▸ Dieser Artikel
- Seite
- Diskussion
- Bearbeiten
- Versionsgeschichte

▸ Benutzer
- Nicht angemeldet
- Diskussionsseite
- Beiträge
- Benutzerkonto erstellen
- Anmelden

▸ Suche
Ausführen Suche

▸ Werkzeuge
- Links auf diese Seite
- Änderungen an verlinkten Seiten
- Spezialseiten

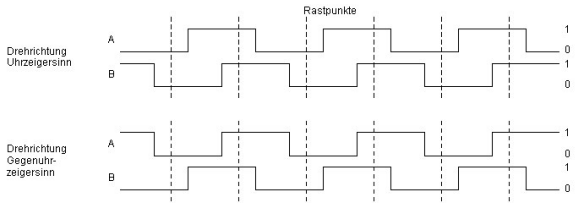
Drehgeber

Aus der Mikrocontroller.net Artikelsammlung, mit Beiträgen verschiedener Autoren (siehe Versionsgeschichte)

Drehgeber (auch Inkrementaldrehgeber, Quadraturencoder, Drehencoder, Drehimpulsgeber genannt) dienen der dynamischen Erfassung von Winkeländerungen bei Achsen und Wellen. Sie werden sowohl für die manuelle Eingabe von Werten, als auch zur Ermittlung von Drehgeschwindigkeiten eingesetzt.

Inhaltsverzeichnis

Funktion



Drehgeber erzeugen bei Drehung der Achse an zwei Datenleitungen am Ausgang ein sogenanntes Gray-Code-Signal. Der Vorteil dieser Codierung ist, dass ein Entprellen deutlich einfacher wird, da dieser Code die Eigenschaft besitzt, dass sich zwischen benachbarten Codes nur jeweils ein Bit ändert. Dies ermöglicht die asynchrone Abtastung, ohne weiter als einen Schritt vom wahren Ergebnis entfernt zu sein, weil ein sich änderndes Bit lediglich verspätet, aber nicht falsch erfasst wird.

Differenzierung von Drehgebern

Man muss unterscheiden zwischen Drehgebern, die von schnellaufenden Maschinen angetrieben werden z. B. zum Messen des Drehwinkels und solchen die von Hand bedient werden z. B. zum Einstellen der Lautstärke. Letztere haben in der Regel Rastpunkte, bei denen der Knopf leicht "einschnappt". Hierdurch fühlt der Benutzer das Fortschreiten des zu bedienenden Vorgangs z. B. das Hochregeln der Lautstärke. Die maschinengetriebenen Drehgeber sind jedoch in der Regel frei drehbar und haben keine mechanischen Rastpunkte.

Handbediente Drehgeber

Aus Kosten- und Platzgründen sind handbediente Drehgeber häufig als schaltende Kontakte ausgeführt, was die Gefahr des Prellens mit sich bringt. Die erste dagegen ergriffene Maßnahme ist die oben genannte Nutzung eines Codes, bei dem sich nie zwei Bits gleichzeitig ändern. Diese Maßnahme allein ist aber nicht ausreichend.

Das oben gezeigte Signaldiagramm zeigt mit den senkrecht gestrichelten Linien die Rastungen so an, wie sie für Handbedienung verbreitet sind, nämlich mit zwei Codewechseln zwischen den Rastungen. Die Software sollte dies berücksichtigen und pro Rastung nur um einen Schritt fortschreiten, was bedeutet, dass nur zwei vollendete aufeinanderfolgende elektrische Wechsel ausgewertet werden. Dies hat den Vorteil, dass ein kurzfristiges Prellen eines Kontaktes durch Erschütterung des Encoders z. B. durch vibrierende Maschinen oder durch Kontaktprellen beim Drehen nicht zu einer Fehlinterpretation kommt. Dennoch ist die Reaktionszeit kurz, da verzögernde Timerschleifen zum Entprellen überflüssig sind. Die hierbei auftretende Halbierung der Auflösung ist wegen des durch menschliche Fähigkeiten begrenzt dosierbaren Drehwinkels irrelevant. Zudem entsteht so haptische Übereinstimmung zwischen der Anzahl der gefühlten Rastungen und dem Fortschreiten des zu bedienenden Prozesses, sofern der bediente Wert entsprechend inkrementiert wird.

