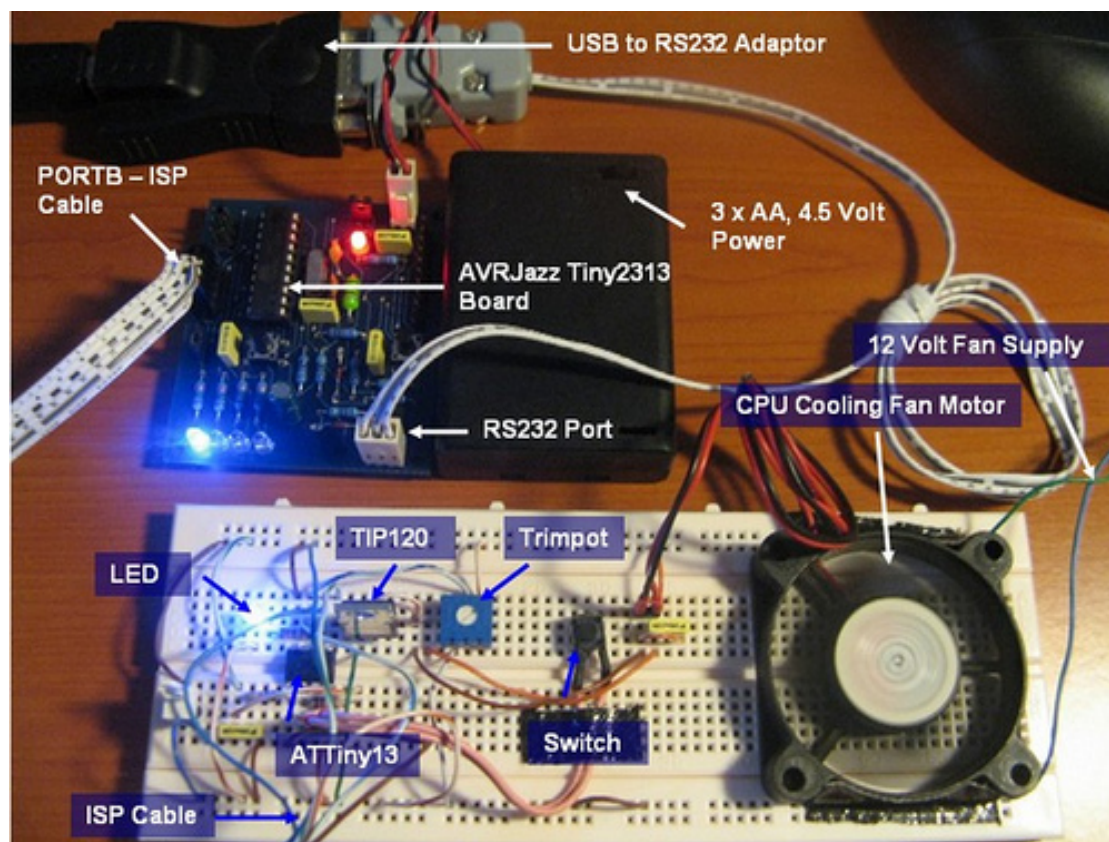


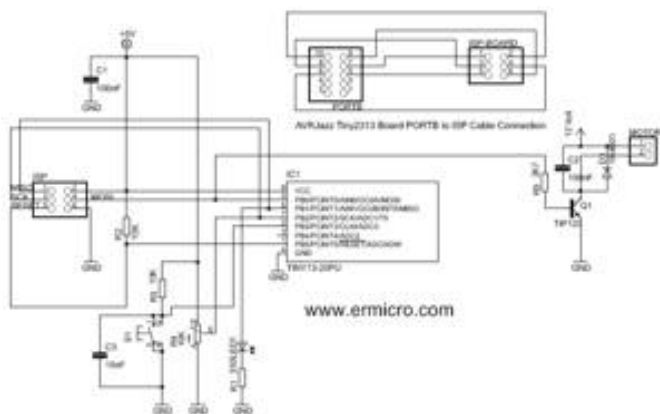
... alebo, ako na periférne moduly ADC a PWM v ATtiny13



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 perifériami 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:

ATtiny13	ATtiny13L	ATtiny13LF	ATtiny13L-1.6	ATtiny13L-1.6V	ATtiny13L-1.6V-1.6	ATtiny13L-1.6V-1.6V	ATtiny13L-1.6V-1.6V-1.6	ATtiny13L-1.6V-1.6V-1.6V	ATtiny13L-1.6V-1.6V-1.6V-1.6
----------	-----------	------------	---------------	----------------	--------------------	---------------------	-------------------------	--------------------------	------------------------------

K demonštrovaní 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 AvrOsp1l 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 maličkkej 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

9.
;*****

10.
.include "tn13def.inc"

11.

