Technische Universität Clausthal – Institut für Informatik

Reengineering des originalen Maschinenprogramms

Ming Wu

Peiji Liu

Hong Li

Technische Universität Clausthal
Institut für Informatik

Inhalt

- Kommunikationssystem
  - Kontrolle Methode
  - Download
  - EEPROM lesen
- Befehlsystem
  - ROBOBASIC
  - Befehlszusammenfassung
  - Befehls Analysierung
- Bewegungssystem
  - Frequenz definieren
  - Überprüfungsprozess
  - Winkeldrehungsprozess
  - Null rückstellen

Kontrolle Methode

Zwei Mögliche Kontrolle Methoden (Sub0093)

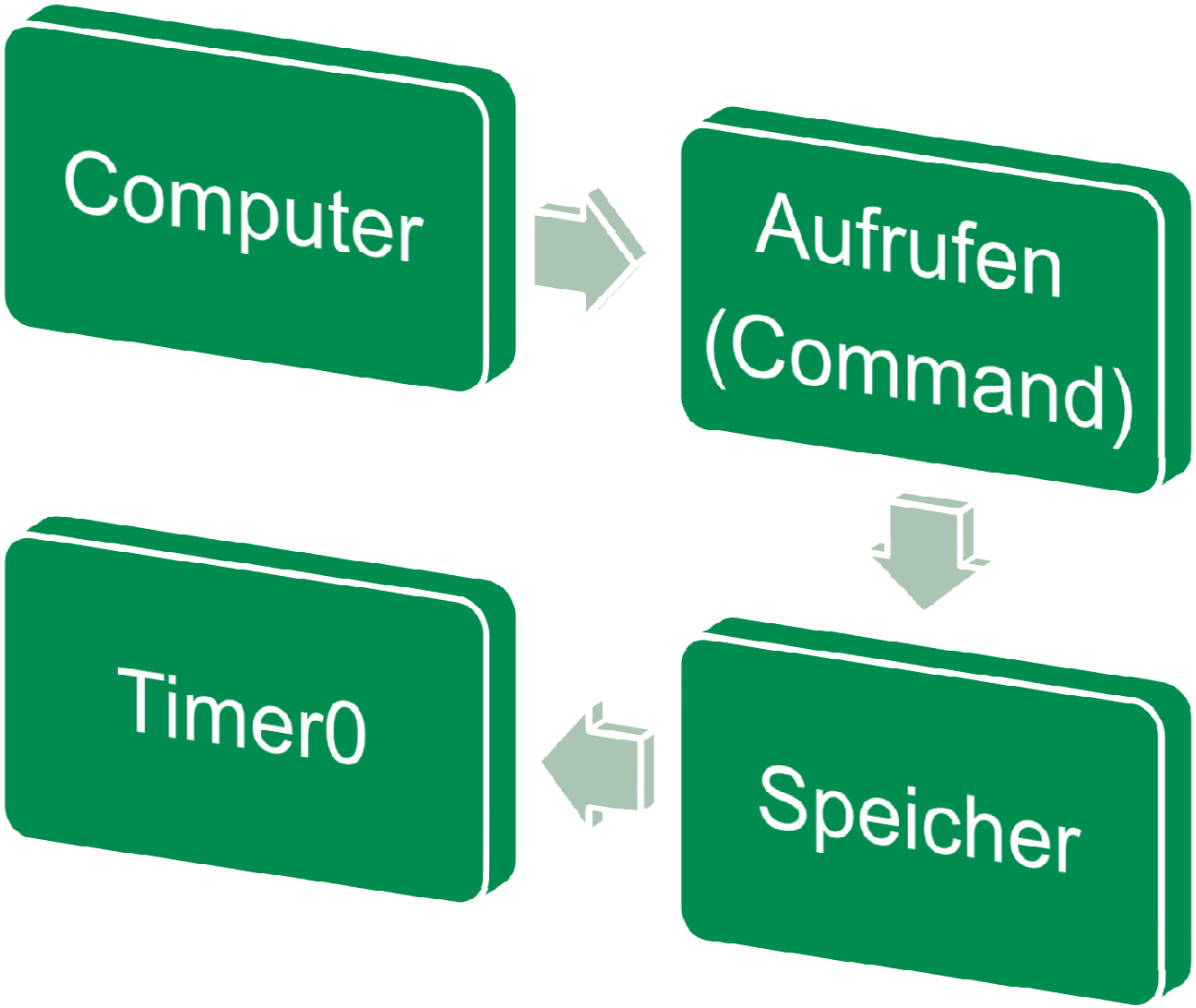
Echtzeit Kontrolle

- **Vorteile:**
  - ✓Echtzeit Kontrolle
  - ✓Echoinformationen
- **Nachteile:**
  - ✓Zeitdauer