Es sei hier aber angemerkt, dass es auch handbediente Drehgeber mit anderer Anordnung der Rastungen gibt, die somit auch eines anderen Algorithmus bedürfen. Bei manchen Billigprodukten fallen die Rastpunkte mit den Zustandsübergängen zusammen.

Maschinengetriebene Drehgeber

Der maschinengetriebene Drehgeber hat in der Regel keine Rastpunkte. Zudem dreht er sich häufig so schnell, dass für entprellende Maßnahmen nach dem Verzögerungsprinzip (Kondensator, Delayschleife oder zyklischer Timerüberlaufinterrupt) die Zeit fehlt. Daher sind solche Drehgeber in aller Regel zum Zwecke der prellfreien Ausgänge aufwendiger und somit teurer konstruiert. Realisiert wird dies durch kleines mechanisches Spiel, Leichtgängigkeit und optische Maßnahmen. Diese Prellfreiheit ist aber dann nutzlos, wenn der Drehgeber von einer Mechanik getrieben wird, welche Vibrationen bisweilen mikroskopischer Größe aufweist, z. B. ausgelöst durch mechanisches Spiel. Diese Vibration ist der Drehbewegung mechanisch überlagert. Dies kann dazu führen, dass beim Überschreiten eines elektrischen Schaltpunktes mikroskopisch mehrfach zurück und wieder vorgedreht wird und somit ein dem Prellen ähnliches Ereignis auftritt. Dies kann durch Auswertung zweier vollendeter aufeinanderfolgender elektrischer Wechsel vermieden werden. Dies stellt nach heutigem Kenntnisstand (2009) die schnellste Reaktionsmöglichkeit dar. Will man hierdurch nicht die Auflösung halbieren (wie es bei den oben genannten handbedienten Drehgebern passiert), müssen jeweils parallel zwei aufeinanderfolgende Impulse um die Periodenlänge eines Schaltimpulses phasenverschoben in zwei getrennten "Pipelines" überwacht werden. Vereinfacht ist dies hier für eine Drehrichtung illustriert. Die Berücksichtigung beider Drehrichtungen ist komplexer:

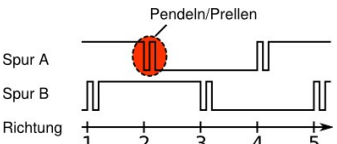
```
__|--|--|__|__
__|__|--|--|__
   |  |  |  |
   1  2  3  4
```

Pipeline1 überwacht die vollendete Schaltfolge 1-2. Pipeline2 überwacht die vollendete Schaltfolge 2-3. Pipeline1 überwacht die vollendete Schaltfolge 3-4 etc. Jede der beiden Pipelines gibt einen Zählimpuls ab, so dass trotz Abwartens von zwei aufeinander folgenden Schaltpunkten bei jedem Schaltpunkt ein Fortschreitungsimpuls (Zählimpuls) erfolgt. Letztendlich liefert das Verfahren beim Wechsel zwischen Vor- und Rückwärtsdrehen eine mechanische Hysterese von der Länge eines Schaltzustandes. Dies ist aber aufgrund der Vibrationen bzw. des mechanischen Spieles unvermeidbar. Der Drehgeber kann eben nicht genauer sein, als die ihn treibende Mechanik.

Insgesamt gesehen müssen die ergriffenen Maßnahmen auf die mechanischen Gegebenheiten und die Anforderungen an Auflösung und Geschwindigkeit angepasst werden. Die Angabe eines sehr aufwendigen parametrierbaren Algorithmus (dessen Wahl der Parameter wiederum nicht trivial wäre), der in allen Lebenslagen funktioniert, wäre zwar möglich, ist aber mit Blick auf die hohe Prozessorlast nicht angezeigt. In Extremfällen empfehlen sich möglicherweise FPGAs zur Auswertung. "Einigermaßen langsame" Anwendungen und geringe Auflösungen können aber mit denselben Methoden erschlagen werden, die sich auch für den handbetriebenen Drehgeber eignen und finden sich im Artikel unten.

