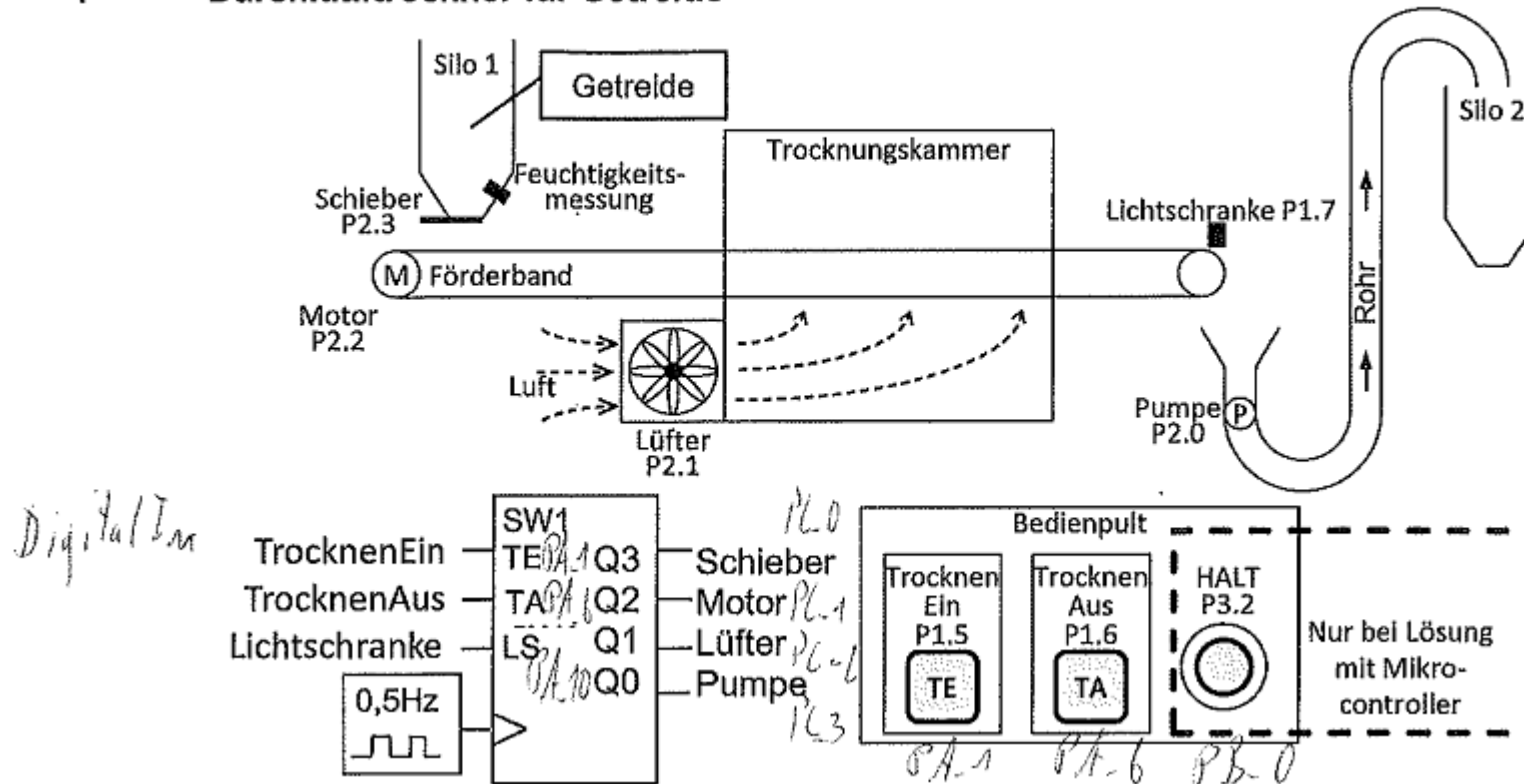


Musterlösung HP1920 A1

Punkte

1 Durchlauftrockner für Getreide



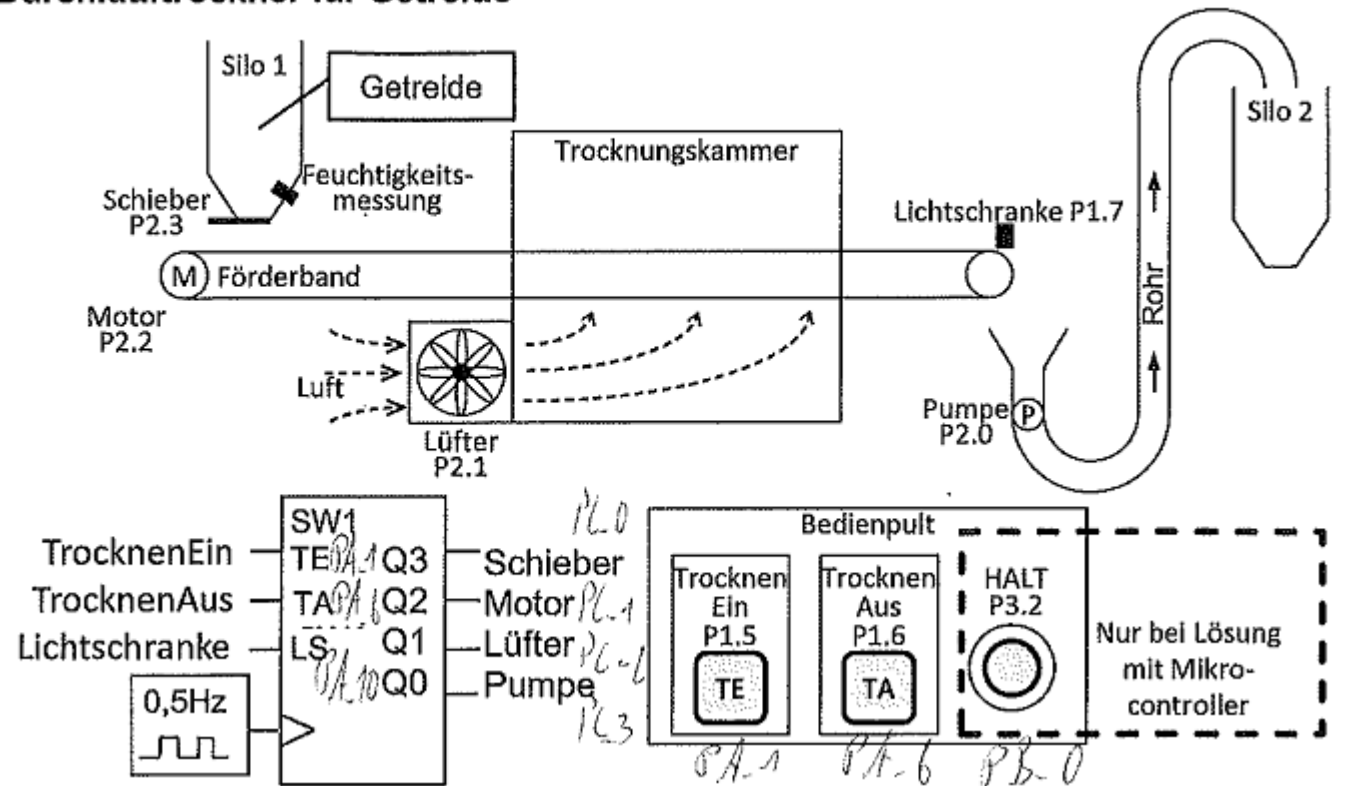
Musterlösung HP1920 A1

Punkte

1

Durchlauftrockner für Getreide

```
DigitalIn TE(PB_4);  
DigitalIn TA(PB_3);  
DigitalIn LS(PB_7);  
InterruptIn HALT(PB_0);  
PortOut ausgaenge(PortC,0x0F);  
DigitalOut Schieber(PC_0);  
DigitalOut Motor(PC_1);  
DigitalOut Luefter(PC_2);  
DigitalOut Pumpe(PC_3);
```



Musterlösung HP1920 A1

1.2.1

Hauptprogramm

Zeichnen Sie einen PAP oder ein Struktogramm für das Hauptprogramm, in dem, nach dem Aufruf eines Unterprogramms `init()`, die Steuerung des Durchlauftrockners entsprechend der Beschreibung (ohne Erweiterung aus 1.1.5), in einer Endlosschleife, erfolgt.

```
3  DigitalIn TE(PB_4);  
    DigitalIn TA(PB_3);  
    DigitalIn LS(PB_7);  
    InterruptIn HALT(PB_0);  
    PortOut ausgaenge(PortC,0x0F);  
    DigitalOut Schieber(PC_0);  
    DigitalOut Motor(PC_1);  
    DigitalOut Luefter(PC_2);  
    DigitalOut Pumpe(PC_3);
```