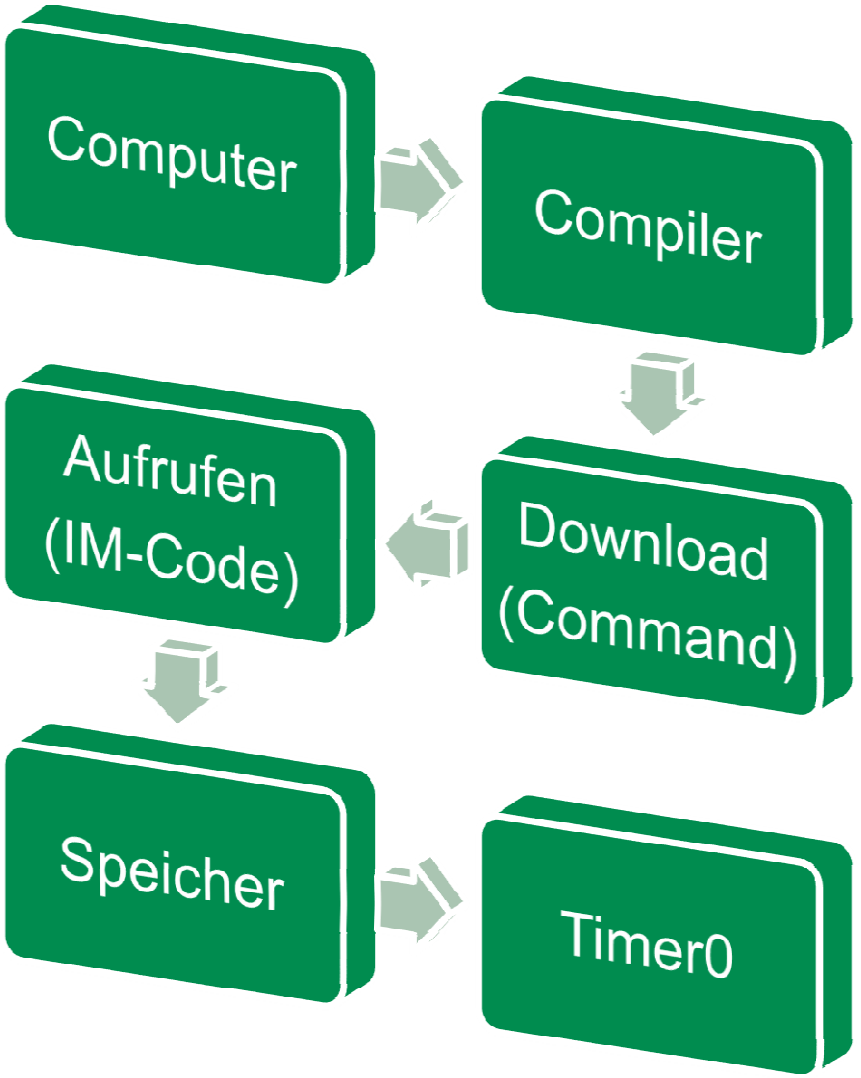
Non Echtzeit Kontrolle

- **Vorteile:**
  - ✓Zeitdauer
- **Nachteile:**
  - ✓Kein Echtzeit Kontrolle
  - ✓Kein Echoinformationen

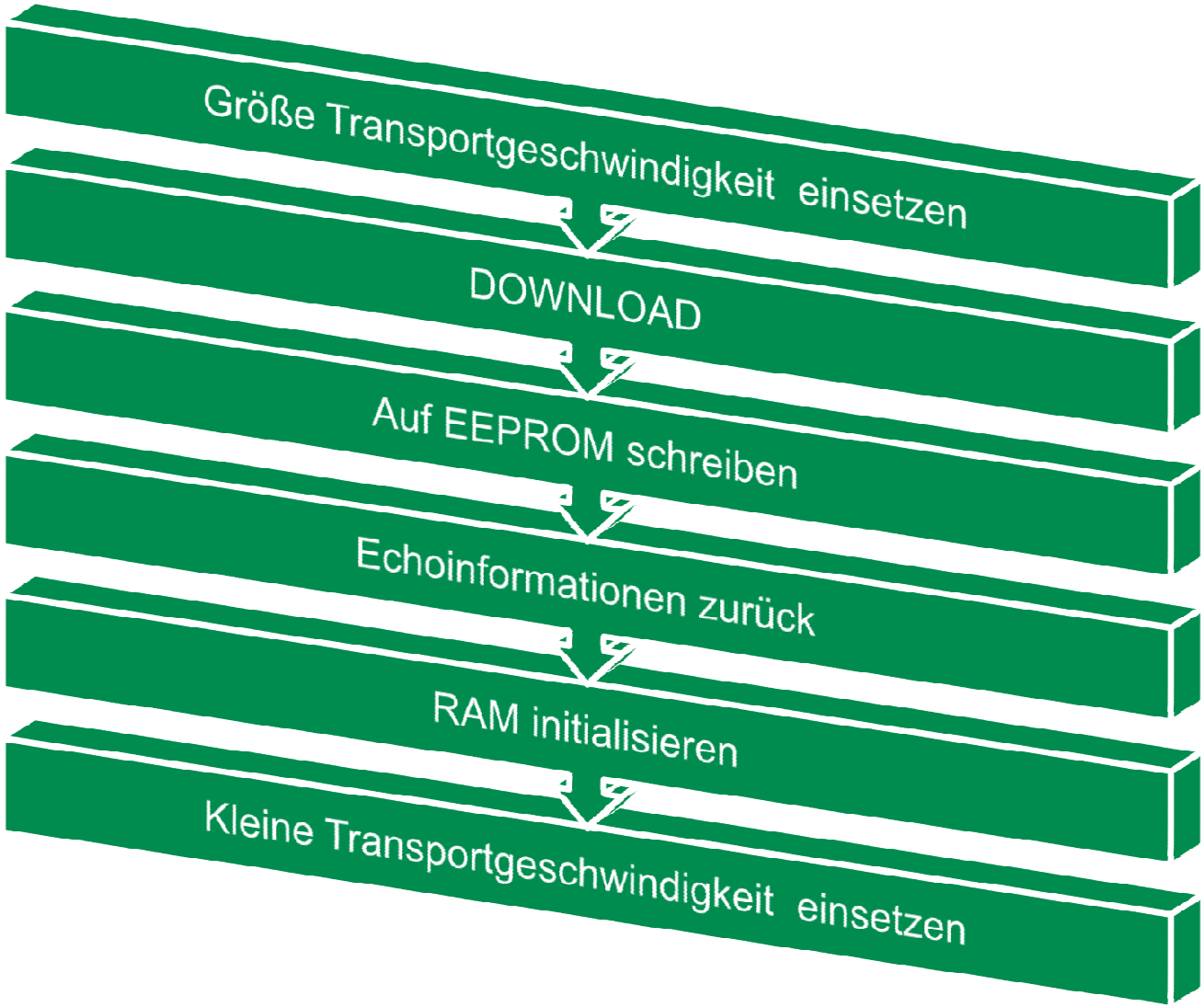
Echtzeit Kontrolle



Non Echtzeit Kontrolle



Download
Avr0522



Download
Avr0522

PC → C3024	C3024 → PC	Erläuterung
0xEC	0xEC	1 Byte (Command Download aufrufen)
Length Low/High	Length Low/High	2 Bytes (Paket Größe)
0x3F	0x21	1 Byte (0x3F, falls weiteres Bytes kommt)
Code	None	128 Bytes (Zwischencode auf EEPROM gespeichert werden)
Checksum	None	1 Byte (Prüfsumme)

