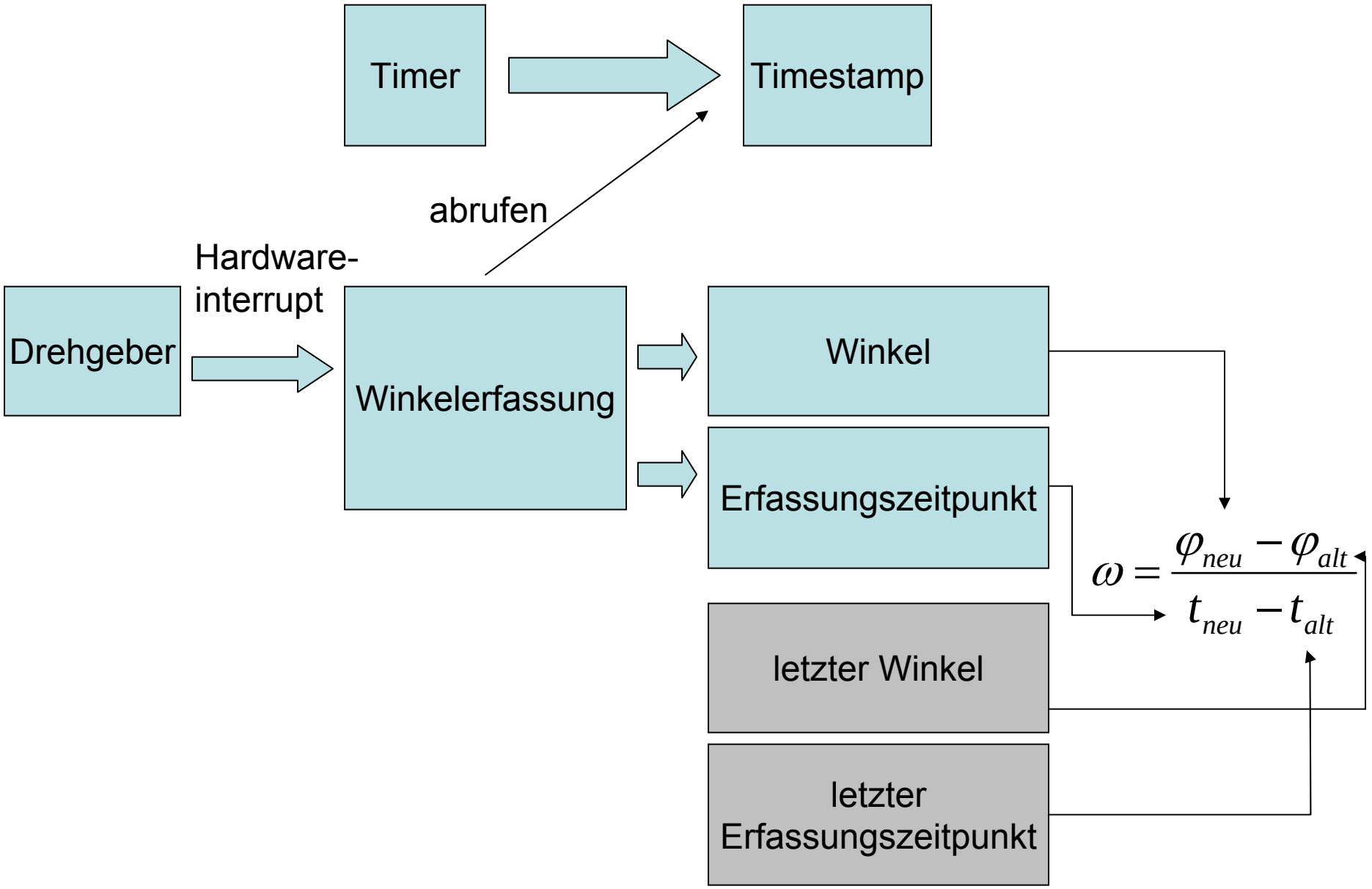
Aufbau einer rudimentären Geschwindigkeitsregelung für das COACH-Vehikel

Es soll eine verbesserte Drehzahlerfassung entsprechend dem invertierenden Pendel implementiert werden.

Der aktuelle Stand für die Klassen zur Ansteuerung der Peripherie findet sich unter folgendem Link:

http://www.kramann.info/72_COACH2/13_Skript/01_Funkkorrektur/coach2.2_06_funk_PA7_plus5Volt.zip

Aufbau des Reglers:

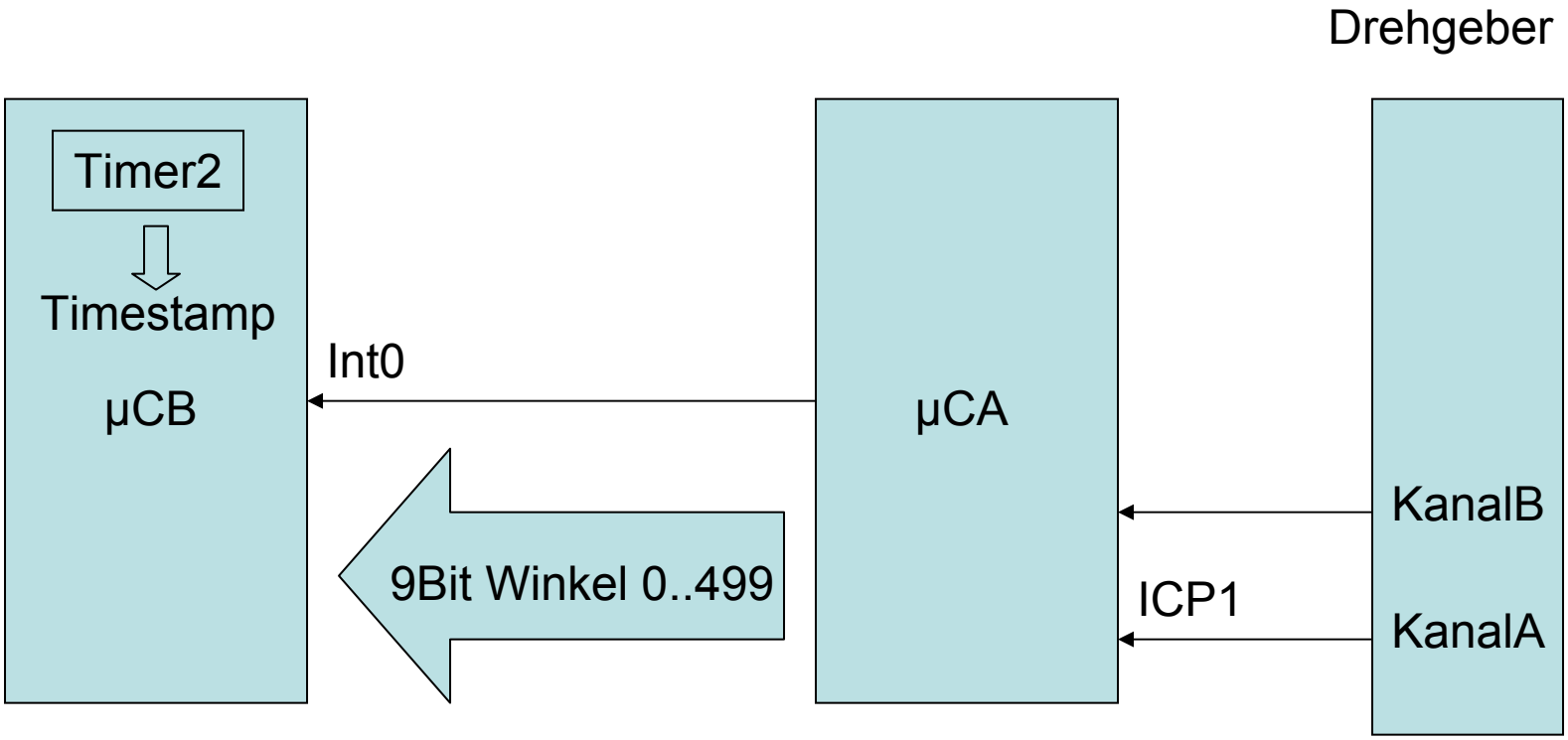


Aufbau des Reglers (Fortsetzung):

```
if( ω > ω_soll )
{
    PWM --;
}
else if ( ω < ω_soll )
{
    PWM ++;
}
```

Integrierender Regler: Bei Regelabweichungen wird die Geschwindigkeit solange verändert, bis sich Übereinstimmung einstellt.
Wichtig: PWM++ muß in einem nicht zu schnellen Zyklus ausgeführt werden.

Zunächst soll die Interrupt-Struktur bei der Mikrocontrollerschaltung des invertierenden Pendels analysiert werden.



Die Aktualisierung des Winkels erfolt bei µCB über Interrupt0. Der Timestamp wird mit Hilfe von Timer 2 generiert.

Die Erfassung der Drehgeberflanken erfolgt bei µCA über den Capturing-Eingang. Damit steht Timer1 auch nicht mehr für andere Dinge zur Verfügung.

Elemente der Winkelerfassung beim invertierenden Pendel

1. Timestamp erzeugen (µCB):

```
class Zeitgeber
{
    public:
        bool zustand;
        uint32_t zeitmarke;
        void start()
        {
            zustand = false;
            zeitmarke = 0;
            TCCR2 = (1<<FOC2) | (0<<WGM20) | (0<<COM21) | (0<<COM20) | (0<<WGM21)
                  | (0<<CS22) | (1<<CS21) | (0<<CS20);
            TCNT2=0; //Timer2 Zählregister Null setzen.
            TIMSK |= (1<<TOIE2); //Interrupt durch Timer2 aktivieren
        }
        void reset()
        {
            zustand = false;
            zeitmarke = 0;
            TCNT2=0;
        }
        void warten()
        {
            while(TCNT2<128);
            TCNT2=0;
        }
};
Zeitgeber zeitgeber;
SIGNAL(SIG_OVERFLOW2)  { zeitgeber.zeitmarke++; }
```

