4.4 Multi Thread

4.4.1 Datenstrukturen

4.4.1.1 Thread Erzeugen Beenden

4.4.1.2 Stack Bedarf - Liste mit Stackpositionen

4.4.1.3 Liste der wartenden Thread (wait)

4.4.1.4 Liste der lauffähigen Thread (ready)

4.4.1.5 Zeiger auf aktiven Thread (run)

4.4.2 Kontext-Wechsel

4.4.2.1 PC und Register (R0 bis R3 und R4 bis R15)

4.4.2.2 SF Status-Register

4.4.2.3 SF MUL-Register

4.4.3 Synchronisation - Semaphore

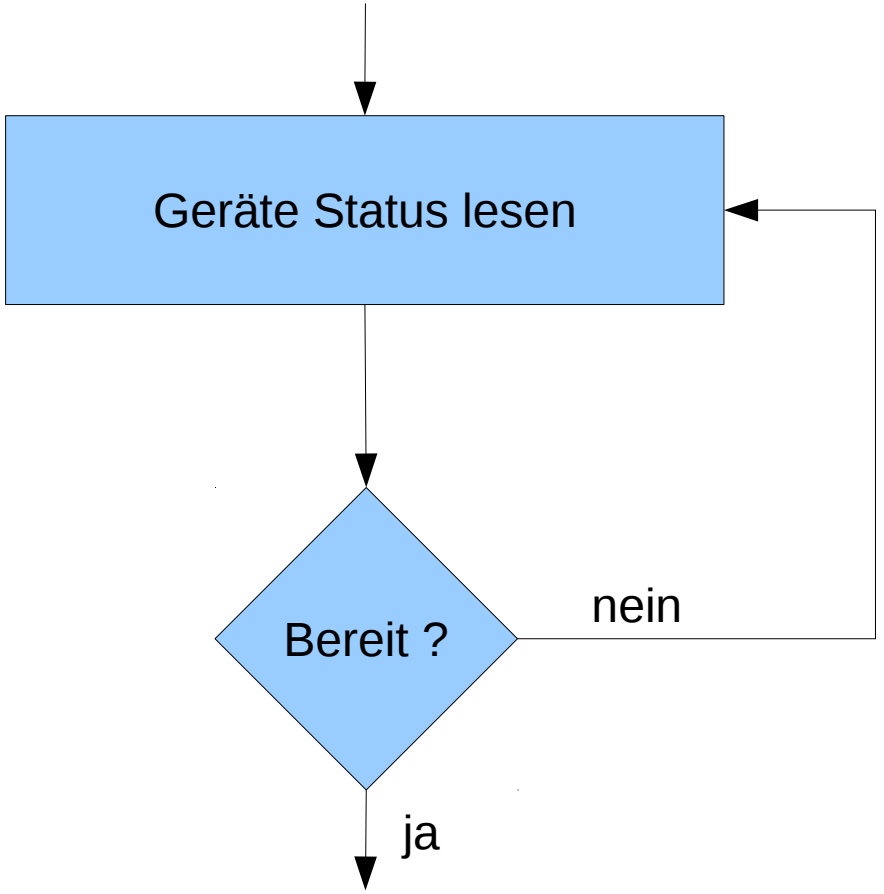
Up - Down Beispiel

4.4.4 Priorität

4.5 Programme mit Speicherschutz

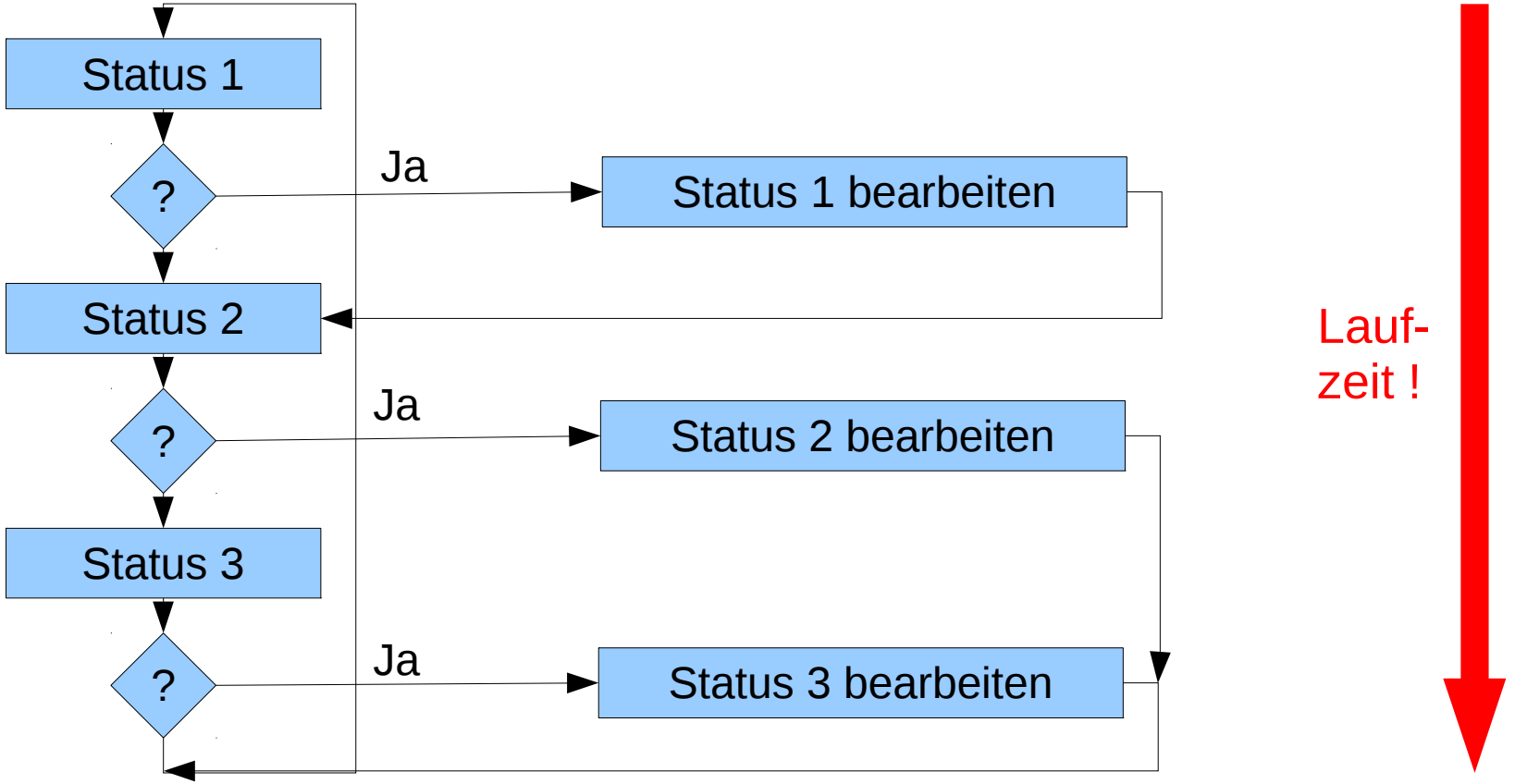
- Warum
- Formen
- Umsetzung

4.1 Polling
4.1.1 Konzept



- Polling: zyklische Abfrage von Status Informationen

4.1.1 Konzept (2)



- Programm Schleife zum Test der Bereitschaft mehrerer Geräte
- Bei Bedarf in die Behandlung des Zustands verzweigen

4.1.1 Konzept (3)

- Status peripherer Geräte wird in Geräteregistern abgefragt
- Zustand von einem oder mehreren Bits muss getestet werden
  - FIFO leer
  - FIFO voll
  - Fehlerzustände

4.1.2 Anwendbarkeit

- Es ist keine andere Möglichkeit zur Abfrage des Status vorhanden (LCD am SpartanMC).
- Die Abfrage soll besonders schnell sein.
- Es muss so wie so auf die Bereitschaft eines Gerätes gewartet werden.
- Polling ist nicht anwendbar, wenn die Laufzeit des längsten Programmpfades bis zum nächsten Polling größer als die geforderte Reaktionszeit ist.

