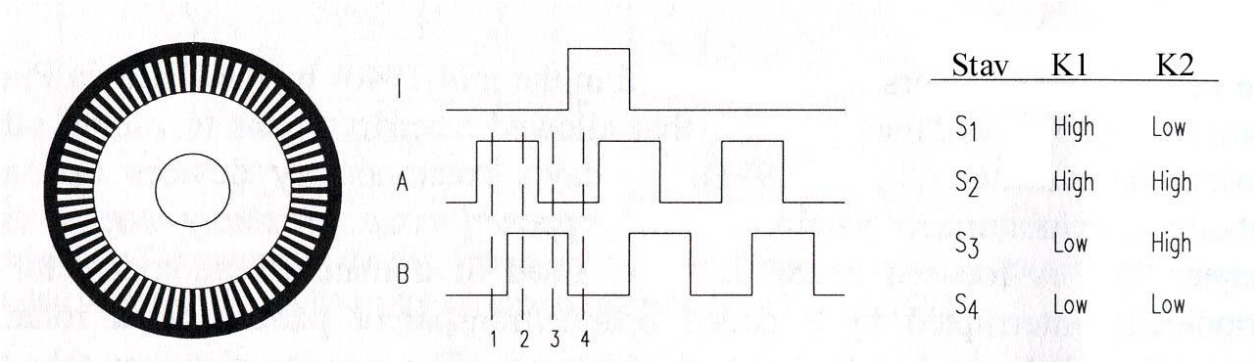
Úvod

Pro kvalitní řízení robota potřebujeme vědět, jakou vzdálenost robot ujel a jak rychle se pohybuje. Nejpoužívanější metodou pro určení polohy robota je odometrie, která používá informace o otočení kol pro výpočet robotovy pozice (viz. průvodce Odometrie http://robotika.cz/guide/odometry/cs). Potřebujeme tedy snímač, který nám tuto informaci zajistí. Nejčastěji se používají optické inkrementální snímače, buď jako průchozí nebo reflexní optická závora. Bližší popis v průvodci kapitola Enkodéry (http://robotika.cz/guide/encoders/cs).

Popis funkce

Výstupem kvadraturního enkodéru jsou dva fázově posunuté signály A, B viz. obrázek. Díky tomu můžeme kromě počtu „tiků“ tj. pootočení určit i směr pohybu.



Zpracování těchto signálů je možno realizovat hardwarově pomocí speciálních obvodů nebo softwarově.

Hardwarové zpracování

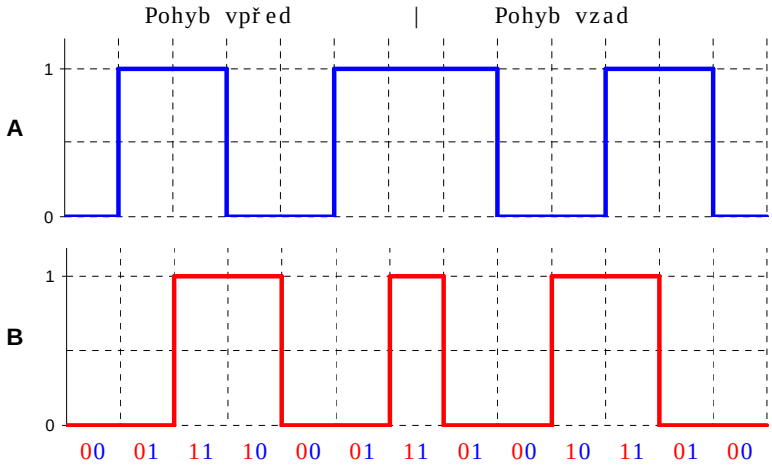
se používá hlavně pro velmi rychlé signály (>10 kHz). Výstupem specializovaných obvodů je jeden výstup generující impuls při „každé“ změně vstupních signálů z enkodéru a druhý výstup určuje směr pohybu (tím určíme zda impulsy přičítat či odčítat). Druhý typ poskytuje dva výstupy, kde jeden poskytuje impulsy při kladném směru pohybu a druhý při záporném. Takto předzpracovaný signál se obvykle dále přivádí na vstup rychlého čítače, který realizuje počítání impulsů (obvykle 8 bitů). Tento čítač poskytuje informaci o změně polohy nadřízenému systému, který musí dostatečně často vyčítat tuto informaci, aby nedošlo k přetečení hardwarového čítače.
Výhodou tohoto řešení je vysoká rychlost a minimální zatížení nadřazeného systému. Nevýhodou je vyšší obvodová náročnost řešení a tudíž i cena.
Příkladem obvodů pro dekódování signálu z kvadraturních enkodérů jsou obvody LS708x, které kromě zpracování signálu zajišťují i filtraci, která zamezuje zpracování vstupních pulsů o šířce menší, než je nastavená hodnota. Kvadraturní dekodér LS7084 je určený k dekódování kvadraturních signálů A a B na pulsy a směr, LS7083 poskytuje na jednom výstupu impulsy, které se přičítají, a na druhém impulsy, které se odečítají. Komplexnější řešení poskytují např. obvody HCTL-2022/2032, které kromě dekódování signálu obsahují i čítač. Jinou možností je realizace dekódování a následného čítaní v programovatelných obvodech CPDL, …