Signalauswertung

Im folgenden ein Bild, wie ein reales Signal eines Drehgebers aussehen kann:



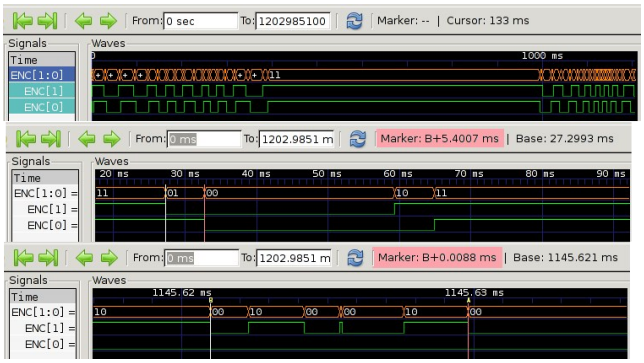
Die Auswertung eines Drehgebers macht man am besten in einem Timer-Interrupt, der mit einer festen Frequenz ausgeführt wird. Die Abtastfrequenz muss so hoch sein, dass bei maximaler Drehzahl zwischen zwei Codewechseln mindestens 1 Abtastung erfolgt, besser jedoch mehr.

Es gibt bei manchen Mikrokontrollern Timer, die Drehencodersignale direkt erfassen können. Im Datenblatt bzw. User Manual sollte das unter den Stichworten "quadrature" oder "incremental" Encoder zu finden sein.

Nachfolgend findet sich die Signalfolge eines realen mechanischen ALPS-Encoders (30 Raststellungen pro Umdrehung), mit einem Logikanalysator aufgenommen. Gezeigt ist dabei ca. eine halbe Umdrehung im Uhrzeigersinn, gefolgt von einer Drehung in Gegenrichtung. Das benutzte Value-Change-Dump-Dateiformat kann beispielsweise von GTKwave dargestellt werden.

Medium:Rotaryenc.vcd

Das folgende Bild zeigt dieses Signal als GTKwave-Darstellung:



Ganz oben das Signal im Ganzen. Man erkennt deutlich den Versatz der beiden Spuren, der je nach Drehrichtung unterschiedlich ist.

In der Mitte ist ein Ausschnitt, der beide Spuren beim Wechsel von einem Rastpunkt auf den nächsten zeigt. Die (hervorgehobene) Zeitdifferenz zwischen Cursor und B-Markierung im oberen Bereich zeigt, dass dabei der Code zwischen den Rastpunkten innerhalb weniger Millisekunden durchlaufen wird. Bei der Auswertung muss man daher sicher sein, diesen Bereich mindestens einmal abgetastet zu haben.

Der benutzte ALPS-Drehgeber prellt verhältnismäßig wenig. Die meisten Übergänge von einem Code zum nächsten lassen in der benutzten Auflösung des Logikanalysators (10 ns) kein Prellen erkennen. In den wenigen Fällen, in denen ein Prellen zu finden ist, bewegt sich dieses im Bereich einiger Mikrosekunden, wie im unteren Teil dargestellt.

Warum Sparvarianten nicht gut sind

Oft sieht man im Netz "clevere" Sparvarianten, welche angeblich ebensogut zur Auswertung von Drehgebern geeignet sind. Ein genaueres Hinschauen sowie Tests unter realen Bedingungen zeigen jedoch schnell die Schwächen dieser Ansätze.

Flankenerkennung von A und Pegelauswertung von B

Viele Sparvarianten verwenden einen externen Interrupt, welcher auf die steigende oder fallende Flanke von Spur A auslöst und dann den Pegel von B auswertet. Ist B=0, dann dreht der Encoder nach rechts, anderenfalls nach links. Diese Auswertung hat zwei Schwachstellen.

1. Die Auflösung wird auf ein Viertel reduziert, weil nur jede steigende Flanke von A ausgewertet wird.
2. Pendelt der Encoder zwischen zwei Codes, bei denen A seinen Pegel wechselt,
    1. kommt es zu (sehr) vielen Interrupts, die den Mikrocontroller vollkommen auslasten können.
    2. interpretiert die Auswertung jede steigende Flanke als neuen Schritt. Der Encoder scheint sich für die Auswertung immer weiter zu drehen (wenn man nicht prüft, ob auch B den Pegel ändert), obwohl er nur pendelt.

Das Pendeln kann zwei Ursachen haben.