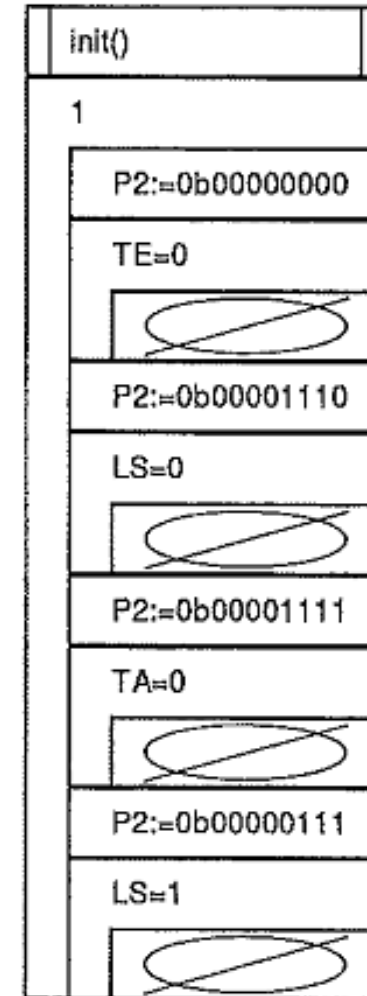
Musterlösung HP1920 A1

1.2.1

Hauptprogramm

Zeichnen Sie einen PAP oder ein Struktogramm für das Hauptprogramm, in dem, nach dem Aufruf eines Unterprogramms `init()`, die Steuerung des Durchlauftrockners entsprechend der Beschreibung (ohne Erweiterung aus 1.1.5), in einer Endlosschleife, erfolgt.

3



Musterlösung HP1920 A1

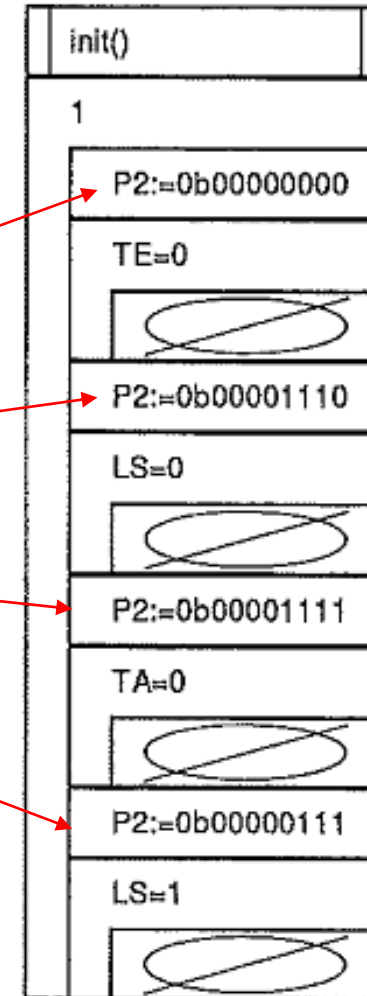
1.2.1

Hauptprogramm

Zeichnen Sie einen PAP oder ein Struktogramm für das Hauptprogramm, in dem, nach dem Aufruf eines Unterprogramms `init()`, die Steuerung des Durchlauftrockners entsprechend der Beschreibung (ohne Erweiterung aus 1.1.5), in einer Endlosschleife, erfolgt.

ausgaenge=

3



Musterlösung HP1920 A1

1.2.2 HALT-Funktion

Zusätzlich zum regulären Ablauf soll es eine **HALT-Funktion** geben, um gegebenenfalls Schmutz oder Fremdkörper aus dem Getreide entfernen zu können. Wenn Schalter „HALT“ auf Low geschaltet wird, müssen Motor, Pumpe und Lüfter angehalten und der Schieber geschlossen werden. Solange **HALT** auf Low gestellt ist, darf der Ablauf nicht gestartet oder fortgesetzt werden können. Nachdem Schalter „HALT“ zurück auf High geschaltet wird, wird der Ablauf an der unterbrochenen Stelle, mit den vorherigen Stellsignalen für Motor, Pumpe, Lüfter und Schieber, fortgesetzt.