Softwarové zpracování

Pro pomalejší signály lze provádět celé vyhodnocení softwarově v jednočipovém počítači. Toto řešení je z hlediska obvodu jednodušší, levnější, ale přináší zvýšené výpočetní zatížení mikroprocesoru. Vzhledem výkonu současných mikrokontrolerů a potřebnému výkonu pro řízení malých robotů to však není problém.

Způsoby dekódování

Při pohybu generuje výstup enkodéru během jedné periody 4 různé stavy, viz. obrázek.



Pokud provádíme zpracování pouze při jedné změně signálu A (např. při vzestupné hraně) a signál B použijeme pouze pro určení směru pohybu, pak je rozlišení snímače přímo rovno počtu štěrbin, či odrazových plošek.
Hlavně u amatérsky vyráběných enkodéru ovšem bývá základní rozlišení příliš nízké, proto je možné využít dělení celé periody na části. Při využití obou hran kanálu A získáme dvojnásobné rozlišení a při využití obou hran kanálu A i B dokonce čtyřnásobné rozlišení.

Způsoby zpracování

Asynchronní

Při asynchronním zpracování je jeden nebo oba výstupní signály enkodéru připojeny na vstup mikrokontroleru, který je schopen generovat přerušení. Přerušení je vyvoláno při vzestupné, sestupné nebo obou hranách signálů.
Pro dekódování s jednoduchým nebo dvojnásobným rozlišením stačí jeden přerušovací vstup, pro čtyřnásobné rozlišení jsou nutné dva přerušovací vstupy.
Výhodou asynchronního zpracování je poměrně nízké zatížení procesoru, které však ze zvětšující se rychlostí roste. Poměrně velkou nevýhodou je, že nelze jednoduše realizovat filtraci proti zákmitům, které mohou nastat, pokud se enkodér zastaví na rozhraní sousedních stavů. Tedy toto zpracovaní dobře funguje při pohybu, ale při zastavení může generovat velké množství falešných tiků bez jednoduché možnosti ošetření.

Synchronní

Při tomto způsobu zpracování se vyhodnocení provádí v daných časových okamžicích. To je nejčastěji realizováno periodickým vyvoláváním přerušení od časovače. Frekvence vzorkování musí být alespoň tak vysoká jako frekvence změn stavů enkodéru, tj. čtyřnásobek frekvence výstupního signálu enkodéru (malou fintou při dekódování lze tento požadavek zmírnit téměř na polovinu).
Nevýhodou synchronního zpracování je vyšší, ale stálé zatížení procesoru. Výhodou je automatické filtrování impulsů, které mají kratší délku než je perioda vzorkování. Tím je potlačen problém zákmitů výstupů enkodéru při zastavení na hraně.