1. Der Encoder pendelt wirklich; das kann z. B. bei hochauflösenden Encodern ohne Rastung geschehen, welche an jeder beliebigen Stelle stehen bleiben können und durch geringe mechanische Erschütterungen dann zwischen zwei Codes pendeln; das kann z. B. bei hochauflösenden Encodern in CNC-Maschinen der Fall sein.
2. Die Signale prellen; das kommt vor allem bei billigen elektromechanischen Drehknöpfen vor, welche einfache Schleifkontakte zur Kodierung nutzen.

Wie man sieht ist diese Methode nicht geeignet, einen Drehgeber solide zu dekodieren.

Auswertung mit Interrupt durch Pegelwechsel

Es wird bisweilen die Auffassung vertreten, dass mit Hilfe von sog. Pin Change Interrupts Rechenzeit gespart werden kann. Dabei wird bei einem Pegelwechsel von Spur A oder B ein Interrupt erzeugt. Dort werden dann A und B eingelesen und vollständig ausgewertet. Diese Methode ist besser, aber nicht gut genug. Sie vermeidet Fehler 1. und 2.2 der oben genannten Auswertung, aber nicht 2.1, da auch sie durch einen pendelnden/prellenden Encoder die CPU stark belastet.

Solide Lösung: Beispielcode in C

```
/*****************************************************************************/
/*                                                                           */
/*                      Reading rotary encoder                               */
/*                      one, two and four step encoders supported            */
/*                                                                           */
/*              Author: Peter Dannegger                                      */
/*              target: ATmega16                                             */
/*****************************************************************************/
#include <avr/io.h>
#include <avr/interrupt.h>

#define XTAL        8e6         // 8MHz

#define ENCODER_PIN PINA
#define PHASE_A     (1<<PA1)
#define PHASE_B     (1<<PA3)

#define LEDS_DDR    DDRC
#define LEDS        PORTC           // LEDs against VCC

volatile int8_t enc_delta;          // -128 ... 127
static int8_t last;

void encode_init( void ) {
  int8_t new, tmp;

  // init encoder
  tmp = ENCODER_PIN;
  new = 0;
  if( tmp & PHASE_A ) new  = 3;
  if( tmp & PHASE_B ) new ^= 1;       // convert gray to binary
  last = new;                         // power on state
  enc_delta = 0;

  // init timer 0
  TCCR0 = (1<<WGM01) | (1<<CS01) | (1<<CS00);   // CTC, prescaler 64
  OCR0 = (uint8_t)(XTAL / 64.0 * 1e-3 - 0.5);   // 1ms
  TIMSK |= 1<<OCIE0;
}

ISR( TIMER0_COMP_vect ) {           // 1ms for manual movement
  int8_t new, diff, tmp;
```

```
  tmp = ENCODER_PIN;
  new = 0;
  if ( tmp & PHASE_A ) new  = 3;
  if ( tmp & PHASE_B ) new ^= 1;   // convert gray to binary
  diff = last - new;               // difference last - new
  if( diff & 1 ) {                 // bit 0 = value (1)
    last = new;                    // store new as next last
    enc_delta += (diff & 2) - 1;   // bit 1 = direction (+/-)
  }
}

// read 1, 2, or 4 step encoders
int8_t encode_read( uint8_t step )    {
  int8_t val;

  // atomic access to enc_delta
  cli();
  val = enc_delta;
  switch (step) {
    case  2: enc_delta = val & 1; val >>= 1; break;
    case  4: enc_delta = val & 3; val >>= 2; break;
    default: enc_delta = 0; break;
  }
  sei();
  return val;                   // counts since last call
}

int main( void ) {
  int32_t val = 0;

  PORTA |= PHASE_A | PHASE_B;       // activate internal pull up resistors
  LEDS_DDR = 0xFF;
  encode_init();
  sei();

  for(;;){
    val += encode_read(1);          // read a single step encoder
    LEDS = val;
  }
}
```

Je nach Encodertyp ruft man encode_read() mit der passenden Anzahl elektrischer Codes/Rastung auf. Für manuelle Eingabe ist ein Abfrageintervall von 1ms meist ausreichend. Das Auslesen im Hauptprogramm mit den Funktionen encode_read() muss mit mindesten 127tel der Frequenz des Timers erfolgen, hier im Beispiel mit 1kHz/127 ~ 8Hz. Ansonsten können im Extremfall Überläufe der Variable enc_delta auftreten und zu Fehlfunktionen des Programms führen.

