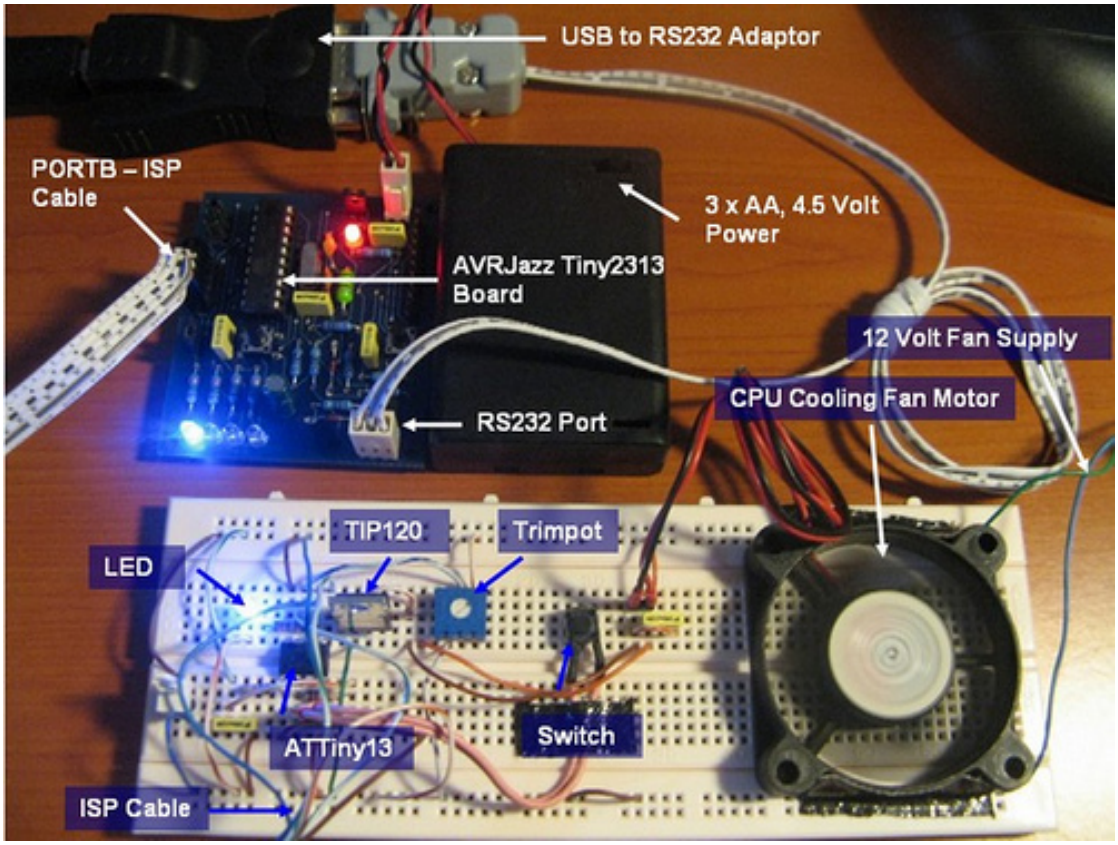
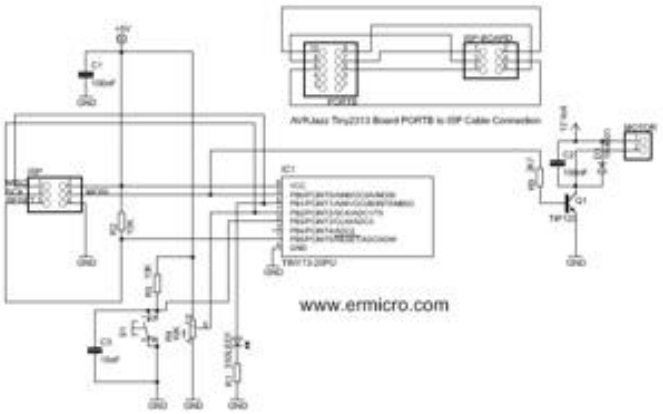
... alebo, ako na periférne moduly ADC a PWM v ATtiny13