12.
; The Tiny13 Default Frequency Clock

13.
.equ F_CPU = 9600000

14.

15.
.cseg

16.

17.
; Start on the flash ram's address 0

18.
.org 0

19.
main: ldi R24,RAMEND ; Initial Stack Pointer

20.
      out SPL,R24 ; SP = RAMEND

21.

22.
; Initial I/O

23.
      ldi R16,0b00010011 ; Set PB0=Output, PB1=Output, PB2=Input, PB3=Input, PB4=Output

24.
      out DDRB,R16 ; DDRB=0x13

25.

26.
; Initial ADC

27.
      ldi R16,0b10000110

28.
      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 & (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:

GPIO	Logická	Smernosť	Pin	Function Name	Function
GPIO1	In	Out	5	PERIODIC	PWM Output (Digital)
GPIO1	Out	In	6	PER	LED Output (Digital)
GPIO0	In	In	7	PERIODIC1	ADC Input (Analog)
GPIO0	In	In	2	PER	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.

ADIFSC	ADIFS1	ADIFS2	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. `Idi 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.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ADSC	ADIFSC	ADIFS1	ADIFS2	ADIF	ADSC	ADIFSC	ADIFS1	ADIFS2	ADIF	ADSC	ADIFSC	ADIFS1	ADIFS2	ADIF	ADSC
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ADIFSC	ADIFS1	ADIFS2	Trigger Source
0	0	0	Start Conversion
0	0	1	Analog Comparator
0	1	0	External Interrupt/Event 0
0	1	1	External Interrupt/Event 1
1	0	0	Timer/Counter Compare Match 0
1	0	1	Timer/Counter Compare Match 1
1	1	0	Timer/Counter Compare Match 2
1	1	1	Timer/Counter Compare Match 3

1. `Idi 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 periférnemu modulu povedať, ktorý zo vstupov budeme používať ako aktuálny zdroj analógového signálu. Toto vykonáme nastavením dvojice bitov MUX0 a MUX1 v registri ADMUX.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ADSC	ADIFSC	ADIFS1	ADIFS2	ADIF	ADSC	ADIFSC	ADIFS1	ADIFS2	ADIF	ADSC	ADIFSC	ADIFS1	ADIFS2	ADIF	ADSC
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

9 / 14

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



COM0	COM0	Description
0	0	Normal port operation, I/O as programmed
0	1	Reserved - E-Serial Port Operation, I/O as programmed
1	0	Clear OC0A on Compare Match, set OC0A on TOP
1	1	Set OC0A on Compare Match, clear OC0A on TOP

Mode	WGM0	WGM1	WGM2	WGM3	Time/Center Mode of Operation	TCF	Update of OCRA at	TCF Plug In at
0	0	0	0	0	Normal	Diff	Immediate	SAK
1	0	0	0	1	PWM Phase Correct	Diff	TCF	BOTTOM
2	0	0	0	1	CTC	OCRA	Immediate	SAK
3	0	0	1	0	Fast PWM	Diff	TCF	SAK
4	1	0	0	0	Reserved	—	—	—
5	1	0	1	0	PWM Phase Correct	OCRA	TCF	BOTTOM
6	1	1	0	0	Reserved	—	—	—
7	1	1	1	0	FastPWM	OCRA	TCF	TCF

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_SwifthFlag = 0;
- 6.
7.
for(;;) {
8.
 if (switch_PB3_is_pressed()) {
9.
 if (R17_SwifthFlag == 0) {
10.
 R17_SwifthFlag = 1;
11.
 Enable_PWM();
12.
 Turn_On_LED();

```
13.     } else {  
14.         R17_SwifthFlag = 0;  
15.         Disable_PWM();  
16.         Turn_Off_LED();  
17.     }  
18. }  
19.  
20. if (R17_SwifthFlag == 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`

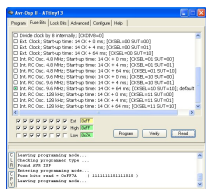
Čím väčšia bude hodnota v registri ADCH (výsledok A/D prevodu), tým dlhšie bude trvanie log. 1 v PWM signále na výstupe OC0A, čo znamená, že tým rýchlejšie sa bude točiť motor a tým dlhšie bude svietiť LED dióda. Naopak, čím bude hodnota v registri ADCH menšia, tým kratšie bude trvanie log.1 vo výstupnom 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 pamäť v ATtiny13 je postačujúca pre priemernú aplikáciu využívajúcu periférne moduly ADC a PWM. Skôr ako pomocou programu AVR - ISP 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](#)