4.1.3 Beispiele
4.1.3.1 Ermittlung der Reaktionszeit

- **Zeit eines Dump** von 0x00000 bis 0x0CFFF soll im Simulator ermittelt werden.
- Einlesen und Starten des **Monitors im Simulator**.
- **Breakpoint am Anfang** (0x0CEB8) **und am Ende** (0x0CECF) des Display Kommandos setzen.
- **Löschen der Statistik am 1. Breakpoint** und Fortsetzung mit RUN.
- **Speichern der Statistik am 2. Breakpoint** mit der Anzahl der Ausgeführten Befehle.
- 51250303 Takte sind bei 25 MHz eine Zeit von 2,05 Sekunden.

4.1.3.2 Polling der UART

```
#include <system/peripherals.h>
#include <uart.h>
#include <stdio.h>
#include <stdio_uart.h>

void main(void) {
// Initialisierung der Geraete
interrupt_disable();
uart_wait_idle(UART_0);
UART_0->ctrl_stat  = (UART_BPS_115200|UART_DATA_LEN_8|UART_TWO_STOP|UART_RX_EN|
                      UART_TX_EN);
stdio_uart_open(UART_0);
UART_1->ctrl_stat  = UART_RX_EN|UART_TX_EN|UART_DATA_LEN_8|UART_BPS_300;
// zyklisches Senden und Empfangen mit Polling der UART
printf("\r\nUART 2 im Polling senden und empfangen\r\n");
```

4.1.3.2 Polling der UART (2)

```
    printf(„ Tx  Rx\r\n“);
    unsigned char b = 0;
    unsigned char c = 0;
    while (1) {
          while (UART_1->status & UART_TX_FULL);
          UART_1->tx_data = b;
           while (UART_1->status & UART_RX_EMPTY);
          c = UART->rx_data;
          printf(„ %02x %02x\r“, b, c);
          b++;
          If (b > 255) b = 0;
       }
    }
 }
```

4.2 Interrupt
4.2.1 Konzept

- Interrupt ist eine kurze asynchrone Unterbrechung eines Programms, um zeitkritische Verarbeitung einzuschieben
- meist für Ein- / Ausgabe Operationen
- Bei gleichzeitigen Interrupts entscheidet eine Prioritäts-reihenfolge, welcher als erster behandelt wird
- Latenz ist >> als beim Polling
- manche Interruptsysteme erlauben auch die Unterbrechung eines laufenden Interrups durch eine Anforderung mit höherer Priorität. ACHTUNG die Behandlungszeiten addieren sich und die Reaktionszeiten sind nicht mehr vorhersehbar!

4.2.2 Anwendbarkeit

- Es sind nur durch die Hardware unterstützte Behandlung asynchroner Unterbrechungen möglich.
- Werden alle notwendigen Aktionen in der ISR bearbeitet, kann die Laufzeit für diesen Interrupt größer als die geforderten Reaktionszeiten der anderen ISR werden.
- Nur die nötigsten Reaktionen in der ISR ausführen, um andere Unterbrechungen nicht zu blockieren.
- ISR wird beim SpartanMC durch eine Assembler Programm in C als Funktion ausgeführt.
- Nur 14 Takte bis zum 1. Befehl der C-Funktion sind 0,28µs
- Nur 3 Takte vom Ende der C-Funktion bis zum Hauptprogramm sind 0,06µs bei 50MHz
- Ein UART Tx Interrupt benötigt insgesammt 39 Takte = 0,78µs

4.2.3.2 Realisierung einer Uhr mit Sekunden Interrupt

```
#include <system/peripherals.h>
#include <interrupt.h>
#include <led7.h>

/*
 * Konstanten für das Programm
 */
#define  LI_TIMER1      ((50000000 / 256)+1)            // limit fuer Timer 1
#define  VT_TIMER1     5
#define  VT_RTI0         3
// Variablen der ISR-Fuktionen
unsigned int  i;              // Wert zur Ausgabe auf die LEDs
```