TWI
Abkürzung der TWI-Registern

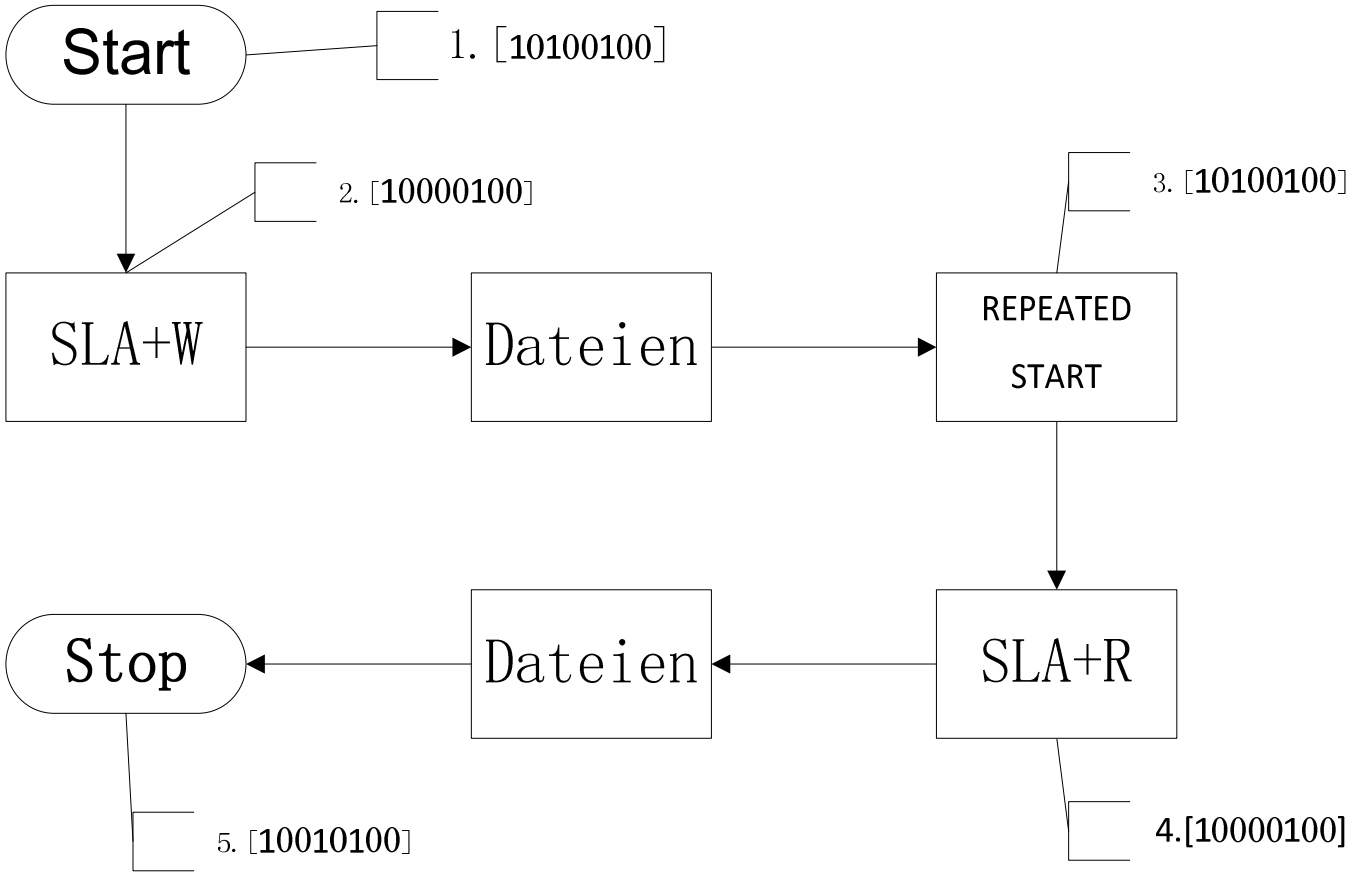
- TWBR : Bauderate Register
- TWCR : Kontroll Register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	--	TWIE

  - TWINT : Interrupt Bit
  - TWEA : Enable Echo Bit
  - TWSTA : START Bit
  - TESTO : STOP Bit
  - TWEN : Enable Bit
  - TWIE : Interrupt Enable Bit
- TWDR : Dateien Register
- TWSR : Status Register

- SLA : Slave Address +W : write +R : read

Aufrufen
EEPROM lesen(Sub07D6)



EEPROM lesen

```
TWCR = (TWINT ←1)&(TWSTA ←1)&(TWEN ←1);   // 10100100 send
  START-Signal
while (!(TWINT ←1));   //wart auf TWINT gesetzt wird.(START gesendet)
if (TWSR != 0x08) {ERROR();}  //überprüfen, ob START empfangen.


TWDR = SLA+W;
TWCR = (TWINT ←1) &(TWEN ←1);  // 10000100
while (!(TWINT ←1));  // SLA+W (SLA+W gesendet)
if (TWSR != 0x20) {ERROR()}; //überprüfen, ob SLA+W empfangen.
TWDR = DATEIEN;
TWCR = (TWINT ←1) &(TWEN ←1);  // 10000100
while (!(TWINT ←1));
if (TWSR != 0x30) {ERROR()};  //überprüfen, ob Dateien empfangen
```

EEPROM lesen

```
TWCR = (TWINT ←1)&(TWSTA ←1)&(TWEN ←1);   // 10100100 send
   REPEATED START-Signal
while (!(TWINT ←1));   //wart auf TWINT gesetzt wird.(Signal gesendet)
if (TWSR != 0x10) {ERROR();}  //überprüfen, ob START empfangen.

TWDR = SLA+R;
TWCR = (TWINT ←1) &(TWEN ←1);  // 10000100
while (!(TWINT ←1));  // SLA+R (SLA+R gesendet)
if (TWSR != 0xA8) {ERROR()}; //überprüfen, ob SLA+R empfangen.
TWDR = DATEIEN;
TWCR = (TWINT ←1) &(TWEN ←1);  // 10000100
while (!(TWINT ←1));
if (TWSR != 0x30) {ERROR()};  //überprüfen, ob Dateien empfangen

TWCR = (TWINT ←1)&(TWEN ←1)&(TWSTO ←1);  //10010100  STOP
```

ROBOBASIC
Befehls-Bedienungsanleitung

- RoboBASIC ist eine spezielle Programmiersprache zur Steuerung von Robotern. RoboBASIC stellt eine Erweiterung der allgemeinen Grundprogrammiersprache mit Befehlen zur Steuerung von Robotern dar.