Iste budete so mnou súhlasiť, keď poviem, že by bolo celkom zaujímavé vyskúšať, čo dokáže tento najmenší a najlacnejší 8 pinový mikrokontrolér z 8 bitovej rodiny Atmel AVR, vybavený periériami ako dvojkanálový 8 bitový PWM modul a štvorkanálový 10 bitový A/D prevodník. Mikrokontrolér disponuje 1KB programovej pamäte, 64B pamäte RAM a rovnako veľkou 64B internou pamäťou EEPROM, čo je viac menej postačujúce pre väčšinu jednoduchých aplikácií využívajúcich moduly ADC a PWM. Ak by ste ale predsa potrebovali viac pamäte, môžete namiesto ATtiny13 použiť niektorý z jeho pinovo kompatibilných bratov nachádzajúcich sa v nasledujúcej tabuľke:

K demonštrovaniu schopností mikrokontroléra som sa ho rozhodol použiť na ovládanie otáčok ventilátora, ktorý kedysi slúžil ako chladenie procesora Intel Celeron v mojom starom počítači. Pripojením motora k výstupu OC0A (pin 5) mikrokontroléru pomocou darlingtonového tranzistora TIP120, dokážeme prostredníctvom interného PWM modulu ATtiny13 jednoducho a elegantne ovládať rýchlosť otáčok ventilátora. Rýchlosť otáčania ventilátora nastavujeme 10K trimerom R4 pripojeným k pinu 7 ATtiny13, ktorý je nakonfigurovaný ako analógový vstup. Aby to bolo celé trošku zaujímavejšie pridal som do zapojenia tlačítko S1 slúžiace na vypnutie a zapnutie ventilátora. Ďalej si na schéme môžete všimnúť diódu LED1, ktorá má za úlohu indikovať beh programu a to tak, že čím sa ventilátor točí pomalšie tým rýchlejšie bliká LEDka a naopak čím sa ventilátor točí rýchlejšie tým je blikanie LEDky pomalšie.



Skôr ako pristúpime k samotnému programu zhrnieme ešte hardvér a softvér, ktorý bol použitý v tomto projekte. Takže celkovo tu máme:

- rezistory 1x 300R, 1x 2K7, 2x 10K a 1x 10K trimer
- kondenzátory 1x 10n, 2x 100n
- jednu diódu 1N4001 a jednu LEDku
- jeden darlingtonový tranzistor TIP120
- mikrokontrolér ATtiny13
- 12V jednosmerný motor - starý ventilátor od procesora
- dosku AVRJazz Tiny2313 z www.ermicro.com/
- programovací softvér AvrOspII do Mikea Henninga a Atmel AVR Studio 4