4.2.3.2 Realisierung einer Uhr mit Sekunden Interrupt (2)

```
void main(void) {
    // Initialisierung der Geraete
    interrupt_disable();
    TIMER_1->control    = TIMER_EN|TIMER_PRE_EN|(TIMER_PRE_VAL*(VT_TIMER1-1));
    TIMER_1->limit      = LI_TIMER1;                    // getesteter Wert fuer eine Sekunde

    RTI_0->ctrl         = RTI_EN|RTI_EN_INT|(RTI_PRE_VAL*VT_RTI0);

    // Initialisierung beendet
    interrupt_enable();

    while (1) {
        }
}
```

4.2.3.2 Realisierung einer Uhr mit Sekunden Interrupt (3)

```
/* Funktionen zur Interrupt Behandlung */
// Sekunden Interrupt von RTI0
void isr00() {
    volatile int  rtictrl;

    rtictrl  = RTI_0->ctrl;          // Interrupt ruecksetzen

    i++;
    if (i>=128) i=0;
    led7_set(i);                     // Wert auf LEDs ausgeben
}
```

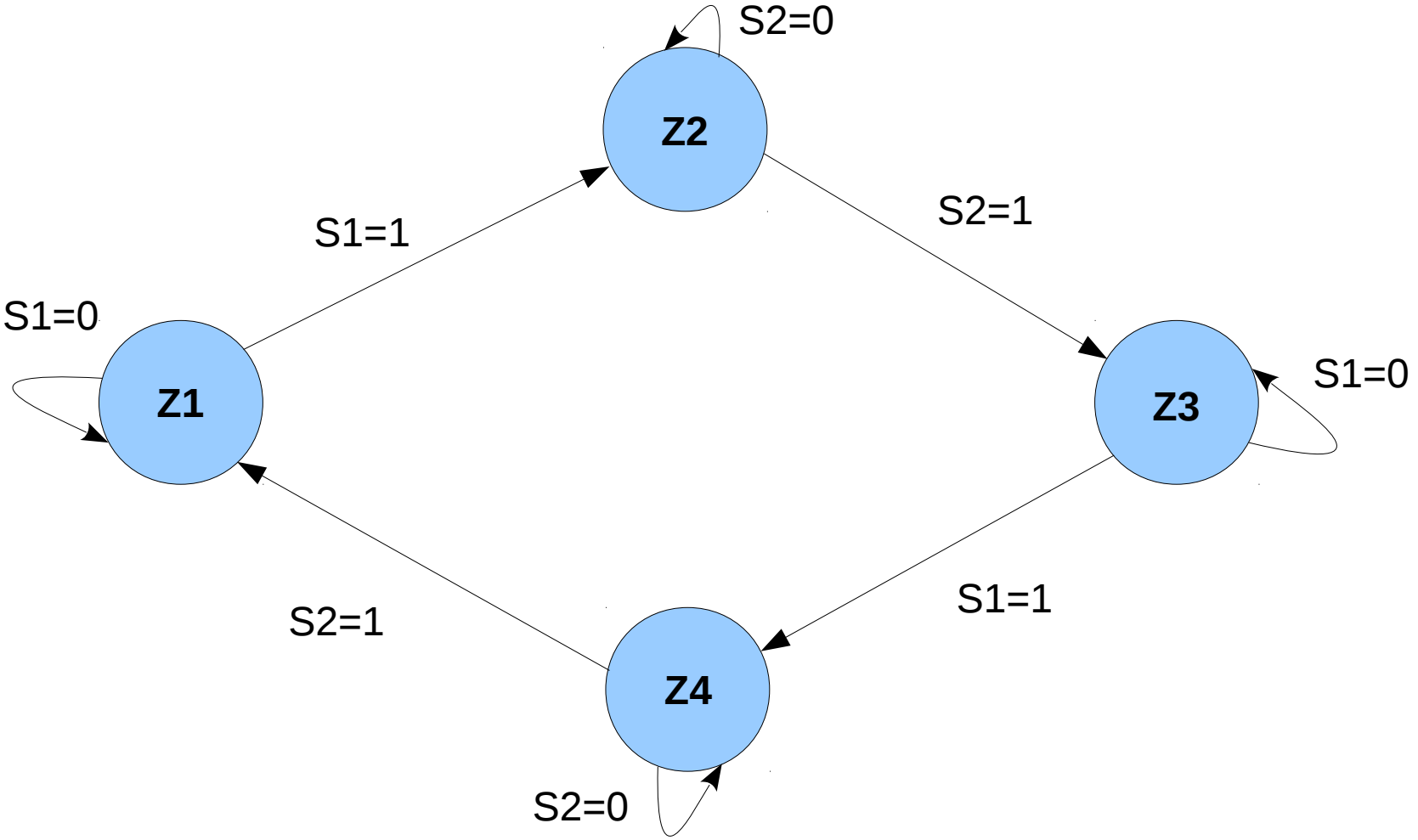
4.2.3.2 Realisierung einer Uhr mit Sekunden Interrupt (4)