Beispiel von Robobasic Programm:

```
GETMOTORSET G24, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1,
   1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0
SPEED 5
MOTOR G24

MOVE G6A,100,  76, 145,  93, 100, 100
MOVE G6D,100,  76, 145,  93, 100, 100
MOVE G6B,100,  30,  80, 100, 100, 100
MOVE G6C,100,  30,  80, 100, 100, 100
WAIT
```

Befehlszusammenfassung für roboBASIC

- Befehlszusammenfassung
- Befehle, die der Deklaration/Definition dienen

```
DIM             Variable deklarieren
BYTE            Variable bei der Deklaration als Byte definieren
...
```

- Ablaufsteuerbefehle

```
IF              Beginn einer bedingten Anweisung
GOTO            Teilung des Programmablaufs
FOR             Beginn einer Wiederholungsanweisung
...
```

- Befehle zur Motorsteuerung

```
MOTOR           Einschalten des Ausgangsports vom Stellmotor
MOVE            Steuerung mehrerer Motoren zum selber Zeitpunkt
SPEED           Einstellen der Geschwindigkeit des Stellmotors
...
```

- Parameter, welche die Motorgruppe zuordnen

```
G8A             Servomotoren #0-#7 zu Gruppe A zuordnen
G32             Servomotoren #0-#31 zuordnen
...
```

Befehlszusammenfassung für roboBASIC

- Digitale Signaleingabe und –ausgabebefehle

```
IN              Signal vom Eingangsport lesen
OUT             Signal zum Ausgangsport senden
...
```

- Befehle für den Speicher

```
PEEK            Daten vom Controller-Arbeitsspeicher lesen
POKE            Daten in den Controller-Arbeitsspeicher schreiben
...
```

- Befehle für das LCD

```
LCDINIT         Initialisieren des LCD-Moduls
...
```

- Befehle zur externen Kommunikation

```
ERX             RS-232-Signal durch RX-Port empfangen
...
```

- Sonstige Befehle

```
RND             Zufallszahl generieren
REMARK          Erzeugen eines Eintrages in Textform
```

Befehls Analysierung

- **MOVE**

  eine Gruppe von Servos wird bewegt

  Befehlsstruktur

  ```
  MOVE [spezielle Gruppe], [Winkel von Motor n]
  ```

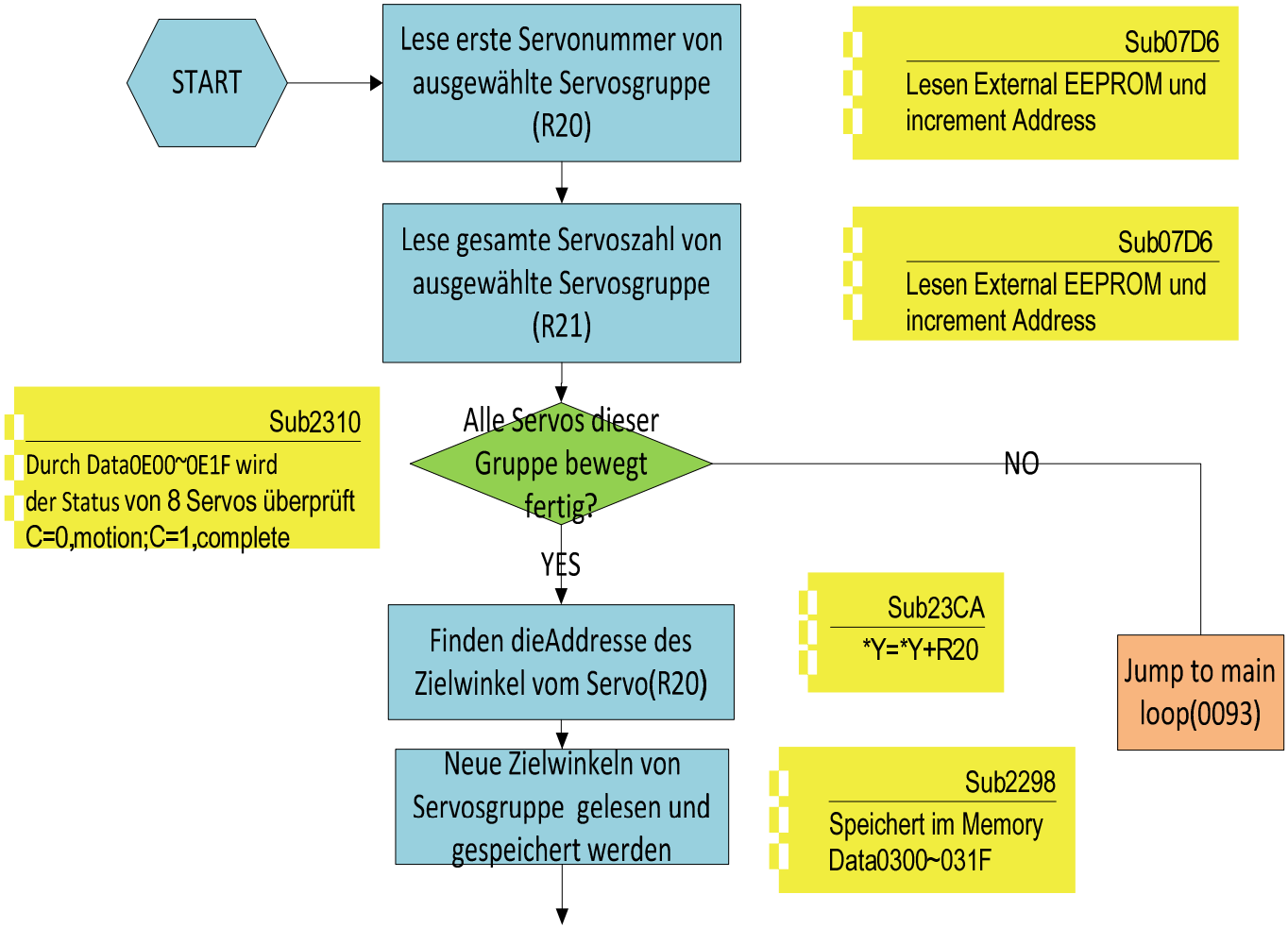
  Beispiel des Befehls

  ```
  MOVE G6A, 85, 113, 72, 117, 115, 100
  MOVE G6C, 75, 45,66,96, 123, 122
  MOVE G8A, 85, 113, 72, 117, 115, 100, 95, 45
  ```