Firmvér

K napísaniu programu do ATtiny13 som sa pre tentokrát rozhodol použiť AVR assembler namiesto jazyka C, pretože som si jednoducho chcel byť istý, že sa mi celý program vojde do tej maličkej 1KB pamäte a zároveň som bol zvedavý aká bude výsledná veľkosť kódu. Nuž pozrime sa teda na samotný program:

```
1. ;***********************************************************************
2. ; Program          : t13pwm.asm
3. ; Description      : Tiny13 Fast PWM and ADC Fan Controller
4. ; Last Updated     : 15 December 2008
5. ; Author           : RWB
6. ; IDE/Compiler     : Atmel AVR Studio 4.14
7. ; Programmer       : AvrOspII v5.47 from Mike Henning
8.
```

```
;                    : AVRJazz Tiny2313 Board

;*********************************************************************

.include "tn13def.inc"



; The Tiny13 Default Frequency Clock

.equ F_CPU = 9600000



.cseg



; Start on the flash ram´s address 0

.org 0

main:   ldi R24,RAMEND        ; Initial Stack             Pointer

        out SPL,R24           ; SP = RAMEND



; Initial I/O

        ldi   R16,0b00010011   ; Set            PB0=Output, PB1=Output,            PB2=Input, PB3=Input, PB4=Output

        out   DDRB,R16        ; DDRB=0x13



; Initial ADC

        ldi   R16,0b10000110

        out   ADCSRA,R16      ; Turn On the ADC, with prescale 64
```

```
29.
        ldi   R16,0b00000000

30.
        out   ADCSRB,R16      ; Free running mode

31.
        ldi   R16,0b01100001

32.
        out   ADMUX,R16       ; Internal Reference 1.1 Volt,Left Adjust, Channel: PB2 (ADC1)

33.
        ldi   R16,0b00000100   ; Disable              Digital Input on PB2

34.
        out   DIDR0,R16

35.


36.
    ; Initial PWM

37.
        ldi   R16,0b10000011

38.
        out   TCCR0A,R16      ; Fast PWM Mode, Clear on OC0A

39.
        ldi   R16,0b00000100

40.
        out   TCCR0B,R16      ; Used fclk/256 prescale

41.


42.
    ; Initial the Button Flag and PORTB

43.
        ldi   R17,0           ; Initial Button Flag

44.
        out   PORTB,R17

45.


46.
    lb_00:  sbic  PINB,PB3        ;             if (PB3 == 0)

47.
        rjmp  lb_20           ; else goto lb_20

48.
        ldi   R19,5           ; Use delay for simple debounce
```

```
49.
        rcall delay_func
50.
        sbic  PINB,PB3        ; if (PB3 == 0), read again
51.
        rjmp  lb_20           ; else goto lb_20
52.
        cpi   R17,1           ; Process if button pressed
53.
        brne  lb_10           ; if (R17 != 1) goto lb_10
54.
        ldi   R17,0           ; else R17=0
55.
56.
        ldi   R16,0           ; Disable PWM Clock
57.
        out   TCCR0A,R16      ; TCCR0A = 0
58.
        out   TCCR0B,R16      ; TCCR0B = 0
59.
        cbi   PORTB,PB0       ; Turn off Motor
60.
61.
        cbi   PORTB,PB1       ; Turn off LED
62.
        rjmp  lb_20
63.
64.
    lb_10:  ldi   R17,1                   ; R17=1
65.
        ldi   R16,0b10000011
66.
        out   TCCR0A,R16      ; Fast PWM Mode, Clear on OCR0A
67.
        ldi   R16,0b00000100
68.
        out   TCCR0B,R16      ; Used fclk/256 prescale
69.
```

```
        sbi   PORTB,PB1       ; Turn on LED

70.

71.
    lb_20:  cpi   R17,1                   ; if (R17 != 1)

72.
        brne  lb_40           ; goto lb_50

73.
                                  ;

74.
        sbi   ADCSRA,ADSC      ; Start ADC conversion

75.
    lb_30:  sbic  ADCSRA,ADSC      ; while (ADCSRA &amp; (1<<ADSC))

76.
        rjmp  lb_30

77.
        in    R16,ADCH         ; Read the result Ignore the              last 2 bits in ADCL

78.
        out   OCR0A,R16        ; OCR0A = R16

79.

80.
        cbi   PORTB,PB1        ; Turn off LED

81.
        mov   R19,R16

82.
        rcall delay_func       ; Call Delay Function Parameter R19

83.
        sbi   PORTB,PB1        ; Turn on LED

84.

85.
    lb_40:  mov   R19,R16

86.
        rcall delay_func       ; Call Delay Function Parameter R19

87.
        rjmp  lb_00

88.

89.
    ; Simple Delay Function
```

```
90.
    delay_func:

91.
    delay0: ldi   R20,25          ; R20 = 25

92.
    delay1: ldi   R21,255         ; R21 = 255

93.
    delay2: dec   R21             ; Decrease R21

94.
            brne  delay2          ; if (R20 != 0) goto delay2 label

95.
            dec   R20             ; Decrease R20

96.
            brne  delay1          ; if (R20 != 0) goto delay1 label

97.
            dec   R19             ; Decrease R19

98.
            brne  delay0          ; if (R19 != 0) goto delay0 label

99.
            ret                   ; Return to the caller

100.
    .exit
```

Prvá skupina údajov, resp. prvý odsek hlavného programu patrí inicializácii zásobníka, ktorý bol umiestnený na koniec 64B - vej pamäte RAM. Za ním nasleduje inicializácia vstupno/výstupného portu B, pričom počiatočné nastavenie logických úrovní a smeru jednotlivých pinov sa nachádza v nasledujúcej tabuľke:

DDRB	Logical	Direction	PIN	Function Name	Function
PB0	1	Out	5	PB0/OC0A	PWM Output (Digital)
PB1	1	Out	6	PB1	LED Output (Digital)
PB2	0	In	7	PB2/ADC1	ADC Input (Analog)
PB3	0	In	2	PB3	Switch Input (Digital)

Inicializácia A/D prevodníka

V programe ďalej nasleduje inicializácia periférneho modulu A/D prevodníka (pre komplexné porozumenie modulu odporúčam preštudovať si datasheet mikrokontroléra ATiny13). Na to aby sme mohli využívať služby tohto periférneho modulu, musíme najskôr vykonať nasledujúce kroky:

A. Nastaviť predeličku a aktivovať samotný periférny modul

Obvod predeličky realizuje delenie taktovacej frekvencie (z interného alebo aj externého zdroja taktovacej frekvencie) v pomere, ktorý nastavuje programátor prostredníctvom bitov ADPS2, ADPS1 a ADPS0 v registri ADCSRA. Taktovací signál za predeličkou je následne použitý pri samotnom A/D prevode, založenom na metóde postupnej aproximácie. Na získanie maximálneho možného rozlíšenia je možné vybrať si frekvenciu taktovania medzi 50 až 200 kHz, pričom v tomto projekte som zvolil deliaci pomer 1:64, čo v konečnom dôsledku znamená taktovaciu frekvenciu pre prevodník okolo 150 kHz.

