       SBI   DDRB,5          ;CONFIG SPEAKER PORT
       LDI   A,0b0000_0010   ;SELECT INPUT 0=TEMP,2=L
       STS   ADMUX,A
       SBI   PORTF,PORTF3    ;INTIALIZE PORT F

MLUPE:
       LDI   A,0b1100_0100   ;ENABLE, START & SET PRE
       STS   ADCSRA,A        ;START ANALOG TO DIGITAL
WAIT4:
       LDS   A,ADCSRA        ;WAIT FOR ADC CONVERSION
       ANDI  A,0b001_0000    ;(ADIF)
        BREQ WAIT4
       LDS   A,ADCL          ;MUST READ ADCL BEFORE A
       LDS   AH,ADCH
       RCALL BPAUSE          ;WAIT
       SBI   PINB,5          ;CLICK THE SPEAKER
        RJMP MLUPE           ;DO IT AGAIN

BPAUSE:                      ;PAUSE ROUTINE
```

```
BLOOP:  LDI I,10             ;TIME DEPENDS ON "A"
BPLUPE: DEC I
         BRNE BPLUPE
        DEC  A
         BRNE BLOOP
          RET
```

If you programmed your Butterfly and connect the LDR as described. You can wave your hand over the LDR and the speaker will produce a sound reminiscent of sci-fi movies of the 1950s.

As the amount of light hitting the LDR changes, the resistance of the LDR will change, causing the voltage at our input to the ADC to shift. This voltage change will be converted into a digital value by the ADC. We read this value into the "A" Register and produce a varying frequency to our output speaker, by varying the length of time we spend in our PAUSE routine. The result is a musical intrument that we control by waving our hand over the circuit.

CHAPTER 3: FREE RUNNING MODE

With the Free-Running Method, the ADC is set to self-trigger after each conversion. We don't wait for the conversion to complete (and the ADIF Flag to be set). By reading the ADC output registers in free-running mode, we can pick-up the conversion value from its most recent reading.

This time we turn on the ADC with the ADC Enable bit of the ADC Control and Status Register (ADCSRA). We tell the system we want the Automatic Update by setting the ADATE bit, and we start the conversion process by setting the ADCS bit. The lower two bit set our pre-scaler/divider to divide-by-sixteen:

```
       LDI   A,0b1110_0100  ;[ADEN,ADSC,ADATE,ADIF,_,
       OUT   ADCSRA,A       ;START ANALOG TO DIGITAL
```

Since the ADC will run on its own, we remove the set-up of ADCSRA from the main loop by moving the MLUPE label down. We can eliminate the parts of our program that wait for the ADIF flag to be set. After we remove that part of the code and make the above changes your program should look like this:

```
.INCLUDE "M169DEF.INC"

.DEF A       = R16
.DEF AH      = R17
.DEF I       = R21

.ORG $0000
       RJMP RESET

RESET: LDI   A,HIGH(RAMEND)  ;SETUP THE STACK POINTER
       OUT   SPH,A           ;AT TOP OF MEMORY AND
       LDI   A,LOW(RAMEND)   ;GROW DOWNWARDS
       OUT   SPL,A
       SBI   DDRB,5          ;CONFIG SPEAKER PORT
       LDI   A,0b0000_0010   ;SELECT INPUT 0=TEMP,2=L
       STS   ADMUX,A
       SBI   PORTF,PORTF3    ;INTIALIZE PORT F
       LDI   A,0b1110_0100   ;ENABLE, START & SET PRE
       STS   ADCSRA,A        ;START ANALOG TO DIGITAL

MLUPE:
       LDS   A, ADCL         ;MUST READ ADCL BEFORE A
       LDS   AH, ADCH
       RCALL BPAUSE          ;WAIT
       SBI   PINB,5          ;CLICK THE SPEAKER
        RJMP MLUPE           ;DO IT AGAIN