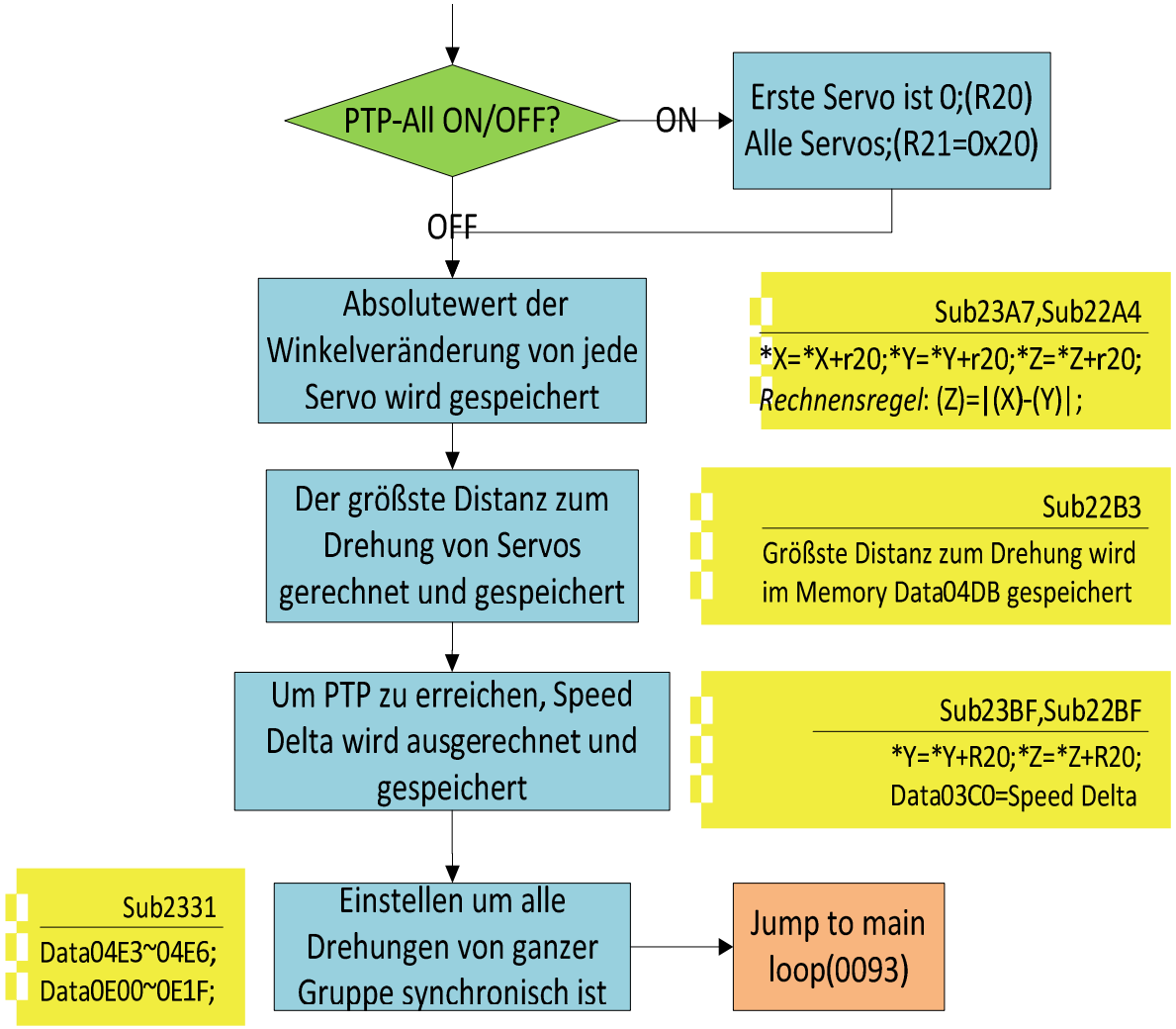
  Kodierung der MOVE-Befehl:

Byte	Inhalt
1	0xB0
2-3	Gruppenbytecode
4-...	Jedes Byte gibt einen Winkel an, den der entsprechende Servo einnehmen soll (10 -190)

Flussdiagramm von MOVE



Flussdiagramm von MOVE



Verlaufsschritt von MOVE-Befehl

- Am Anfang wird *Sub07D6* 2mal angerufen ,dann bekommt 2 Werten und die 2 Werten wird in Register r20 und r21 gespeichert. r20=der erste Servonummer von diese Servosgruppe, r21=insgesamt Servoszahl von diese Servosgruppe.
- Durch Data04E3~04E6 wird die Status von jeder Servo (32 Servos) beobachtet, und speichern die Zustände von jeder Servo mit 0xFF oder 0x00 im Data0E00~0E1F,die Data0E00~0E1F entsprechen 32 Servos,(0xFF bedeutet diese Servo hat fertig bewegt,0x00 bedeutet diese Servo bewegt sich noch).Bit(c) vom SREG wird verändert.

Data Memory	Byte	Inhalt
Data04E3	1	Motor group G8A in motion (0 = moving 1 = complete)
Data04E4	1	Motor group G8B in motion (0 = moving 1 = complete)
Data04E5	1	Motor group G8C in motion (0 = moving 1 = complete)
Data04E6	1	Motor group G8D in motion (0 = moving 1 = complete)
Data0E00~0E1F	32	Irgend einer Motor in motion (0 x00= moving 0xFF= complete)

Verlaufsschritt von MOVE-Befehl

Überprüfen ob die ganze Servosgruppe fertig bewegt mit Bit(c) von Status -Register (SREG):

a) Bit(c)=0,MOVE-Befehl abbrechen;

b) Bit(c) =1,zum nächste schritt gehen(weiter);

- Durch *Sub23CA* wird die Zielwinkel von dem Servo(r20) gefunden,danach durch *Sub2298* wird die alle neue Zielwinkeln von jeden Servo in diese Servosgruppe eingelesen und speichern im entsprechende Addresse (Data0300~ 031F).

  Es ist möglich, einen leeren Parameter anzugeben, falls ein Servo nicht bewegt werden soll. Leerparameter werden im Bytecode mit 0x00 kodiert.

Data Memory	Byte	Inhalt
Data0300~031F	32	Zielwinkel der Servo

- Entscheiden PTP-ALL ON/OFF.

  (PTP-All ON:R20 <- 0x00,R21 <- 0x20;)

Data Memory	Byte	Inhalt
Data04D2	1	Controller Status Byte Bit 7 = PTP ALL 1 = on, 0 = off

PTP-ALL: set all 32 Servos into PTP-Mode.

Verlaufsschritt von MOVE-Befehl

PTP-Mode:Falls dieser Modus aktiviert ist, so wird die Laufzeit aller Bewegungen vom Roboter berechnet. Alle Bewegungen werden dann so ausgeführt, dass sie gleichzeitig enden.

- *Sub23A7,Sub22A4* wird die Absolutewert der Winkelveränderung von jede Servo in diese Servosgruppe ausgerechnet und ins entsprechende Addresse(Data0400) gespeichert.

Data Memory	Byte	Inhalt
Data0300	32	Zielwinkel der servo (10 to 190)
Data0320	32	Aktuelle Winkel der Servo(10 to 190)
Data0400	32	Absolute Winkelveränderung

Funktionsweise von Sub23A7: **X=*X+r20;*Y=*Y+r20;*Z=*Z+r20.*

//*X, *Y,*Z zeigt die Addresse, here *X ist Data0300, *Y ist Data0320,*Z ist Data0400.

Rechnensregel: (Z)=|(X)-(Y)|.