Příklady realizace softwarového zpracování

Ukázky dékodování jsou psány v jazyku C a používají následující definice:

```
#define     IRC_PORT                … port, na ktery jsou pripojeny vystupy enkoderu
#define     PIN_A                   … cislo pinu kanalu A enkoderu
#define     PIN_B                   … cislo pinu kanalu B enkoderu
#define     MASK_PIN_A  (1<<PIN_A)
#define     MASK_PIN_B  (1<<PIN_B)
#define     MASK_PIN_AB (MASK_PIN_A | MASK_PIN_B)

volatile int Position;              … promenna, ve ktere je poloha enkoderu

#define     GetPosition(x)    {sei(); x = Position; cli();}
                                    … cteni pozice musi byt pri zakazanem preruseni
```

Ukázky asynchronního zpracování jsou psány co nejjednodušším způsobem, lze je samozřejmě optimalizovat.

Asynchronní dékodování se základním rozlišením

Nastaveno přerušení na vstupu kanálu A při vzestupné hraně

```
interrupt EXT_PIN_A;
{
  if ((IRC_PORT & MASK_PIN_B) == 0)// Je-li při vzestupné hraně A,
    Position++;                    //   B = 0, pricti 1 tik
  else
    Position--;                    //   B = 1, odecti 1 tik
}
```

Asynchronní dékodování s dvojnásobným rozlišením

Nastaveno přerušení na vstupu kanálu A při obou hranách

```
interrupt EXT_PIN_A;
{
  unsigned char IRC;
  IRC = IRC_PORT;

  if ((IRC & MASK_PIN_A) == 0)           // sestupna hrana A kanalu
  {
    if ((IRC & MASK_PIN_B) == 0)
      Position--;
    else
      Position++;
  }
  else                                   // vzestupna hrana na A
  {
    if ((IRC & MASK_PIN_B) == 0)
      Position++;
    else
      Position--;
  }
}
```

Asynchronní dékodování se čtyřnásobným rozlišením

Nastaveno přerušení na vstupech kanálů A i B při obou hranách

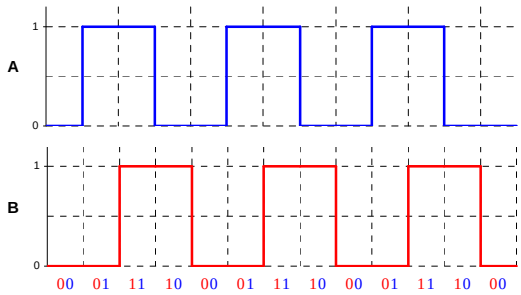
```
interrupt EXT_PIN_A;
{
  unsigned char IRC;
  IRC = IRC_PORT;

  if ((IRC & MASK_PIN_A) == 0)           // sestupna hrana A kanalu
  {
    if ((IRC & MASK_PIN_B) == 0)
      Position--;
    else
      Position++;
  }
  else                              // vzestupna hrana na A
  {
    if ((IRC & MASK_PIN_B) == 0)
      Position++;
    else
      Position--;
  }
}

interrupt EXT_PIN_B;
{
  unsigned char IRC;
  IRC = IRC_PORT;

  if ((IRC & MASK_PIN_B) == 0)           // sestupna hrana B kanalu
  {
    if ((IRC & MASK_PIN_A) == 0)
      Position++;
    else
      Position--;
  }
  else                              // vzestupna hrana na B
  {
    if ((IRC & MASK_PIN_A) == 0)
      Position--;
    else
      Position++;
  }
}
```

Synchronní dékodování se čtyřnásobným rozlišením

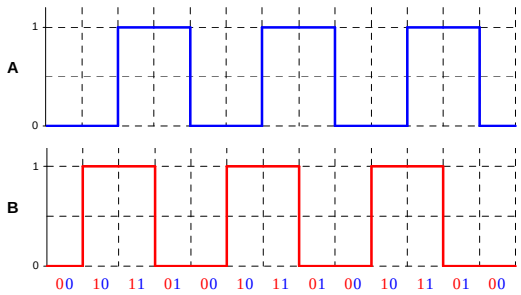
U synchronního zpracování je nutné kromě aktuálního stavu hodnot kanálů A a B enkodéru znát i předchozí stav signálů. Dekódování lze realizovat různými způsoby: vyhodnocením sekvenční logické funkce, stavovým automatem nebo dekódováním grayova kódu. Příklad ukáže nepříliš často používanou metodu, která vychází z dekódování grayova kódu. Tento postup lze velmi dobře optimalizovat a dosáhnout tak, velmi nízkého zatížení procesoru, resp. vysoké frekvence zpracování.