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Pretože nevyužívame funkcie automatického spúšťania prevodu a ani prerušenia po jeho skončení, ponecháme bity ADATA, ADIE a ADIF nastavené na nulu, vyberieme deliaci pomer 1:64 a bitom ADEN povolíme funkciu modulu (zapneme ho ...), tak ako to ukazuje nasledujúci kúsok kódu:

1.
```
ldi   R16,0b10000110
```

2.
```
out   ADCSRA,R16       ; Turn On the ADC, with              prescale 64
```

B. Vybrať zdroj (udalosť) spustenia A/D prevodu

V projekte je na spúšťanie A/D prevodu využitý tzv. free running mode, čo znamená, že periférny modul ADC vykoná prevod vždy keď dostane pokyn na jeho uskutočnenie. Výber tohto módu dosiahneme nastavením bitov ADTS2, ADT21 and ADTS0 v registri ADCSRB na log. nulu.

Bit	7	6	5	4	3	2	1	0	
	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter Compare Match A
1	0	0	Timer/Counter Overflow
1	0	1	Timer/Counter Compare Match B
1	1	0	Pin Change Interrupt Request

1.
```
ldi   R16,0b00000000
```

2.
```
out   ADCSRB,R16       ; Free running mode
```

C. Zvoliť príslušný kanál

Periférny modul ADC v Atiny13 obsahuje jeden samostatný A/D prevodník, ktorý môže vykonávať prevod analógového signálu na digitálny z jedného zo štyroch možných analógových vstupov v jednom čase. Pre nás to znamená, že musíme perifernemu modulu povedať, ktorý zo vstupov budeme používať ako aktuálny zdroj analógového signálu. Toto vykonáme nastavením dvojice bitov MUX0 a MUX1 v registri ADMUX.

Bit	7	6	5	4	3	2	1	0	
	–	REFS0	ADLAR	–	–	–	MUX1	MUX0	ADMUX
Read/Write	R	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

MUX1..0	Single Ended Input
00	ADC0 (PB5)
01	ADC1 (PB2)
10	ADC2 (PB4)
11	ADC3 (PB3)

Z vyššie uvedenej tabuľky vidíme, že pre náš projekt (trimer R4 je pripojený na vstup PB2/ADC1) musíme nastaviť hodnoty bitov MUX0 = 1 a MUX1 = 0. Ďalej nastavením bitu REFS0 na log. jednotku zvolíme 1,1V internú napäťovú referenciu.

REFS0	Voltage Reference Selection
0	V_{CC} used as analog reference.
1	Internal Voltage Reference.

Po skončení A/D prevodu sa jeho výsledok bude nachádzať v registrovom páre ADCH a ADCL. Pretože v rámci tohto projektu postačoval 8 - bitový prevod, jednoducho sme nastavením bitu ADLAR v registri ADMUX na log. jednotku zarovnali výsledok prevodu na ľavú stranu a prečítali len register ADCH, pričom spodné dva bity prevodu nachádzajúce sa v registri ADCL sme ignorovali.

1.
```
ldi   R16,0b01100001
```

2.
```
out   ADMUX,R16     ; Reference 1.1 Volt, Left Adjust, Channel: PB2
```

D. Na požadovanom pine vypnúť funkciu digitálneho vstupu/výstupu

Poslednou úlohou v inicializačnom procese je vypnúť funkciu digitálneho vstupu/výstupu na príslušnom požadovanom pine, ktorý má pracovať ako analógový vstup. Toto dosiahneme nastavením príslušného bitu v registri DIDR0 na log. jednotku, v prípade tohto projektu je to bit ADC1D.

