AVR ASM INTRODUCTION

Search this site

AVR ASSEMBLER TUTOR >

3a. USING THE ADC

A MORON'S GUIDE TO ADC v2.4

by RetroDan@GMail.com

TABLE OF CONTENTS:

An Analog to Digital Converter (ADC) is used to 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.

For this primer we use the AVR ATtiny13 for its simplicity & small number of pins. This chip has one ADC which can be used to read up to four inputs. The ATtiny13 runs at 1.2MHz (9.6MHz Oscillator divided by 8) with 1K of RAM. The example programs should run on the ATtiny 13, ATtiny25, ATtiny45 or ATtiny85. The circuits & code should be easy to adapt to most any AVR Core chips.

The Pin outs of the ATtiny13 chip are:

```
                ATTINY13
                .------.
ADC0/PB5 --|1 A 8|-- Vcc
```

```
ADC3/PB3 ->|2 T 7|-- PB2/ADC1
ADC2/PB4 --|3 N 6|-- PB1
     GND --|4 Y 5|-->PB0
           `-----'
```

From the diagram we can see that the ADC inputs are on pins 1, 7, 2 & 3 or PortB5, PortB2, PortB3 & PortB4. We will be using ADC3/PortB3 (Pin 2) for our input and Port B0 (Pin 5) for output.

CHAPTER 1: MEASURING RESISTANCE WITH ADC:

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

```
                      Vcc
                    |   |
                    |   Z INTERNAL
                    |   Z PULL-UP
                    |   Z RESISTOR
                    |   |
       ADC3 PIN         |
           +------------+-->[ADC]
           |
           Z         |
RESISTOR   Z         | (ATTINY13)
           Z         |
           |
          GRND
```

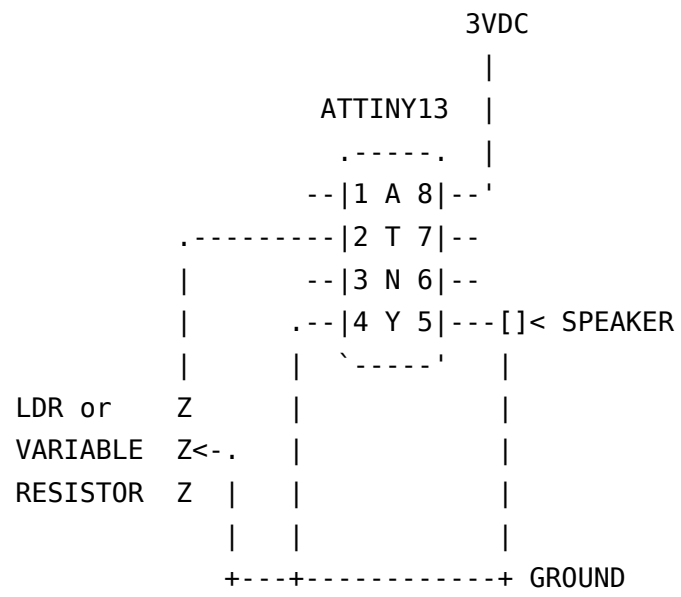
The two resistors form a voltage divider and the voltage at the input pin will be dependent on the value of our variable resistor:

```
              3VDC
               |
               Z INTERNAL
               Z RESISTOR
               Z
               |
               +---------> 0-3VDC TO ADC
               |
```

```
VARIABLE  Z   |
RESISTOR  Z<--'      (A TYPICAL VOLTAGE DIVIDER)
          Z
          |
        GROUND
```

The circuit we use is the one below. We connect a Light Dependent Resistor (LDR) between pin 2 (PortB3) and ground. An LDR varies its resistance based on how much light enters the device. LDRs are also known as Cadmium DiSulfide Cells (CDS). I pulled one from an automatic night light I picked up at the dollar store. If you don't have an LDR you can substitute a variable resistor (Potentiometer).

For output connect a small speaker on pin 5 (PortB0) to ground. If you are using more than 3 Volts then put a 100-220 Ohm resistor between the speaker and ground:



We are constructing my version of a Theremin Aetherphone like the ones used in classic science fiction films of the 1950s.

Below is a block diagram of the ADC in the ATtiny13. A multiplexer (MUX) selects one of four 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:

```
        .-------.    .-----.
        | INPUT |    | ADC |=====>[------98]A
ADC0 -->|  MUX  |===>|     |      [76543210]A
ADC1 -->|       |    |     |
ADC2 -->|       |    |     |<-- TIMER-PRESCAL
ADC3 -->|       |    `-----'
        |       |<------------- INPUT-SELECTO
        `-------'
```

CHAPTER 2: WAIT FOR CONVERSION METHOD:

Our first program is listed below. An explanation follows the listing. Our strategy is to start an Analog-to-Digital Conversion on ADC3, 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:

```
;--------------------------------------;
; A BASIC THEREMIN AETHERPHONE         ;
; PLAYS NOTES ACCORDING TO THE LIGHT   ;
; ATTINY13 VERSION, RETRODAN@GMAIL.COM ;
;--------------------------------------;

.include "TN13DEF.INC" ;AVR ATTINY13 DEFINITI

.DEF A       = R16
.DEF AH      = R17
.DEF B       = R18
.DEF C       = R20

.ORG $0000

RESET: SBI   PORTB,3        ;ENABLE PULL-UP R
       SBI   DDRB,0         ;SET PORTB0 FOR O
RLOOP:
       LDI   A,0b1100_0011  ;ENABLE, START CO
```

```
        OUT   ADCSRA,A
        LDI   A,0b0000_0011  ;SELECT ADC INPUT
        OUT   ADMUX,A

WAIT:   IN    A,ADCSRA       ;READ THE STATUS
        ANDI  A,0b0001_0000  ;CHECK FLAG (1<
```

First we tell the assembler to load in the definitions for the chip we are using. The .DEF statements are our own definitions for the registers we use:

```
.include "TN13DEF.INC" ;AVR ATTINY13 DEFINITI

.DEF A       = R16
.DEF AH      = R17
.DEF B       = R18
.DEF C       = R20
```

Next we tell the assembler where in memory to place our program. Since we are not using any interrupts, we can start our program at zero, the bottom of program flash memory:

```
.ORG $0000
```

We are using PortB3 (ADC3) as our input. Previously we shut off the chip reset function of this pin. Now we write a one to it in order to activate its pull-up resistor. We also set the bit in the Data Direction Register for Port B (DDRB) that we will be using as output to our speaker:

```
RESET: SBI   PORTB,5        ;ENABLE PULL-UP R
       SBI   DDRB,0         ;SET PORTB0 FOR O
```

Next we configure the ADC with the ADC Control & Status Register (ADCSRA). The high bit (ADEN) enables the ADC. And the 7th bit tell the ADC to start a conversion. The lower three bits of this register select our pre-scaler/divider. The data-sheet tells us that the ADC works best at a speed between 50Khz and 200Khz. Our main clock speed is 1.2Mhz, if we divide this by eight it will give us an ADC speed of 150Khz:

```
RLOOP:
      LDI   A,0b1100_0011  ;ENABLE, START CON
      OUT   ADCSRA,A
```

We select the ADC #3 input by setting the lowest two bit of the ADMUX Register to three. Note that you must do this after the ADC has been enabled by the previous commands:

```
      LDI   A,0b0000_0011  ;SELECT ADC INP
      OUT   ADMUX,A
```

Now we wait for the conversion to be complete by watching the ADC flag (ADIF). When it is set to one, the conversion is done:

```
WAIT:  IN    A,ADCSRA       ;READ THE STATUS
       ANDI  A,0b0001_0000  ;CHECK FLAG (1<
```

Here we read the results. It is important to remember the the Low Byte must be read first and that both registers must be read for the ADC to function properly. We must read the ADCH Register, even though we do not use its results. For this version of the program we are only using the lowest eight bits of the ten-bit conversion, the value stored in the ADCL register:

```
      IN    A,ADCL         ;MUST READ ADCL B
      IN    AH,ADCH        ;REQUIRED, THOUGH
```

The PAUSE routine waits a length of time based on the lowest eight bits of our ADC conversion that we read into the "A" register.

```
      RCALL PAUSE          ;VARIABLE TIME DE
```

The SOUND routine simply toggles the output pin we have attached to our speaker on Port B pin 0:

```
      RCALL SOUND          ;MAKE A SOUND O
```

After updating our speaker output and creating a sound, we loop-back and start the process again:

```
        RJMP RLOOP
```

The SOUND routine reads in PortB then toggles the lower-most bit, PortB,0 the pin where we have our speaker connected, the result is a click to our speaker. We make the clicks fast enough and we can hear the result as a tone coming from the speaker:

```
SOUND: LDI C,1
       IN  B,PORTB
       EOR B,C
       OUT PORTB
        RET
```

The PAUSE routine takes the value of our conversion (ADCL) that was read into the "A" Register and starts subtracting one until it reaches zero then it returns. The result is a variable-length pause that is controlled by the value stored in the "A" Register, this will cause a change in the frequency of our sound output.

```
PAUSE: DEC A
        BRNE PAUSE
         RET
```

If you programmed your chip and connect the circuit 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 instrument 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-eight:

```
        LDI   A,0b1110_0011  ;[ADEN,ADSC,ADATE
        OUT   ADCSRA,A       ;START ANALOG TO
```

We can eliminate the parts of our program that wait for the ADIF flag to be set. We read the output registers ADCL & ADCH. Remember we must read both, and that ADCL must be read first:

```
RLOOP:
        IN    A,ADCL         ;MUST READ ADCL B
        IN    AH,ADCH        ;REQUIRED, THOUGH
```

Previously we generated sound by toggling the output pin with the following routine:

```
SOUND: LDI C,1
       IN  B,PORTB
       EOR B,C
       OUT PORTB,B
        RET
```