Elemente der Winkelerfassung beim invertierenden Pendel
2. Winkel und Erfassungszeitpunkt aktualisieren (µCB):

```
 …
 MCUCR |= (1<<ISC01);
 GICR  |= (1<<INT0);
 sei();
 …


SIGNAL(SIG_INTERRUPT0)
{
  drehgeber.zeitmarke_alt = drehgeber.zeitmarke;
  drehgeber.zeitmarke = zeitgeber.zeitmarke;
  drehgeber.dauer = drehgeber.zeitmarke - drehgeber.zeitmarke_alt;

  drehgeber.position = PINC;
  drehgeber.position |=  ((PIND & 0b00001000)<<5);
  drehgeber.position_alt = drehgeber.position;
}
```

Elemente der Winkelerfassung beim invertierenden Pendel 3. Drehgeber-Capturing (µCA)

```
 void start()
 {
            DDRA = 0b00000000;
            DDRB = 0b00000000;
            DDRC = 0b00000000;
            DDRD = 0b00000000;
            //Über PC0 (bei muecB an PD6 verbunden), kann der Drehgeber
            //resetet werden:
            DDRC &= 0b11111110;
            PORTC |= 0b00000001; //Pullup (Low-Potential soll auslösend sein.)

            //PD7 ist Ausgang, um bei muecB Interrupt auszulösen (bei PD2 / INT0)
            DDRD |= 0b10000000;
            //PD7 hochziehen (fallende Flanke löst bei muecB Interrupt aus!)
            PORTD |= 0b10000000;
            akku               = 0;
            position           = 0;
            erstes_mal         = true;
            referenz_erreicht = false;
            kanal_c_position  = 0;

            DDRA = 0b11111111;
            DDRC |= 0b10000000;

            //PD6==ICP1==KanalA als Eingang
            //und PD2==INT0==KanalB als Eingang: GEÄNDERT ZU PD5 !!!!
            //PD3==INT1==KanalC
            DDRD &=  0b10010011;
            //interne Pullup-Widerstände setzen:
            PORTD |= 0b01101100;
TCCR1B = (1<<ICNC1) | (0<<ICES1) | (0<<WGM13) | (0<<WGM12) | (1<<CS12) | (0<<CS11) | (1<<CS10);
              TIMSK |= (1<<ICF1);
              sei(); //Interrupts frei
 }
```

Elemente der Winkelerfassung beim invertierenden Pendel

3. Drehgeber-Capturing (µCA) Fortsetzung:

```
SIGNAL(SIG_INPUT_CAPTURE1)  //Capturing an ICP1 / PD6 von
{
    drehgeber.icr1 = ICR1;  //auslesen, um Interrupt wieder frei zu setzen.
    /* Save global interrupt flag */
    drehgeber.register_save = SREG;
    /* Disable interrupts */
    cli();

    drehgeber.akku = PIND;
    if( (drehgeber.akku & 0b00100000) > 0 ) //KanalB PD5 untersuchen
        drehgeber.position--;
    else
        drehgeber.position++;

    drehgeber.position+=500;  //auf Bereich 0..499 beschränken
    drehgeber.position%=500;

//KanalC / PD3 untersuchen:
    if( drehgeber.erstes_mal && (drehgeber.akku & 0b00001000) > 0 ) {
        drehgeber.kanal_c_position = drehgeber.position;
        drehgeber.erstes_mal = false;
        drehgeber.referenz_erreicht = true;
    }
    else if( !drehgeber.erstes_mal && (drehgeber.akku & 0b00001000) > 0 )
    {
        drehgeber.position = drehgeber.kanal_c_position;
        drehgeber.referenz_erreicht = true;
    }
```

Elemente der Winkelerfassung beim invertierenden Pendel

3. Drehgeber-Capturing (µCA) Fortsetzung 2:

```
//Wert des Drehgebers ausgeben:
    //muecA    -> muecB
    //PA0..PA7 -> PC0..PC7
    //PC7      -> PD3 / INT1
    PORTA = drehgeber.position;
    PORTC = drehgeber.position>>1;  //Bit8 von pos kommt auf Bit7 von PORTC
    /* Restore global interrupt flag */
    SREG = drehgeber.register_save;
    /* Enable interrupts*/
    sei();

    //Einige Taktzyklen diesen Zustand halten:
    DDRB = 0b00000000;
    DDRB = 0b00000000;
    DDRB = 0b00000000;
    //Signale an muecB, dass neuer Wert anliegt.
    //muecA -> muecB
    //PD7   -> INT0/PD2
    PORTD &= 0b01111111;

    //Einige Taktzyklen diesen Zustand halten:
    DDRB = 0b00000000;
    DDRB = 0b00000000;
    DDRB = 0b00000000;
    DDRB = 0b00000000;
    DDRB = 0b00000000;
    //..dann wieder hochziehen:
    PORTD |= 0b10000000;
}
```