Der Sekundentakt wird dabei aus dem Systemtakt mit einem Timer- und einem RTI-Modul gebildet. Der Systemtakt vom 25 MHz wird mit dem Takteingang des Timer-Moduls verbunden und das höchste Bit des 18 Bit Zählers im Timer mit der Takteingang vom RTI-Modul. Damit dieses Bit auch aktiv wird, darf sich der Wertebereich für das „limit-Register“ nur im Bereich von größer 2^{17} bis 2^{18} -1 (131072 < limit < 262143) bewegen. Wird mit den beiden Vorteilern zusammen ein Wert von 2^{7} = 128 eingestellt, dann bleibt für limit ein Wert von (25000000 / 128) = 195312,5 übrig. Dieser Wert liegt genau im zulässigen Wertebereich. Da nur ganzzahlige Werte zulässig sind, wurde mit 195312 und 195313 getestet. Bei der Verwendung von 195313 = int((25000000 /128)+1) läuft die Uhr über mehrere Tage ohne Abweichung. Eingestellt wurde daher:

```
TIMER_1->control    = TI_EN|TI_PRE_EN|(TI_PRE_VAL*(4-1));    // TI_PRE_VAL = 2^4
TIMER_1->limit      = ((25000000 / 128)+1);                  // getesteter Wert für eine Sekunde
RTI_0->control      = RTI_EN|RTI_EN_INT|(RTI_PRE_VAL*3);     // RTI_PRE_VAL = 2^3
```

4.3 State machine
4.3.1 Der Zustands Graph

Zustände eines kleinen Automaten.



4.3.2 CASE in der Realisierung der State machine

```
// Kodierung der Zustände
typedef enum {
        ZUSTAND1 =0,
        ZUSTAND2,
        ZUSTAND3,
        ZUSTAND4
};
// Anfangszustand initialisieren
unsigned int state      = ZUSTAND1;
unsigned int old_state  = ZUSTAND1;
```

4.3.2 CASE in der Realisierung der State machine (2)

```
void stateMachine()
{
    switch (state) {
      case ZUSTAND1:
          led7_set(state);          // Wert auf LEDs ausgeben
          old_state   =     state;
          if(signal1()==1){   state=ZUSTAND2; }
          else{     state = ZUSTAND1; }
          break;
      case ZUSTAND2:
          led7_set(state);          // Wert auf LEDs ausgeben
          old_state   =     state;
          if(signal2()==1){   state=ZUSTAND3; }
          else{     state = ZUSTAND2; }
          break;
```

4.3.2 CASE in der Realisierung der State machine (3)

```
        case ZUSTAND3:
            led7_set(state);            // Wert auf LEDs ausgeben
            old_state    =       state;
            if(signal1()==1){   state=ZUSTAND4; }
            else{      state = ZUSTAND3; }
            break;
        case ZUSTAND4:
            led7_set(state);            // Wert auf LEDs ausgeben
            old_state    =       state;
            if(signal2()==1){   state=ZUSTAND1; }
            else{      state = ZUSTAND4; }
            break;
    }
}
```