- dse-FAQ
- Forumsbeitrag: Drehgeber auslesen
- Forumsbeitrag: Drehimpulsgeber mit Rasterstellung bei 00/11 auswerten

Kurze Erklärung zur Funktionsweise des Codes: Unter der Voraussetzung, dass die Abtastung häufig genug erfolgt, dass in jeder Code-Kombination mindestens einmal abgetastet wird, kann sich zwischen dem aktuellen und dem zurückliegenden Wert als Differenz immer nur 0, 1, -1, 3 oder -3 ergeben. (Eine 2 würde bedeuten, dass die Abtastung einen Schritt übersprungen hat.) Dabei stellen 1 und -3 die Bewegung in eine Richtung, -1 und 3 in die andere Richtung dar. Deren Bitmuster sind 0b0…001, 0b1…101 bzw. 0b1…111, 0b0…011. Damit wird klar, dass man Bit 0 (diff & 1) als Kennzeichen heranziehen kann, dass sich in diesem Schritt überhaupt etwas geändert hat, und Bit 1 (diff & 2) als Kennzeichen für die Richtung der Änderung.

Ferner ist beim Auswerten zu beachten, dass die beiden Signalleitungen des Drehgebers möglichst zeitgleich erfasst werden. Dies ist insbesondere dann wichtig, wenn der Drehgeber zusammen mit weiteren Bedienelementen an einen Multiplexer angeschlossen ist! Das wird hier durch die Zwischenvariable tmp erreicht.

Dekoder für Drehgeber mit wackeligen Rastpunkten

Im wahren Leben gibt es immer wieder Dinge, welche der Theorie zwar widersprechen, dennoch weit verbreitet sind. Da machen Drehgeber keine Ausnahme. Gerade die heute so beliebten Drehgeber für manuelle Bedienung sind in großer Anzahl von verschiedenen Herstellern verfügbar. Umso merkwürdiger ist es, dass hier die Rastpunkte oft **genau** auf dem Pegelwechsel einer Spur liegen, meist Spur B. So zum Beispiel beim Drehgeber EC11E15244B2 von Alps, welcher u.a bei Pollin ehältlich ist.

Bei diesem Drehgeber kommt es bei der klassischen, eigentlich soliden Auswertung zu dem Effekt, dass der Drehgeber in Ruhelage auf einem Rastpunkt pendeln kann. Damit erhält das Programm sporadisch einen Schritt vor und zurück. Auch wenn sich die Auswertung daran nicht verschluckt, so ist dieses Pendeln doch ärgerlich, denn eine Menusteuerung würde dann komische Sachen machen.

Die solide Lösung des Problems ist recht einfach. Man wertet in der bekannten Manier weiterhin die abgetasteten Spuren A und B aus, allerdings mit der Änderung, dass man nur die Pegelwechsel der Spur A auswertet. Damit halbiert man zwar die Auflösung, das ist hier aber paradoxerweise gut! Denn damit bekommt man automatisch genau einen Zählimpuls pro Rastpunkt.