According to the data-sheet, we can toggle the outputs of the PortB Register by writing a one to its corresponding PIN number. So the entire SOUND subroutine can be replace by the single command:

```
         SBI   PINB,0         ;TOGGLE SPEAKER
```

After making these changes our simplified "free running" program now looks like this:

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

.DEF A       = R16
.DEF AH      = R17

.ORG $0000
```

```
RESET:                      ;ADJUST PRESCALER
       LDI   A,0b1110_0011  ;[ADEN,ADSC,ADATE
       OUT   ADCSRA,A       ;START ANALOG TO
       LDI   A,0b0000_0011  ;SELECT ADC #3 (P
       OUT   ADMUX,A
       SBI   PORTB,PORTB3   ;ENABLE PULL-UP R
       SBI   DDRB,0         ;SET PORTB0 FOR O
RLOOP:
       IN    A,ADCL         ;MUST READ ADCL B
       IN    AH,ADCH        ;REQUIRED, THOUGH
       RCALL PAUSE          ;VARIABLE TIME DE
       SBI   PINB,0         ;TOGGLE SPEAKER O
        RJMP RLOOP

PAUSE: DEC A
        BRNE PAUSE
         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,0b1110_0011   ;[ADEN,ADSC,ADAT
       OUT   ADCSRA,A        ;START ANALOG TO
       LDI   A,0b0010_0011   ;SET TO LEFT SHI
       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
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 B
        IN    A,ADCH          ;
```

The results of these changes is the program below:

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

.DEF A       = R16

.ORG $0000
RESET:                      ;ADJUST PRESCALER
       LDI   A,0b1110_0011 ;[ADEN,ADSC,ADATE,
       OUT   ADCSRA,A       ;START ANALOG TO
       LDI   A,0b0010_0011  ;SET TO LEFT SHIF
       OUT   ADMUX,A
       SBI   PORTB,PORTB3   ;ENABLE PULL-UP R
       SBI   DDRB,0         ;SET PORTB0 FOR O
RLOOP:
       IN    A,ADCL         ;MUST READ ADCL B
       IN    A,ADCH         ;
       RCALL PAUSE          ;VARIABLE TIME DE
       SBI   PINB,0         ;TOGGLE SPEAKER O
        RJMP RLOOP

PAUSE: DEC A
        BRNE PAUSE
         RET
```

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, the 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-sheet we find that the interrupt vector for the ADC is located at $0009, so we put a jump to our interrupt routine there:

```
.ORG $0009
       RJMP ANA_CONV
```

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

```
                SEI                ;ENABLE INTER
```

Now we are going to set the ADC Status & Control Register the same as last time except we are going to set the ADC Interrupt 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 Interrupt Enable ADIE bit is set; and the pre-scaler/divider is set to divide-by-eight:

```
              LDI   A,0b1110_1011  ;[ADEN,ADSC,ADATE
              OUT   ADCSRA,A       ;START ANALOG TO
```

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
     PUSH AH              ;SAVE CONTENTS OF
     IN   A,SREG          ;SAVE THE SYSTEM S
```

```
        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 BE
        IN    AH,ADCH         ;REQUIRED, THOUGH
        RCALL PAUSE           ;VARIABLE TIME DEL
        SBI   PINB,0          ;TOGGLE SPEAKER ON
```

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
            OUT SREG,A
            POP AH            ;RESTORE "AH" RE
            POP A             ;RESTORE "A" REG
             RETI
```

After making these changes our complete program becomes:

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

.DEF A       = R16
.DEF AH      = R17

.ORG $0000
       RJMP RESET
.ORG $0009
       RJMP ANA_CONV
RESET:                        ;ADJUST PRESCALER
       LDI   A,0b1110_1011    ;[ADEN,ADSC,ADATE
       OUT   ADCSRA,A         ;START ANALOG TO
       LDI   A,0b000_0011     ;SELECT ADC #3 FO
       OUT   ADMUX,A
       SBI   PORTB,PORTB3     ;ENABLE PULL-UP R
       SBI   DDRB,0           ;SET PORTB0 FOR O
       SEI                    ;ENABLE INTERRUPT
RLOOP: RJMP RLOOP
```

```
ANA_CONV:
        PUSH A                  ;SAVE CONTENTS OF
        PUSH AH                 ;SAVE CONTENTS OF
        IN   A,SREG             ;SAVE THE SYSTEM
        PUSH A
        IN    A,ADCL            ;MUST READ ADCL B
        IN    AH,ADCH           ;REQUIRED, THOUGH
        RCALL PAUSE             ;VARIABLE TIME DE
        SBI   PINB,0            ;TOGGLE SPEAKER O
        POP A                   ;RESTORE SYSTEM S
        OUT SREG,A
        POP AH                  ;RESTORE "AH" REG
        POP A                   ;RESTORE "A" REGI
         RETI

PAUSE: DEC A
        BRNE PAUSE
         RET
```

Subpages (1): TIMER COUNTERS & PWM

Comments

You do not have permission to add comments.