Bit	7	6	5	4	3	2	1	0	
	-	-	ADC3D	ADC2D	ADC1D	ADC0D	AREFD	AIN0D	DIDR0
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

1.
```
ldi   R16,0b00000100   ; Disable Digital Input on PB2
```

2.
```
out   DIDR0,R16
```

Generovanie PWM signálu

ATtiny13 obsahuje dve nezávislé porovnávacie (komparačné) jednotky s podporou generovania PWM signálu, ktorých výstupy OC0A a OC0B sa nachádzajú na pinoch PB0 a PB1, pričom prvý z menovaných výstupov je využívaný aj v tomto projekte. K aktivácii PWM periférie v mikrokontroléri ATtiny13 potrebujeme urobiť nasledujúce kroky:

A. Zvoliť režim generovania PWM signálu

Mikrokontrolér ATtiny13 podporuje dva PWM režimy. Prvým z nich je tzv. *fast PWM mode* (použitý v tomto projekte) umožňujúci generovať na výstupe signál s pomerne vysokými frekvenciami. Druhým je tzv. *phase correct PWM mode*, ktorý na úkor nižších

frekvencií generovaného výstupného signálu poskytuje väčšie rozlíšenie v porovnaní s režimom fast PWM.

Bit	7	6	5	4	3	2	1	0	
	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal PoE Operation, OC0A Disconnected. WGM02 = 1: Toggle OC0A on Compare Match.
1	0	Clear OC0A on Compare Match, set OC0A at TOP
1	1	Set OC0A on Compare Match, clear OC0A at TOP

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on [1] [2]
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	TOP	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	TOP	TOP

Nastavením bitu COM0A1 na log.1 a bitu COM0A0 na log.0 vyberieme možnosť "*clear OC0A on compare match, set OC0A on top*", čo znamená, že keď sa bude hodnota v registri TCNT0 (časovač/čítač 0) rovnať hodnote v registri OCR0A, na výstupe OC0A (PB0) sa bude nachádzať logická nula a keď hodnota v registri TCNT0 dosiahne svoje maximum (255), výstup OC0A bude logická jednotka. Následne si nastavením bitov WGM2 = log.0 v registri TCCR0B a WGM1 = WGM2 = log.1 v registri TCCR0A zvolíme režim *fast PWM mode*.

1.
```
ldi   R16,0b10000011
```

2.
```
out   TCCR0A,R16       ; Fast PWM Mode, Clear on OC0A
```

B. Vybrať frekvenciu hodinového signálu pre časovač/čítač 0

Srdce periférneho modulu PWM je v 8-bitovom časovači/čítači. Keď spustíme počítanie, obvod generovania výstupného PWM signálu bude kontinuálne porovnávať aktuálnu hodnotu v registri TCNT0 s hodnotou v registri OCR0A a na základe výsledku porovnania generovať výstupný PWM signál. Na to aby sme aktivovali čítač, k nemu potrebujeme priviesť hodinový signál, ktorého frekvenciu si môžeme vybrať prostredníctvom internej predeličky nachádzajúcej sa v samotnom module časovača/čítača.

Bit	7	6	5	4	3	2	1	0	
	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)

V tomto projekte je pre časovač/čítač použitý hodinový signál, ktorý sme získali z hodinového signálu interného oscilátora jeho delením v pomere 1:256, čo znamená, že frekvenciu generovaného PWM signálu v *fast PWM mode* môžeme vypočítať podľa vzťahu:

$$f_{OCnxPWM} = \frac{f_{clk_I/O}}{N \cdot 256}$$

, kde N predstavuje veľkosť deliaceho pomeru pre hodinový signál časovača/čítača. Takže frekvencia PWM signálu na pine OCA0 bude mať pri internom oscilátore s frekvenciou 9,6 MHz hodnotu **146,48 Hz**. Samotný deliaci pomer je závislý od jednosmerného motora, ktorý používate a tým pádom by bolo vhodné s týmto pomerom trošku experimentovať. Skúste začať s

veľkým deliacim pomerom a postupne ho zmenšovať, pričom pozorujte, ako na tieto zmeny reaguje váš motor.

```
1.
  ldi   R16,0b00000100

2.
  out   TCCR0B,R16       ; Used fclk/256 prescale
```