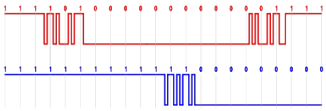
Praktisch heisst das, dass man lediglich die Dekodertabelle für die Auswertung ändern muss. Beachtet werden muss jedoch, dass man bei den "wackeligen" Drehgebern Spur A und B **nicht** beliebig vertauschen kann. Ist also die Auswertung immer noch wackelig, muss man im Quelltext die Defines für PHASE_A und PHASE_B vertauschen. Ein Blick ins Datenblatt des Drehgebers sollte auch hier helfen.

```
/*************************************************************************/
/*                                                                       */
/*            Drehgeber mit wackeligem Rastpunkt dekodieren              */
/*                                                                       */
/*************************************************************************/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>

#define XTAL        8e6                 // 8MHz

#define ENCODER_PIN PINA                // an Pinbelegung anpassen
#define PHASE_A     (1<<PA1)
#define PHASE_B     (1<<PA3)

#define LEDS_DDR    DDRC
#define LEDS        PORTC               // LEDs gegen Vcc geschaltet

volatile int8_t enc_delta;              // Drehgeberbewegung zwischen
                                        // zwei Auslesungen im Hauptprogramm

// Dekodertabelle für wackeligen Rastpunkt
// halbe Auflösung
const int8_t table[16] PROGMEM = {0,0,-1,0,0,0,0,1,1,0,0,0,0,-1,0,0};

// Dekodertabelle für normale Drehgeber
// volle Auflösung
//const int8_t table[16] PROGMEM = {0,1,-1,0,-1,0,0,1,1,0,0,-1,0,-1,1,0};

ISR( TIMER0_COMP_vect ) {               // 1ms fuer manuelle Eingabe
    static int8_t last=0;               // alten Wert speichern
    uint8_t tmp;

    tmp = ENCODER_PIN;
    last = (last << 2)  & 0x0F;
    if (tmp & PHASE_A) last |= 2;
    if (tmp & PHASE_B) last |= 1;
    enc_delta += pgm_read_byte(&table[last]);
}

void encode_init( void ) {              // nur Timer 0 initialisieren
  TCCR0 = (1<<WGM01) | (1<<CS01) | (1<<CS00);      // CTC, XTAL / 64
  OCR0 = (uint8_t)(XTAL / 64.0 * 1e-3 - 0.5);      // 1ms
  TIMSK |= 1<<OCIE0;
}
```

```
int8_t encode_read( void ) {         // Encoder auslesen
  int8_t val;

  // atomarer Variablenzugriff
  cli();
  val = enc_delta;
  enc_delta = 0;
  sei();
  return val;
}

int main( void ) {
  int32_t val = 0;

  PORTA |= PHASE_A | PHASE_B;        // activate internal pull up resistors
  LEDS_DDR = 0xFF;
  encode_init();
  sei();

  while(1){
    val += encode_read();
    LEDS = ~val;
  }
}
```

Automatische Entprellung bei Abtastung (state machine)

Das folgende Bild zeigt einen möglichen Signalverlauf der beiden Spuren, Abtastzeitpunkte und die dazugehörigen digitalen Zustände.



Nimmt man an, dass die obere rote Linie Spur A darstellt und gemäß obigen C-Code Spur A das MSB ist, so erhält man folgende Abtastungen:

11,11,11,
<A_prellt>11,01,11</A_prellt>,
01,01,01,01,01,
<B_prellt>01,01</B_prellt>,
00,00,00,00,
<A_prellt>00,10</A_prellt>,
10,10,10

Es fällt sofort auf, dass nur das erste Prellen von A wirklich "böse" ist.

Nimmt man den obigen C-Code als Beispiel, wird dort immer aus dem letzten Zustand und dem jetzigen Zustand ein halbes Byte generiert und mit diesem dann ein Wert aus einer lookup table geholt (die Addition dieser Werte ergibt die Anzahl [oder ein vielfaches, je nach Drehgeber] der gedrehten Schritte).

Bei obigen Signalverlauf würden diese Bitmuster zur Auswertung kommen:

3 mal 1111 -> table[15] = +0
<böses_prellen>
1 mal 1101 -> table[13] = -1
1 mal 0111 -> table[07] = +1
1 mal 1101 -> table[13] = -1
</böses_ prellen>
6 mal 0101 -> table[05] = +0
1 mal 0100 -> table[04] = +0 oder -1 bei doppelter Auflösung
4 mal 0000 -> table[00] = +0
1 mal 0010 -> table[02] = -1
3 mal 1010 -> table[10] = +0

Betrachtet man nun nur die Stelle an der das Prellen auftritt so sieht man, dass das Prellen heraus gerechnet wird, denn egal wie oft der Wert springt wird am Ende -1 übrig bleiben.

Der Gray-Code bewirkt, dass nur eine Spur prellt während die andere einen definierten Pegel behält. Es kann auch bei auftretendem Prellen zuerst nur ein Wert der dem zuletzt Abgetasteten entspricht (+0) oder ein Wert der dem nächsten Zustand entspräche (-1) abgetastet werden. Würde, durch das Prellen bedingt, doch wieder der alte Zustand dekodiert (+1), so muss diesem ja irgendwann auch wieder der nächste Zustand folgen (-1)[es sei denn der Drehgeber verpufft plötzlich]. Daher ist, in diesem Fall, immer eine (-1) übrig.