BPAUSE:                      ;PAUSE ROUTINE
BLOOP:  LDI I,10             ;TIME DEPENDS ON "A"
BPLUPE: DEC I
         BRNE BPLUPE
        DEC  A
         BRNE BLOOP
          RET
```

CHAPTER 4: THE LEFT ADJUSTED OUTPUT MODE

For this version, we are using the free-running mode but we have the

system automatically left-shift our result into the ADCH register. To put the system into left-shift mode we set the ADLAR bit of the ADMUX Register to one. Note: the contents of the ADMUX Register only take effect AFTER the ADC has been activated by the ADEN bit of the ADCSRA Register, so we place this part of our code AFTER we set the ADCSRA Register:

```
     LDI   A,0b0010_0010   ;SET TO LEFT SHIFT MODE &
     OUT   ADMUX,A
```

In our previous versions of the program the ten-bit result was right shifted into the ADCL register and the two high bits were stored in the ADCH register, which we never used. This time we are going to have the result let-shifted into the ADCH register and ignore the lower two-bits of our ten-bit conversion, which are stored in the ADCL register.

```
               ADCH:             ADCL
PREVIOUSLY:    [-,-,-,-,-,-,9,8] [7,6,5,4,3,2,1,0]
THIS TIME:     [9,8,7,6,5,4,3,2] [1,0,-,-,-,-,-,-] <-
```

So as you can see below, we read the low byte into A then we ignore the result and load the high byte into A the result being the highest eight bits of our result ignoring the lowest two bits:

```
       IN    A,ADCL          ;MUST READ ADCL BEFORE AD
       IN    A,ADCH          ;
```

The results of these changes is the program below:

```
.INCLUDE "M169DEF.INC"

.DEF A       = R16
.DEF I       = R21

.ORG $0000
       RJMP RESET

RESET: LDI   A,HIGH(RAMEND) ;SETUP THE STACK POINTER
       OUT   SPH,A          ;AT TOP OF MEMORY AND
       LDI   A,LOW(RAMEND)  ;GROW DOWNWARDS
       OUT   SPL,A
```

```
       SBI   DDRB,5         ;CONFIG SPEAKER PORT
       LDI   A,0b0010_0010  ;SELECT INPUT 0=TEMP,2=LI
       STS   ADMUX,A
       SBI   PORTF,PORTF3   ;INTIALIZE PORT F
       LDI   A,0b1110_0100  ;ENABLE, START & SET PRES
       STS   ADCSRA,A       ;START ANALOG TO DIGITAL

MLUPE:
       LDS   A, ADCL        ;MUST READ ADCL BEFORE AD
       LDS   A, ADCH        ;WE USE THE RESULT IN ADC
       RCALL BPAUSE         ;WAIT
       SBI   PINB,5         ;CLICK THE SPEAKER
        RJMP MLUPE          ;DO IT AGAIN

BPAUSE:                     ;PAUSE ROUTINE
BLOOP:  LDI I,10            ;TIME DEPENDS ON "A"
BPLUPE: DEC I
         BRNE BPLUPE
        DEC  A
         BRNE BLOOP
          RET
```

You should notice a change in tone since the reading we are taking now drops the two lowest bits.

CHAPTER 5: THE INTERRUPT METHOD

This time we are going to read the results and generate a sound from inside an interrupt routine that is called automatically as each conversion is complete. This leaves the system free to do other tasks. For this example the main loop of the program does nothing but jump to itself:

```
RLOOP: RJMP RLOOP
```

When interrupts are enabled with the SEI command, they system looks to the bottom of memory .ORG $0000 for a jump table to the various interrupts. The power-on/reset jump vector is the first one at .ORG $0000 to we point it to our main program.