4.3.2 CASE in der Realisierung der State machine (4)

```
void main(void) {
        interrupt_disable();
        uart_wait_idle(UART_0);
        UART_0->ctrl_stat  = (UART_BPS_115200|UART_DATA_LEN_8|UART_RX_EN
                              |UART_TX_EN);
        stdio_uart_open(UART_0);
        TIMER_1->control   = TIMER_EN|TIMER_PRE_EN|(TIMER_PRE_VAL*(VT_TIMER1-1));
        TIMER_1->limit     = LI_TIMER1;                    // getesteter Wert für eine Sekunde
        RTI_0->ctrl        = RTI_EN|RTI_EN_INT|(RTI_PRE_VAL*VT_RTI0);
        RTI_1->ctrl        = RTI_EN|RTI_EN_INT|(RTI_PRE_VAL*2);
        printf(„\r\nAutomat mit CASE in C\r\n\n");
        interrupt_enable();
        while (1) {
        }
}
```

4.3.2 CASE in der Realisierung der State machine (4)

```
/* Funktionen zur Interrupt Behandlung */
// Sekunden Interupt von RTI0
void isr_rti0() {
        volatile int   rtictrl;
        rtictrl  = RTI_0->ctrl;              // Interrupt ruecksetzen
        stateMachine();
}
// Interrupt von RTI1 aller 4 Sekunden
void isr_rti1() {
        volatile int   rtictrl;
        rtictrl  = RTI_1->ctrl;              // Interrupt ruecksetzen
        signal_x++;
        if(signal_x >= 4) signal_x = 0;
}
```

4.3.3 Sortierung von Paketen.

Folgende Zustände werden für die Sortierung der 3 Paketsorten benötigt:

- **Stopp** (Warten auf das Auflegen eines Paketes und Überwachen des Drucks.)
- **Fehler** (Zu wenig Druck zur Ansteuerung der Stellzylinder.)
- **Start** (Paket wurde aufgelegt, warten bis L2 unterbrochen wird.)
- **P_Norm** (Normales Paket wurde erkannt ~L1 und ~L3)
- **P_Lang** (Langes Paket wurde erkannt L1 und ~L3)
- **P_Dick** (Dickes Paket wurde erkannt L1 und L3)
- **Warten** (Warten bis das Paket die 2. Sperre passiert hat ca. 5 Sekunden)
- **Neu_P** (Neues Paket, fortsetzen mit Start oder nach max. 25 Sekunden nach Stopp gehen.)

Die Angaben hinter jedem Zustand ist nur eine Auswahl der wichtigsten Aktionen in den Zuständen. Eine vollständige Auflistung aller Aufgaben der einzelnen Zustände und unter welchen Bedingen der Nachfolgezustand eingenommen wird, soll im Folgenden für alle Zustände erläutert werden.

4.3.3 Sortierung von Paketen. (2) **Stopp** Zustand

Der **Stopp** Zustand sollte sofort beim Einschalten mit dem Start des Programms eingenommen werden. In Ihm müssen alle Magnetventile und der Motor für den Antrieb des Förderbandes ausgeschaltet sein. Ist kein ausreichender Luftdruck vorhanden, soll in den **Fehler** Zustand gewechselt werden. Bei ausreichendem Druck und unterbrochener Lichtschranke 1 (ein Paket wurde aufgelegt) muss der Motor eingeschaltet werden und ein Wechsel in den Zustand **Start** erfolgen. Der Zustand wird wieder eingenommen, wenn im Zustand **Neu_P** nach maximal 25 Sekunden kein neues Paket aufgelegt wird. Dann müssen alle Magnetventile und der Motor abgeschaltet werden. Diese Aktionen sollten nur ausgeführt werden, wenn die Variablen „state“ und „old_state“ noch unterschiedlich sind.