Beispielcode in VHDL

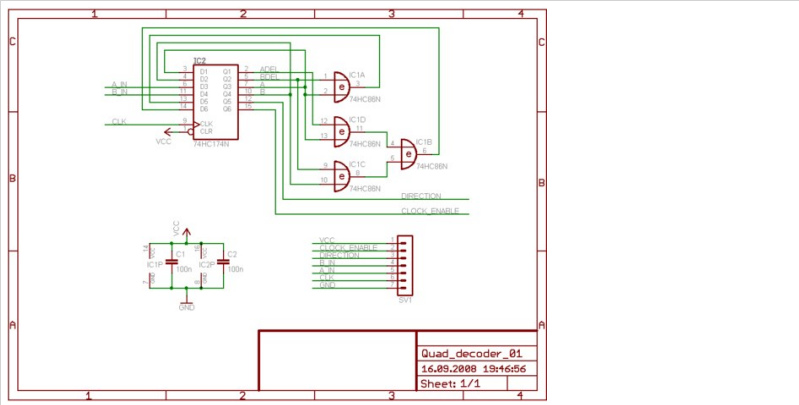
Besonders bei höheren Winkelgeschwindigkeiten und Auflösungen ist eine Auswertung in Software in einem Mikrocontroller irgendwann einmal technisch unmöglich. Dann muss ein Dekoder in Hardware her, heutzutage meist programmierbare Logik in Form eines CPLD oder FPGA. VHDL ist eine weit verbreitete Sprache zur Logikbeschreibung bzw. Synthese. Der folgende Code tastet die beiden Quadratursignale ab und generiert daraus ein UP/DOWN Signal sowie ein Clock Enable für einen Zähler, mit dem dann die aktuelle Position erfasst werden kann. Zusätzlich wird ein illegaler Signalübergang signalisiert, was meist auf einen defekten Drehgeber oder zu niedrige Abtastfrequenz hinweist.

```
--------------------------------------------------------------------------------
--
-- Decoder für Drehgeber
--
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity incremental_encoder is
    Port ( clk      : in std_logic;          -- Systemtakt
           A        : in std_logic;          -- Spur A
           B        : in std_logic;          -- Spur B
           up_down  : out std_logic;         -- Zaehlrichtung
           ce       : out std_logic;         -- Clock Enable
           error    : out std_logic);        -- illegaler Signalübergang
end incremental_encoder;

architecture Behavioral of incremental_encoder is

signal a_in, b_in, a_old, b_old: std_logic;

begin

-- Abtastung und Verzoegerung der Quadratursignale

process(clk)
begin
  if rising_edge(clk) then
    a_old <= a_in;
    a_in  <= A;
    b_old <= b_in;
    b_in  <= B;
  end if;
end process;

-- Dekodierung der Ausgaenge

process(a_in, b_in, a_old, b_old)
variable state: std_logic_vector(3 downto 0);
```

```
begin
  state := a_in & b_in & a_old & b_old;
  case state is
    when "0000" => up_down <= '0'; ce <= '0'; error <= '0';
    when "0001" => up_down <= '1'; ce <= '1'; error <= '0';
    when "0010" => up_down <= '0'; ce <= '1'; error <= '0';
    when "0011" => up_down <= '0'; ce <= '0'; error <= '1';
    when "0100" => up_down <= '0'; ce <= '1'; error <= '0';
    when "0101" => up_down <= '0'; ce <= '0'; error <= '0';
    when "0110" => up_down <= '0'; ce <= '0'; error <= '1';
    when "0111" => up_down <= '1'; ce <= '1'; error <= '0';
    when "1000" => up_down <= '1'; ce <= '1'; error <= '0';
    when "1001" => up_down <= '0'; ce <= '0'; error <= '1';
    when "1010" => up_down <= '0'; ce <= '0'; error <= '0';
    when "1011" => up_down <= '0'; ce <= '1'; error <= '0';
    when "1100" => up_down <= '0'; ce <= '0'; error <= '1';
    when "1101" => up_down <= '0'; ce <= '1'; error <= '0';
    when "1110" => up_down <= '1'; ce <= '1'; error <= '0';
    when "1111" => up_down <= '0'; ce <= '0'; error <= '0';
    when others => null;
  end case;
end process;

end Behavioral;
```