- Im *sub22B3* wird der größte Distanz zum Drehung von Servo gerechnet und in 0x04DB gespeichert.
- Im *sub22BF*,*sub23BF,* um PTP zu erreichen, die Werte von *Speed delta* (bzw. wie groß dreht sich Servo pro Zeiteinheit) gerechnet und gespeichert werden .

Funktionsweise von Sub22BF: *Y=*Y+r20;*Z=*Z+r20.*

//*Y ist Data03C0,*Z ist Data0400.

Verlaufsschritt von MOVE-Befehl

Non-PTP

Alle Servos drehen sich mit gleicher Geschwindigkeit. Je größer absoluter Winkel ist , desto länger braucht es die Zeit.

Der feste Drehungskoeffizient pro Zeiteinheit ist 0x11=17.

*Speedwert*17 = Speed Delta (>=1)*

PTP

Die Drehungen für alle Servos fertigen gleichzeitig. Je größer absoluter Winkel ist , desto mehr dreht sich Servo pro Zeiteinheit. Keine feste Drehungskoeffizient.

(*absolute Winkelveränderung*Speedwert*17) / (größte Distanz) = Speed Delta (>=1) + Rest*

Data Memory	Byte	Inhalt
Data03C0	32	Speed Delta (>=1)
Data04D4	1	Speedwert

- Zum Ende, um alle Drehungen von ganzer Gruppe synchronisch ist, einstellen entsprechende Bits im Data04E3~04E6 von diese Motorsgruppe zu 0.

Beschreibung von GOTO-Befehl

GOTO

Zu einem bestimmten Punkt springen

Befehlsstruktur

```
GOTO [Sprungmarke]
```

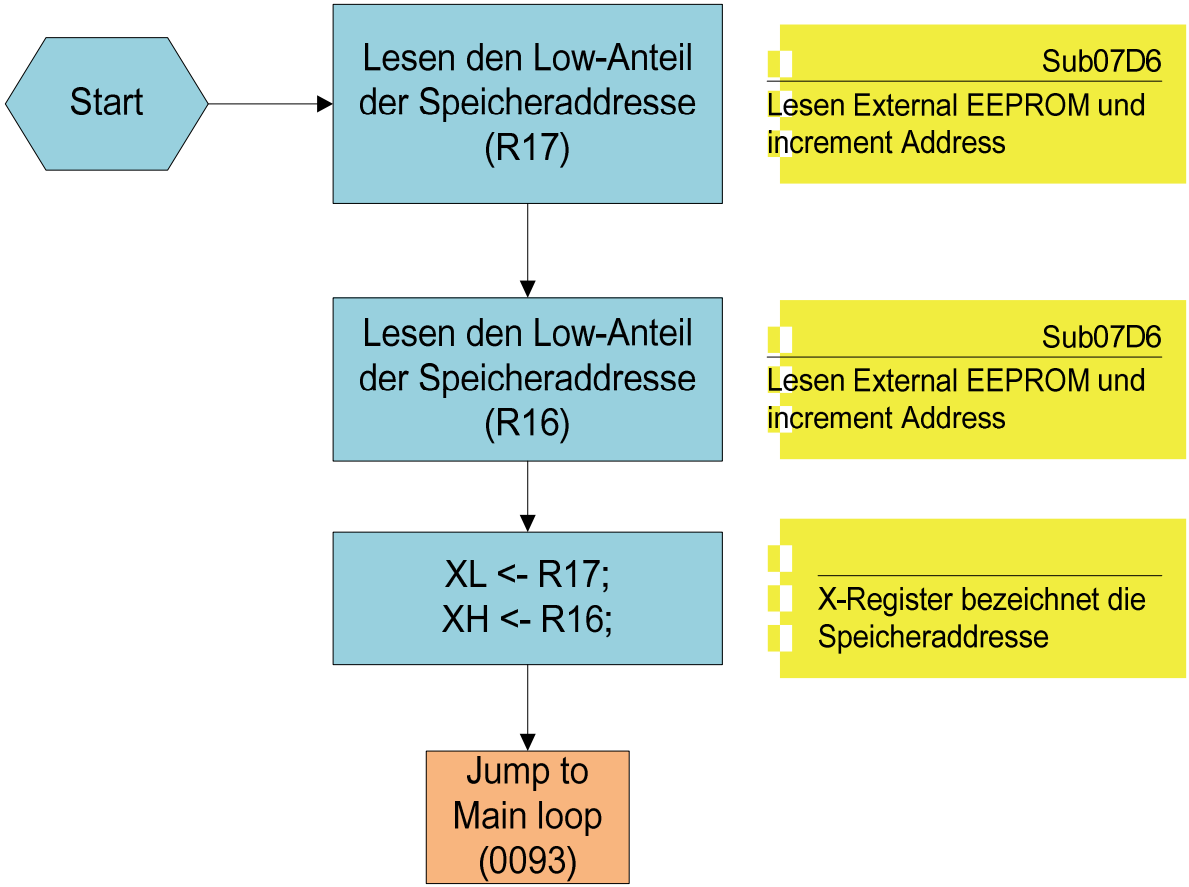
Beispiel des Befehls

```
DIM I AS INTEGER
DIM J AS BYTE
   I = 7
IF I = 6 THEN GOTO L1
   ................
L1: J = 1
```

Kodierung der MOVE-Befehl:

Byte	Inhalt
1	0xC4
2-3	LOW- und HIGH-Anteil der Speicheradresse, an die gesprungen werden soll.

Flussdiagramm von FOR-Befehl



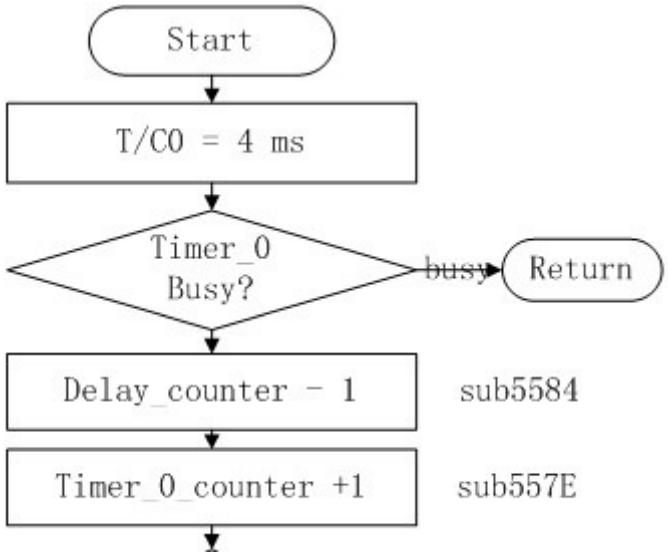
Beschreibung von GOTO-Befehl

- Zweimal sub07D6 angerufen werden 2 Wert(8 bits) vom extern EEPROM ausgelesen und stelle die 2 Werten im Hoch- und Lowbits von X-Register. X-Register ist ein 16 Bits Register. Jetzt bezeichnet der X-Register ein bestimmte Speicheraddresse von einem bestimmt Punkt im Programm-Code,der früher schon definiert.
- Es wird an die Speicheradresse gesprungen, die durch das entsprechende [Sprungmarke] markiert wird. An den Sprung sind keinerlei Bedingungen verknüpft, ebensowenig ist die Richtung des Sprungs entscheidend.