4.3.3 Sortierung von Paketen. (3) **Fehler** Zustand

Der **Fehler** Zustand muss immer dann eingenommen werden, wenn kein ausreichender Luftdruck zum Ansteuern der Stellzylinder vorhanden ist. Die Überwachung des Luftdrucks braucht dabei nur in den Zuständen erfolgen, in denen sich der Automat längere Zeit aufhalten kann. Die drei Zustände für die Sortierung **P_Norm**, **P_Lang** und **P_Dick** müssen nicht unbedingt den Druck überwachen. Der Fehlerzustand sollte eine Meldung auf der Konsole, oder der LCD anzeigen. Es kann aber auch eine blinkende LED zum Signalisieren angewendet werden. Ist wieder ausreichender Luftruck vorhanden, soll immer in den **Stopp** Zustand gewechselt werden. Der Fehlerzustand muss auch alle Magnetventile und den Motor ausschalten. Dies kann immer beim Eintritt in den Zustand erfolgen, wenn die Variablen „state“ und „old_state“ noch nicht übereinstimmen.

4.3.3 Sortierung von Paketen. (4) **Start** Zustand

Der **Start** Zustand muss immer dann eingenommen werden, wenn ausreichender Luftdruck zum Ansteuern der Stellzylinder vorhanden ist und die Lichtschranke L1 durch Auflegen eines Paketes unterbrochen wurde. Mit der Unterbrechung der Lichtschranke L2 muss ein Wechsel in **P_Norm**, **P_Lang** oder **P_Dick** erfolgen. Eine Überwachung des Drucks kann weggelassen werden, da der Zustand nur kurz eingenommen wird. Eine Überwachung der Zeit könnte aber sinnvoll sein, wenn zum Beispiel das Band verklemmt ist. Der Wechsel in den Nachfolgezustand ist vom Zustand der Lichtschranken L1 und L3 abhängig.

L1	**L3**	**Folge Zustand**
frei	frei	**P_Norm**
unterbrochen	frei	**P_Lang**
unterbrochen	unterbrochen	**P_Dick**

4.3.3 Sortierung von Paketen. (5) **P_xxxx** Zustand

Die Zustände **P_Norm**, **P_Lang** oder **P_Dick** werden immer als Folgezustand von **Start** eingenommen. Auch diese 3 Zustände werden nur kurz eingenommen und müssen den Druck nicht überwachen. Auf alle 3 Zustände folgt aber immer der Zustand **Warten**. Sollen normale Pakete gleich hinter den Lichtschranken vom Band fallen, lange Pakete an der 2. Schranke und dicke Pakete bis zum Ende durchlaufen, dann müssen folgende Steuersignale aktiviert werden:

Zustand	Z1a	Z1b	Z2a	Z2b
P_Norm	**off**	**on**	**off**	**off**
P_Lang	**on**	**off**	**off**	**on**
P_Dick	**on**	**off**	**on**	**off**

Nach der Ausgabe dieser Steuersignale kann sofort in den Zustand **Warten** gewechselt werde.