Dekoder mit diskreten Logik-ICs

Für einige Anwender sind Mikrocontroller und programmierbare ICs bisweilen zu komplex oder aus anderen Gründen nicht nutzbar. Dann braucht man eine Lösung mit klassischen Logikbausteinen. Aber auch das ist recht leicht gemacht. Zwei kleine ICs genügen, die Schaltung benötigt allerdings einen externen Takt, welcher von nahezu jeder beliebigen Quelle erzeugt werden kann. Zu beachten ist, dass das Signal DIRECTION nur gültig ist, wenn das Signal CE aktiv (= HIGH) ist. Das ist auf eine vereinfachte Dekodierung zurückzuführen, im Unterschied zur VHDL-Lösung. Ausserdem handelt es sich bei CE um ein Clock Enable Signal, nicht um einen Takt, siehe Taktung FPGA/CPLD. Mit diesem Dekoder kann man dann einen Vorwärts/Rückwärts-Zähler ansteuern, wie er in diesem Beitrag zu sehen ist.



Diskret aufgebauter Dekoder

Das Projekt im Eagleformat kann auf einer kleinen einseitigen Platine aufgebaut werden, es sind nur vier Lötbrücken notwendig (Rote Leitungen im TOP-Layer).

Und so kommt man auf die Schaltung: Das Taktsignal soll bei einer Flanke schalten - eine Flanke im Signal A kann man z.B. mit XOR durch $A^{-1} \oplus A$ (A^{-1} ist dabei das A aus dem letzten Takt, durch ein D-FlipFlop um einen Takt verzögert) ausdrücken, für B analog. Die Fälle "nichts passiert" und "ungültig" (also zwei Flanken gleichzeitig) schließt man ebenfalls durch XOR aus, d.h. das Taktsignal schaltet entweder bei einer Flanke in A oder bei einer Flanke in B, also:

$$Clock = (A^{-1} \oplus A) \oplus (B^{-1} \oplus B).$$

Das Richtungssignal erhält man durch Umformung des logischen Ausdrucks, dieser ergibt sich zunächst aus der Situationstabelle (oder aus dem Graphen) zu:

$$Dir = A^{-1}AB^{-1}B \vee A^{-1}AB^{-1}B \vee A^{-1}AB^{-1}B \vee A^{-1}AB^{-1}B$$

$$Dir = A^{-1}B(AB^{-1} \vee AB^{-1}) \vee A^{-1}B(AB^{-1} \vee AB^{-1})$$

$$Dir = A^{-1}B(A \oplus B^{-1}) \vee A^{-1}B(A \oplus B^{-1})$$

$$Dir = (A \oplus B^{-1})(A^{-1}B \vee A^{-1}B)$$

$$Dir = (A \oplus B^{-1}) \vee (A^{-1} \oplus B)$$

Durch scharfes Hinsehen und Abgleich mit dem Diagramm sieht man jetzt zwei Sachen: Erstens, dass die Formel für jede Flanke gilt, es sich also tatsächlich um eine 4fach-Auswertung handelt und zweitens, dass der zweite Term immer mit dem ersten überein stimmt. Daraus folgt schließlich das Ergebnis mit

$$Dir = (A \oplus B^{-1}).$$

Siehe auch

- Forumsbeitrag: Mehrere Drehgeber abfragen
- Forumsbeitrag: Ein Gray Encoder, welcher in die andere Richtung arbeitet und aus Takt/Richtung die Signale A und B generiert
- Forumsbeitrag: Gray-Code Generator in Postscript von eProfi
- Forumsbeitrag: Beschleunigung für Drehgeber
- Forumsbeitrag: Dekodierung von vier Drehgebern mit ATtiny2313 mit bis zu 869,5 kHz und UART Übertragung.
- Forumsbeitrag: Emulation eines Drehgebers über zwei Tasten
- Forumsbeitrag: 0-10V Signal in Drehimpulsgeber Signal umwandeln
- Forumsbeitrag: Handoptimierte ISR in Assembler zur Drehgeberauswertung

Weblinks

- Optical encoder wheel generator (Onlinetool)
- Drehgeberauswertung mit Beschleunigung
- Drehinkrementalgeber: Allgemeine Betrachtung mit Beispielcode
- Quadraturdekoder mit ATTiny25 (einfache Version) oder ATtiny202, 4xx, 8xx für SIN/COS Dekoder

Download

- "Solide Lösung" als Bibliothek Datei:Encoder.zip

Kategorie:

- Bauteile