Timer 0

- Kontroller für Robot
- Es realisiert die Bewegung von jedem Servo im Form 1 Grad bis nächste 1 Grad.
- 4 Schritte
  - Frequenz definieren
  - Überprüfungsprozess
  - Winkeldrehungsprozess
  - Null rückstellen

Frequenz definieren



T/C0 = 4ms

Arbeitsfrequenz für Robonova wird gerechnet.

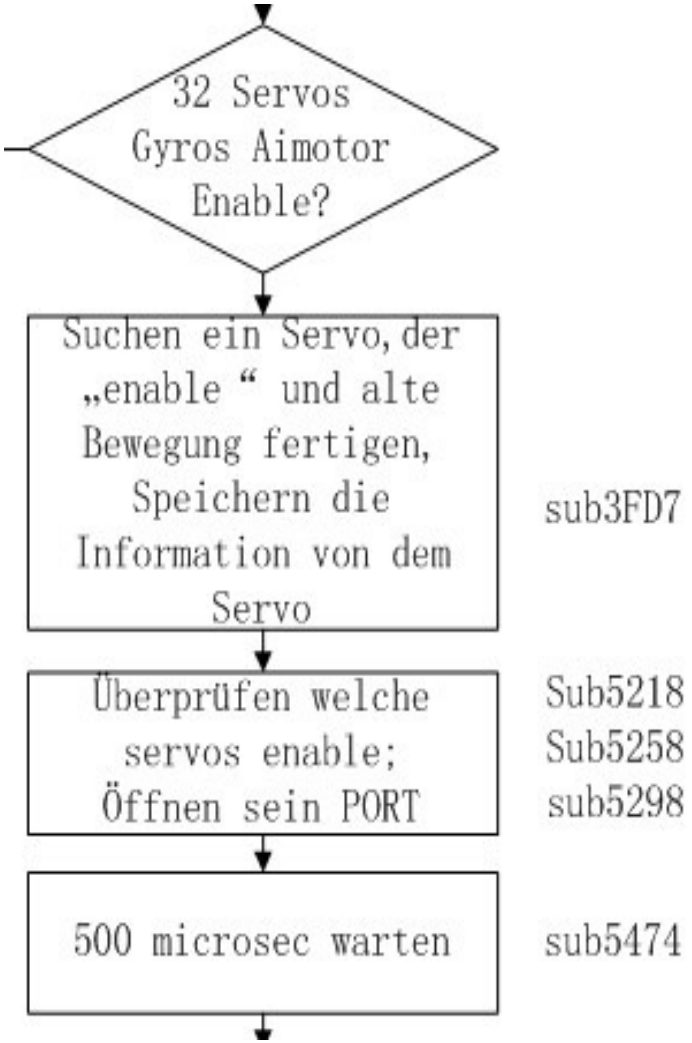
- Beim Anfang von Timer0 wird TCNT0 auf 0x19 definiert. TCNT0 ist ein Timer/Counter in TCCR0(Timer/Counter Control Register). Die Arbeitsweise wird von TCCR0 kontrolliert.
- Beim Programm wird TCCR0 auf 00000101 erstellt.
- Bit 6 und Bit 3 von TCCR0 sind 00 für WGM01:0(Waveform Generation Mode)

  Das bedeutet, dass TCNT0 als normale Weise arbeitet. Und sein maximale Wert ist 0xFF.
- Bit 5 und Bit 4 von TCCR0 sind 00 für COM01:0(Compare Match Output Mode)
- Bit 2 , Bit 1 und Bit 0 von TCCR0 sind 101 für CS02:0(Clock Select)

  Das bedeutet, dass Clock clk_{T0S}/128 gleich 17.36 usec ist.

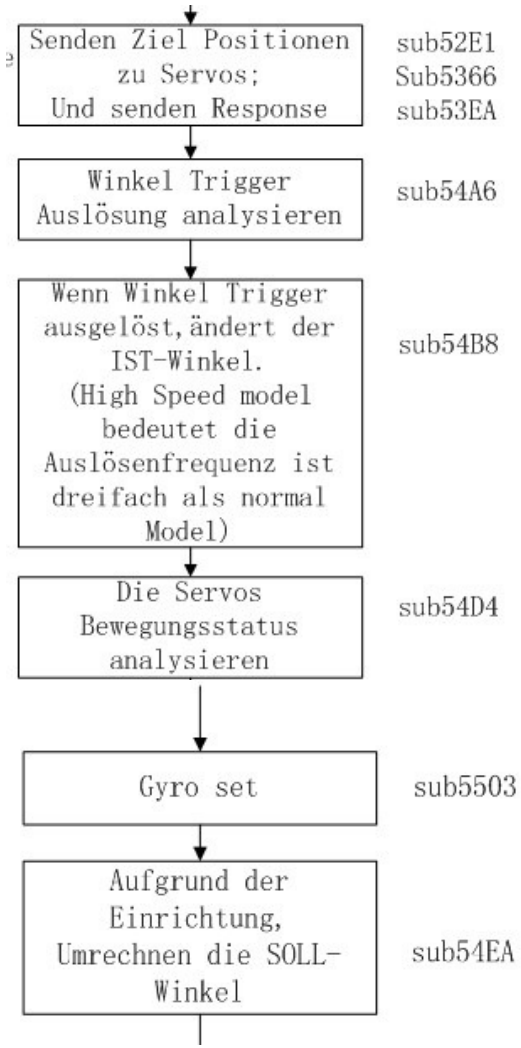
17.36 usec*(0xFF – 0x19) = 0.00001736*(256-25)=4ms