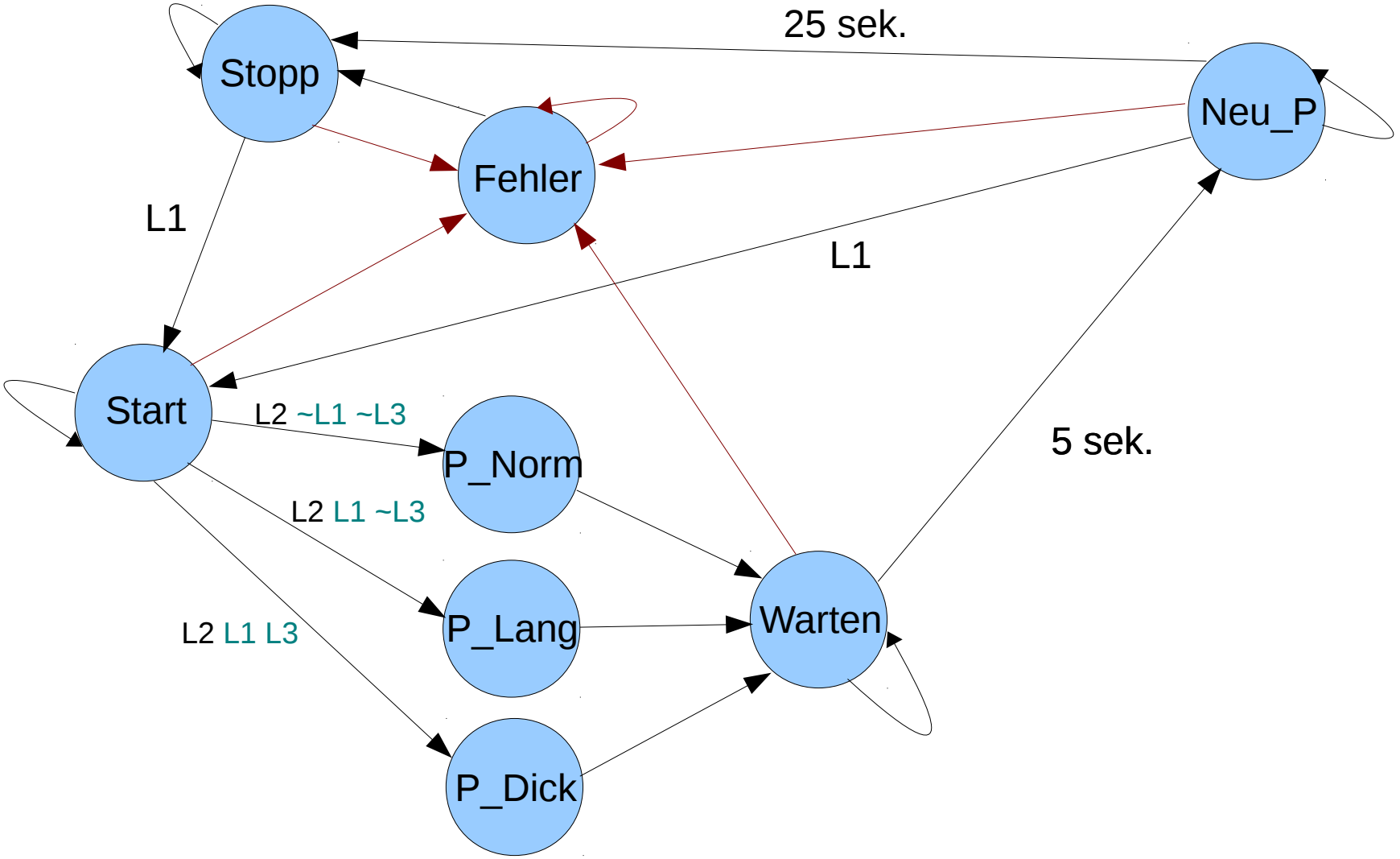
4.3.3 Sortierung von Paketen. (6) **Warte** Zustand

Im Zustand **Warten** muss der Automat solange verharren, wie ein dickes Paket benötigt um bis hinter die 2. Schranke zu laufen. Dies sind etwa **5 Sekunden**. Nach Ablauf dieser Zeit muss immer ein Übergang in den Zustand **Neu_P** erfolgen. Dies ist auch der Zeitpunkt, zu dem ein neues Paket aufgelegt werden darf und alle Magnetventile abgeschaltet werden können. Bei einem Druckverlust muss sofort ein Wechsel in den **Fehler** Zustand erfolgen.

4.3.3 Sortierung von Paketen. (7) **Neu_P** Zustand

Im Zustand **Neu_P** wird auf das Auflegen eines Paketes gewartet. Das kann zum Beispiel mit einer LED signalisiert werden. Der Druck auf den Stellzylindern wird auch nicht mehr benötigt, so dass sie alle ausgeschaltet werden dürfen. Dies sollte man machen, solange die Variable „state“ und „old_state“ noch ungleich sind. Bei einem Druckverlust muss sofort ein Wechsel in den **Fehler** Zustand erfolgen. Wird nach weniger als 25 Sekunden ein neues Paket aufgelegt, dann erfolgt ein Wechsel in den Zustand **Start**. Nach Überschreiten der **25 Sekunden** wird in den Zustand **Stopp** gewechselt und es muss auch noch der Motor für das Förderband abgeschaltet werden.

4.3.3 Sortierung von Paketen. (8) **Z- Graph**



;