Princip dekódování

Pohyb vpřed



Pohyb vzad



Pokud budeme na kombinace hodnot kanálů A a B pohlížet jako na binární čísla, tak při pohybu vpřed získáme posloupnost 0, 1, 3, 2, 0, ... a při pohybu zpět 0, 2, 3, 1, 0, ... Tato posloupnost je vzestupně, resp. sestupně periodicky uspořádána, až na přehozené hodnoty 2 a 3. První fází zpracování je tedy záměna hodnot 2 → 3 a 3 → 2. Tím získáme posloupnosti 0, 1, 2, 3, 0, ..., resp. 0, 3, 2, 1, 0,... Nyní pokud tato čísla bereme jako 2-bitová, tak při pohybu vpřed je jejich rozdíl vždy 1 (1-0=1, 2-1=1, 3-2=1, 0-3=0 s přetečením) a podobně při pohybu vzad -1. To vše za předpokladu, že jsou výstupy enkodéru připojeny na piny 0 a 1 vstupního portu mikroprocesoru. Princip lze rozšířit na libovolné piny, s tím že rozdíl sousedních stavů není ±1 ale ±x. Jedinou podmínkou je, že kanál A je připojen na nižší pin než kanál B.

Příklad zjednodušené implementace v jazyce C, nastaveno přerušení od časovače. Není ošetřen stav, kdy dojde ke ztrátě kroku, tj. dojde k posunu o dva stavy a nelze určit směr pohybu.

```
unsigned char LastIRC;

interrupt CTC;
{
  unsigned char IRC, temp;

  IRC = IRC_PORT & MASK_PIN_AB;     // cteni stavu enkoderu
  if ((IRC & MASK_PIN_B) != 0)      // dekodovani grayova kodu, tj.
    IRC ^= MASK_PIN_A;              //   zamena druheho a tretiho stavu

  // rozdil soucasneho a predchoziho stavu, maskovani
  temp = (IRC – LastIRC) & MASK_PIN_AB;

  if (temp == MASK_PIN_A)           // test zda je rozdil stavu kladny (+1)
    Position++;
  else if (temp == MASK_PIN_AB)     // nebo zaporny (-1)
    Position--;

  LastIRC = IRC;                    // uschovani soucasneho stavu
}
```

Kompletní optimalizovaná varianta implementace pro mikroprocesory AVR a překladač AVR-GCC, obsluha přerušení psána v asembleru. Důvodem je neefektivní implementace obsluh přerušení v C jazyce, kde jsou ukládány „všechny“ i nepoužité registry. To představuje téměř zdvojnásobení doby zpracování přerušení. Optimalizovaná varianta obsluhy přerušení trvá maximálně 55 taktů, což při taktovací frekvenci 16MHz umožňuje teoreticky vzorkovací frekvenci více než 250 kHz. Tato hodnota je více než dostatečná pro běžné enkodéry. Je ošetřen i stav ztráty stavu, který může nastat, je-li frekvence signálu z enkodéru vyšší než frekvence vzorkování.

IntIRC.h

```
#ifndef _IntIRC_H_
#define _IntIRC_H_

#define     IRC_PORT    PIND
#define     PIN_A       2
#define     PIN_B       5
#define     MASK_PIN_A  (1<<PIN_A)
#define     MASK_PIN_B  (1<<PIN_B)
#define     MASK_PIN_AB (MASK_PIN_A | MASK_PIN_B)

#endif
```

IRC.h

```
#ifndef _IRC_H_
#define _IRC_H_

#include "IntIRC.h"

extern volatile int    EncoderPosition;  // pozice v ticich enkoderu

extern void InitIRC(void);

#endif
```