Jadro kódu

Samotný program je v podstate vďaka početným komentárom veľmi dobre vysvetlený, no aj napriek tomu som sa rozhodol jadro programu napísať prostredníctvom pseudo kódu v štýle jazyka C, čo Vám dúfajme lepšie pomôže pochopiť ako vlastne program funguje.

```
1.
  Initial_PORTB();

2.
  Initial_ADC();

3.
  Initial_PWM();

4.

5.
  R17_SwicthFlag = 0;

6.

7.
  for(;;) {

8.
    if (switch_PB3_is_pressed()) {

9.
      if (R17_SwicthFlag == 0) {

10.
        R17_SwicthFlag = 1;

11.
        Enable_PWM();

12.
        Turn_On_LED();
```

```
13.
      } else {
14.
        R17_SwicthFlag = 0;
15.
        Disable_PWM();
16.
        Turn_Off_LED();
17.
      }
18.
    }
19.
20.
    if (R17_SwicthFlag == 1) {
21.
      Enable_ADC();
22.
      Wait_ADC_Conversion();
23.
      PWM_OC0A = ADC_Result_in_ADCH;
24.
25.
      Turn_Off_LED();
26.
      Delay(ADC_Result_in_ADCH);
27.
      Turn_On_LED()
28.
    }
29.
30.
    Delay(ADC_Result_in_ADCH);
31.
  }
```

Z pseudo kódu ste mohli vidieť, že údaje vo vnútri nekonečnej slučky tvorenej príkazom for(;;) sú podobné údajom, ktoré sa nachádzajú v programe napísanom v assembleri medzi nasledujúcimi dvomi príkazmi:

1. lb_00: sbic PINB,PB3
2.
3. ...statement
4. ...statement
5. ...statement
6.
7. rjmp lb_00

Vo vnútri nekonečnej slučky sa vykonávajú činnosti ako sledovanie stavu tlačidla, aktivácia a deaktivácia PWM signálu, A/D prevod, zapínanie a vypínanie LEDky. Nastavením bitu ADSC v registri ADCSRA na log.1 dáme pokyn modulu A/D prevodníka na snímanie kanála ADC1 (pin PB2), vykonanie A/D prevodu a následne si už len počkáme kým sa samotný A/D prevod ukončí:

1. sbi ADCSRA,ADSC ; Start ADC conversion
2. lb_30: sbic ADCSRA,ADSC ; while (ADCSRA & (1<<ADSC))
3. rjmp lb_30
4. in R16,ADCH ; Read the result Ignore the last 2 bits in ADCL
5. out OCR0A,R16 ; OCR0A = R16

Ako som už spomínal predtým, LED diódu využívam na indikáciu toho či môj "program vôbec žije", ale aby to bolo zaujímavejšie, ako parameter oneskorovacej funkcie som použil výsledok A/D prevodu. Takto môžem pri "vrtení" trimrom vidieť meniacu sa hodnotu oneskorenia.

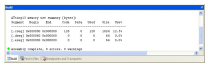
1.
```
mov   R19,R16
```

2.
```
rcall delay_func       ; Call Delay Function Parameter R19
```

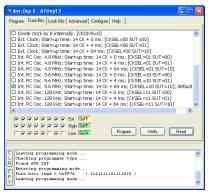
Čím väčšia bude hodnota v registri ADCH (výsledok A/D prevodu), tým dlhšie bude trvanie log. 1 v PWM signále na výstupe OC0A, čo znamená, že tým rýchlejšie sa bude točiť motor a tým dlhšie bude svietiť LED dióda. Naopak, čím bude hodnota v registri ADCH menšia, tým kratšie bude trvanie log.1 vo výstupnom PWM signále a tým pomalšie sa bude točiť motor a zároveň kratšie bude svietiť LED dióda.

Nahrávanie a spúšťanie programu

Po kompilácii a ladení (vždy by ste mali dodržiavať tento postup) by ste mali dostať výsledok, ktorý vidíte na nasledujúcom obrázku:



Skompilovaný program zaberal v pamäti okolo 128B z celkovej veľkosti 1024B, čo podľa mňa vyzerá dobre a je vidieť, že v pamäti je ešte dostatočné množstvo priestoru pre budúce rozširovanie aplikácie. Zároveň je z tohto projektu vidieť, že 1KB programová pamät v ATtiny13 je postačujúca pre priemernú aplikáciu využívajúcu periférne moduly ADC a PWM. Skôr ako pomocou programu AVR - OSP II nahráme samotný kód do mikrokontroléru je potrebné ešte skontrolovať a správne nastaviť poistky (fuse bits), pričom poistky by mali byť nastavené tak, ako je to vidieť z obrázka nižšie:



Aby ste si boli naozaj istý skontrolujte si ešte raz nastavenie poistiek, konkrétne:**Int. RC Osc. 9.6Mhz; Start-up time: 14 CK + 64ms** a po skontrolovaní nahrajte kód do ATtiny13.

A na záver? Projekt s ATtiny13 v akcii:

Zverejnené so súhlasom autora.
Homepage projektu: Controlling DC motor with AVR ATtiny13 PWM and ADC Project

Preklad: Kiwwicek