```
.ORG $0000
```

```
       RJMP RESET
```

If we consult the data-sheets we find that the interrupt vector for the ADC is located at $0026, so we put a jump to our interrupt routine there:

```
.ORG $0026
       RJMP ANA_CONV
```

And we tell the system to enable interrupts with the Set Enable Interrup command SEI:

```
            SEI                ;ENABLE INTERRUPTS GL
```

Now we are going to set the ADC Status & Control Register the same as last time except we are going to set the ADC Interupt Enable bit ADIE to one also. This means the ADC Enable bit ADEN is set; the ADC Start Conversion bit ADSC is set; the Automatic Update bit ADATE is set; the ADC Interupt Enable ADIE bit is set; and the pre-scaler/divider is set to divide-by-sixteen:

```
       LDI   A,0b1110_1100  ;[ADEN,ADSC,ADATE,ADIF,_,
       OUT   ADCSRA,A       ;START ANALOG TO DIGITAL
```

When we service an interrupt with an interrupt routine, it is good practice to save off the system status and the value of any registers that we use. Here we save the system status and contents of "A" register on the stack and later restore them.

```
ANA_CONV:
     PUSH A               ;SAVE CONTENTS OF "A" REGI
     PUSH AH              ;SAVE CONTENTS OF "AH" REG
     IN   A,SREG          ;SAVE THE SYSTEM STATUS FO
     PUSH A
```

In the heart of our interrupt routine we read the results of the ADC conversion by reading the lower byte ADCL followed by the high byte ADCH. Then we call the PAUSE routine, followed by toggling our output bit connected to our speaker:

```
      IN    A,ADCL         ;MUST READ ADCL BEFORE ADC
      IN    AH,ADCH        ;REQUIRED, THOUGH NOT USED
      RCALL PAUSE          ;VARIABLE TIME DELAY BASED
      SBI   PINB,0         ;TOGGLE SPEAKER ON PORTB0
```

After restoring the registers by popping them from the stack, we end our interrupt routine with a Return from Interrupt command RETI:

```
          POP A               ;RESTORE SYSTEM STATUS
          OUT SREG,A
          POP AH              ;RESTORE "AH" RESGISTER
          POP A               ;RESTORE "A" REGISTER
           RETI
```

After making these changes our complete program becomes:

```
.INCLUDE "M169DEF.INC"

.DEF A       = R16
.DEF AH      = R17
.DEF I       = R21

.ORG $0000
       RJMP RESET
.ORG $0026
       RJMP ANA_CONV

RESET: LDI   A,HIGH(RAMEND) ;SETUP THE STACK POINTER
       OUT   SPH,A          ;AT TOP OF MEMORY AND
       LDI   A,LOW(RAMEND)  ;GROW DOWNWARDS
       OUT   SPL,A
       SBI   DDRB,5         ;CONFIG SPEAKER PORT
             LDI   A,0b0010_0010  ;SELECT INPUT 0=TEM
       STS   ADMUX,A
       SBI   PORTF,PORTF3   ;INTIALIZE PORT F
       LDI   A,0b1110_1100  ;ENABLE, START & SET PRES
       STS   ADCSRA,A       ;START ANALOG TO DIGITAL
        SEI                 ;ACTIVATE INTERRUPTS GLOB

RLUPE:  RJMP RLUPE          ;DO NOTHING LOOP

ANA_CONV:
              PUSH  A             ;SAVE OFF REGISTER
              PUSH  AH            ;TO STACK
              IN    A,SREG
         PUSH  A
```

```
          LDS   A, ADCL         ;MUST READ ADCL BEFORE
               LDS   A, ADCH         ;WE USE THE RESULT
               RCALL BPAUSE          ;WAIT
               SBI   PINB,5          ;CLICK THE SPEAKER
          POP   A               ;RESTORE REGISTERS
          OUT   SREG,A
          POP   AH
          POP   A
          RETI                  ;RETURN FROM INTERRUPT

BPAUSE:                         ;PAUSE ROUTINE
BLOOP:  LDI I,10                ;TIME DEPENDS ON "A"
BPLUPE: DEC I
          BRNE BPLUPE
         DEC  A
          BRNE BLOOP
           RET
```

Comments

You do not have permission to add comments.