IRC.c

```
#include "IRC.h"

volatile int  EncoderPosition;     // pozice v ticich enkoderu
volatile char PreviousDir;         // pamet smeru otaceni, pro pripad ztraty hrany -
                                   // IRC snimace
volatile unsigned char LastIRC;    // predchozi obraz vstupu z enkoderu

void InitIRC(void)
{
  // CTC2, predelicka 8 – vstupni frekvence od citace 2 MHz
  // Citac s autoclear on compare match OCR
  TCCR2 = (1<<WGM21) | (0<<CS22) | (1<<CS21) | (0<<CS20);
  OCR2  = 39;          // predvolba reload citace 40, vysledna frekvence 50 kHz
  TIMSK |= (1<<OCIE2);  // interrupt on Output Compare Match 2

  LastIRC = IRC_PORT & MASK_PIN_AB;      // cteni aktualniho stavu IRC
  if ((LastIRC & MASK_PIN_B) != 0)         // dekodovani grayova kodu,
    LastIRC ^= MASK_PIN_A;               // uschovani stavu

  EncoderPosition = 0;                   // nulovani pozice enkoderu
}
```

IntIRC.s

```
#include <io.h>
#include <ctoasm.inc>
#include <macros.inc>

#include "IntIRC.h"

.section    .text

; ====================================================================
;
;     Prerusovaci rutina casovace zpracovani IRC snimace
;
;     Timer 2 ... clock base 50 kHz
;
;     Casova narocnost:
;           Dekodovani IRC          beze zmen   35 taktu    2.1875 uS
;                                   +/-         53 taktu    3.31   uS
;                                   2* +/-      55 taktu    3.4375 uS
;
;     Maximální vytizeni procesoru 17,5%
;
; ====================================================================

.global     SIG_OUTPUT_COMPARE2
.func       SIG_OUTPUT_COMPARE2

#define     IRC   ZL
#define     temp  YH

SIG_OUTPUT_COMPARE2:

IntIRC:

  push      r0
  in        r0, _SFR_IO_ADDR(SREG)
  push      ZH
  push      ZL
  push      temp

  in        IRC, _SFR_IO_ADDR(IRC_PORT)   ; cteni aktualniho stavu IRC

  ldi       temp, MASK_PIN_A              ; dekodovani grayova kodu
  sbrc      IRC, PIN_B                    ; je-li vyssi bit 1
  eor       IRC, temp                     ;   pak nizsi bit negovat
  andi      IRC, MASK_PIN_AB              ; maskovat pouze bity A, B

  lds       temp, LastIRC                 ; vyzvednuti minuleho stavu
  sts       LastIRC, IRC                  ; a ulozeni noveho

  sub       IRC, temp                     ; rozdil aktualniho a minuleho stavu
  andi      IRC, MASK_PIN_AB              ; maskovat pouze bity A,B (mozne přetečeni)

  breq      enc_zero                      ; rozdil je 0, nedoslo k pohybu
  cpi       IRC, MASK_PIN_A               ; rozdil stavu je "1"
  breq      enc_inc                       ;   pricist tik
  cpi       IRC, MASK_PIN_AB              ; rozdil stavu je "-1"
  breq      enc_dec                       ;   odecist tik
```

```
  lds       ZL, PreviousDir             ; rozdil stavu je "2", ztrata stavu
  mov       ZH, ZL                      ;   zjistit prechozi smer pohybu
  lsl       ZL                          ; nasobit 2-mi,
  asr       ZH                          ; pricteme, nebo odecteme 2 tiky
  rjmp      enc_add

enc_inc:
  ldi       ZL, 1                       ; budeme pricitat 1 tik
  clr       ZH
  rjmp      enc_store

enc_dec:
  ldi       ZL, -1                      ; budeme odecitat 1 tik
  mov       ZH, ZL

enc_store:
  sts       PreviousDir, ZL             ; uschova aktualniho smeru pohybu

enc_add:
  lds       temp, EncoderPosition       ; vypocet nove hodnoty pozice
  add       temp, ZL                    ;   enkoderu
  sts       EncoderPosition, temp
  lds       temp, EncoderPosition+1
  adc       temp, ZH
  sts       EncoderPosition+1, temp

enc_zero:
  pop       temp
  pop       ZL
  pop       ZH
  out       _SFR_IO_ADDR(SREG), r0
  pop       r0

  reti

.endfunc
```

Informační zdroje:

http://robotika.cz/guide/

http://elm-chan.org/works/smc/report_e.html - inspirace pro dekódování IRC jako grayova kódu a velice zajímavě řešený kaskádní PID regulátor