Überprüfungsprozess



Control Bit	PORT	
Data04EB(0)	PORTA(0)	Sub5218 Für 12 Servos
Data04EB(1)	PORTA(1)	
Data04EB(2)	PORTA(2)	
Data04EB(3)	PORTA(3)	
Data04EB(4)	PORTA(4)	
Data04EB(5)	PORTA(5)	
Data04EB(6)	PORTA(6)	
Data04EB(7)	PORTA(7)	
Data04EC(0)	PORTB(0)	
Data04EC(1)	PORTB(1)	
Data04EC(2)	PORTB(2)	
Data04EC(3)	PORTB(3)	
Data04EC(4)	PORTB(4)	Sub5258 Für Nächste 12 Servos
Data04EC(5)	PORTB(5)	
Data04EC(6)	PORTB(6)	
Data04EC(7)	PORTB(7)	
Data04ED(0)	PORTC(7)	
Data04ED(1)	PORTC(6)	
Data04ED(2)	PORTC(5)	
Data04ED(3)	PORTC(4)	
Data04ED(4)	PORTC(3)	
Data04ED(5)	PORTC(2)	
Data04ED(6)	PORTC(1)	
Data04ED(7)	PORTC(0)	
Data04EE(0)	PORTE(7)	Sub5298 Für Letzte 8 Servos
Data04EE(1)	PORTE(6)	
Data04EE(2)	PORTD(7)	
Data04EE(3)	PORTD(6)	
Data04EE(4)	PORTD(5)	
Data04EE(5)	PORTG(2)	
Data04EE(6)	PORTG(1)	
Data04EE(7)	PORTG(0)	

Winkeldrehungsprozess



Zielpositionen zu Servos Senden und Response zurückschicken

- Am Anfang wird die Daten der Pulselänge für Soll Positionen von der vorher schon festgelegte Speicheradresse (*von Data0380 anfang bis folgende 32. Byte, jede Adresse entspricht ein Servo*) ausgelesen.

- Durch 205 Mal Schleife überprüft man, welche Servo seinem Position erreicht hat.(Pulselänge [0,205])

- Dann wird der Port ausgeschaltet, um Servo zu stoppen.

- Wenn der AIMotor aktiviert (Bit 7 in data04F0 = 1) ist und USART0 die Daten empfängen kann (UDREn: USART Data Register Empty = 1), wird eine Rückmeldung zu USART0 schicken.

Winkel Trigger Auslösung analysieren

- Man kann die Geschwindigkeiten von den Servos unter einer Beschränkung beliebig ändern.
- 0x03A0 bis 0x03BF : Ist_Geschwindigkeiten
- 0x03C0 bis 0x03DF : Delta-Geschwindigkeiten
- 0x03E0 bis 0x03FF : Winkelauslösung.

Ist_Geschwindigkeiten + Delta_Geschwindigkeiten

>255 : setzt Winkelauslösung aktiv(=0xFF) ein

sonst =0x00

Änderung von IST-Winkel

Ist_Winkel : 0x0320 bis 0x033F

Soll_Winkel: 0x0300 bis 0x031F

Auslösungen: 0x03E0 bis 0x03FF

- Überprüft zuerst, ob die Auslösungen NULL sind. Wenn Ja, geht direkt zum End, wenn nein, kann man hier mit 2 IF Befehlen die Beziehungen erläutern.

```
If (servo_ausloesung != 0)
            { ist_winkel = *X;
             Soll_winkel = *Y;
            If ( ist_winkel > soll_winkel)
                        { ist_winkel = ist_winkel – 1;
                        *X = ist_winkel;}
            elseif ( ist_winkel < soll_winkel)
                        { ist_winkel = ist_winkel + 1;
                        *X = ist_winkel;}
            }
else        { nächte Servo
 *X = *X + 1;
 *Y = *Y + 1;
 *Z = *Z + 1;
}
```

Die Servos Bewegungsstatus analysieren

Die Status von Servo wird gespeichert.

Bewegung oder Still

Data0320 : Ist_Winkel

Data0300 : Soll_Winkel

Data04E3~04E6 : Status Register von Motor GroupG8A~G8D

- wenn Soll_Winkel = Ist_Winkel

  Servo ist still

  stellt die entsprechende Statusbit von Data04E3~04E6 zu 1 ein;

- wenn Soll_Winkel != Ist_Winkel

  Servo is moving

  macht die entsprechende Statusbit von Data04E3~04E6 nichts;

Aufgrund der Einrichtung, umrechnen die SOLL-Winkel

Die zu Servo gesendete Pulsen werden umgerechnet in dieser Funktion. Und dann speichern die 32

Pulselängen in data0380.

Data0360(32 Byte) : nachgestellte Positionen

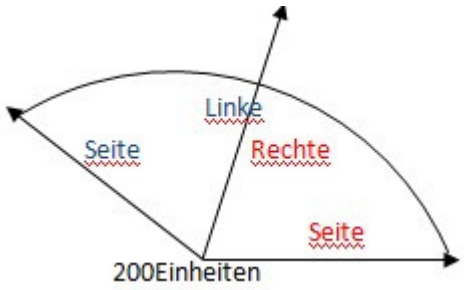
Data0380(32Byte) : die Position ,der nach der Richtung bestimmt wird, und diese neue position werden als

Pulse zu ihre entsprechende Servo gesendet . (‘‘position after direction (pulse) ’’)

- Der maximal Bewegungswinkel vom Motor ist 200 Einheiten. Die Data0360 (nachgestellte Position) von 32 Servos sind alle im Bereich von 0~ 200 Einheit. Verteilen die 200 Einheiten in 2 Bereich, Winkel 0~100 Einheit ist rechte Seite, Winkel 101~200 Einheit ist linke Seite.
- Der ‘‘Servo Direction Bit‘‘ von Data04E7~04EA entscheidet, die Position von diese Servo in welche Seite steht.

  Wenn "Servo Direction Bit" =0, linke Seite, ‘‘position after direction (pulse) ’’=200 – nachgestellte Position;

  Wenn ‘‘Servo Direction Bit "=1, rechte Seite, ‘‘position after direction (pulse) ’’= nachgestellte Position;



“Null“ rückstellen

- Data0320 : Ist positionen
- Data0340 : Zero Position Offset
- Data0360 : nachgestellte Positionen

- Durch zero Position offset von 32 Servos bekommt man die neue nachgestellte Positionen von 32 Servos.

- Ist Position + Zero Position offset = nachgestellt Position

Durch Zero Position Offset werden die nachgestellt Positionen von 32 Servos immer in Wertbereich [1,199]. Die neue nachgestellt Position ist abhängig von der Zahl(Zero Position Offset).

Zusammenfassung

- Alle Funktionen und Befehlssatz vom ganzen Programm werden analysiert.
- Die Start- und Endadresse der Interrupt-Routinen, des Zwischencodeinterpreters und der weiteren Programmbestandteile werden gesucht.
- Dann haben wir auf Flussdiagramm umgeschrieben.
- Das Ziel unseres Arbeit wurde unseres Erachtens mehr als erreicht.

Vielen Dank für Ihre Aufmerksamkeit