Original:

```
C:
void init (void) {
    IT0 = 1; // fallende Flanke
    EX0 = 1; // lokale Interrupt0-Freigabe
    EA = 1; // Globale Freigabe
}
```

Bei uns:

```
void init() {
    HALT.Fall(&HALT_ISR);
    HALT.enable_irq();
    __enable_irq();
}
```



Musterlösung HP1920 A1

1.2.2 HALT-Funktion

Zusätzlich zum regulären Ablauf soll es eine **HALT-Funktion** geben, um gegebenenfalls Schmutz oder Fremdkörper aus dem Getreide entfernen zu können. Wenn Schalter „HALT“ auf Low geschaltet wird, müssen Motor, Pumpe und Lüfter angehalten und der Schieber geschlossen werden. Solange **HALT** auf Low gestellt ist, darf der Ablauf nicht gestartet oder fortgesetzt werden können. Nachdem Schalter „HALT“ zurück auf High geschaltet wird, wird der Ablauf an der unterbrochenen Stelle, mit den vorherigen Stellsignalen für Motor, Pumpe, Lüfter und Schieber, fortgesetzt.

```
C:
void init (void) {
    IT0 = 1; // fallende Flanke
    EX0 = 1; // lokale Interrupt0-Freigabe
    EA = 1; // Globale Freigabe
}
```

```
void init() {
    HALT.Fall(&HALT_ISR);
    HALT.enable_irq();
    __enable_irq();
}
```



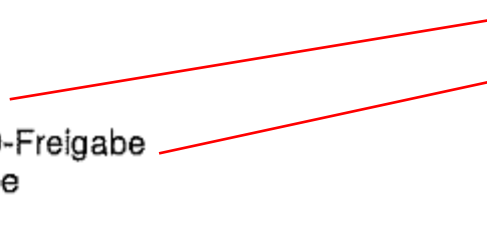
Musterlösung HP1920 A1

1.2.2 HALT-Funktion

Zusätzlich zum regulären Ablauf soll es eine **HALT-Funktion** geben, um gegebenenfalls Schmutz oder Fremdkörper aus dem Getreide entfernen zu können. Wenn Schalter „HALT“ auf Low geschaltet wird, müssen Motor, Pumpe und Lüfter angehalten und der Schieber geschlossen werden. Solange **HALT** auf Low gestellt ist, darf der Ablauf nicht gestartet oder fortgesetzt werden können. Nachdem Schalter „HALT“ zurück auf High geschaltet wird, wird der Ablauf an der unterbrochenen Stelle, mit den vorherigen Stellsignalen für Motor, Pumpe, Lüfter und Schieber, fortgesetzt.

```
C:
void init (void) {
    IT0 = 1; // fallende Flanke
    EX0 = 1; // lokale Interrupt0-Freigabe
    EA = 1; // Globale Freigabe
}

void init() {
    HALT.Fall(&HALT_ISR);
    HALT.enable_irq();
    __enable_irq();
}
```



Musterlösung HP1920 A1

1.2.2 HALT-Funktion

Zusätzlich zum regulären Ablauf soll es eine **HALT-Funktion** geben, um gegebenenfalls Schmutz oder Fremdkörper aus dem Getreide entfernen zu können. Wenn Schalter „HALT“ auf Low geschaltet wird, müssen Motor, Pumpe und Lüfter angehalten und der Schieber geschlossen werden. Solange **HALT** auf Low gestellt ist, darf der Ablauf nicht gestartet oder fortgesetzt werden können. Nachdem Schalter „HALT“ zurück auf High geschaltet wird, wird der Ablauf an der unterbrochenen Stelle, mit den vorherigen Stellsignalen für Motor, Pumpe, Lüfter und Schieber, fortgesetzt.

```
C:
void init (void) {
    IT0 = 1; // fallende Flanke
    EX0 = 1; // lokale Interrupt0-Freigabe
    EA = 1; // Globale Freigabe
}

void init() {
    HALT.Fall(&HALT_ISR);
    HALT.enable_irq();
    __enable_irq();
}
```

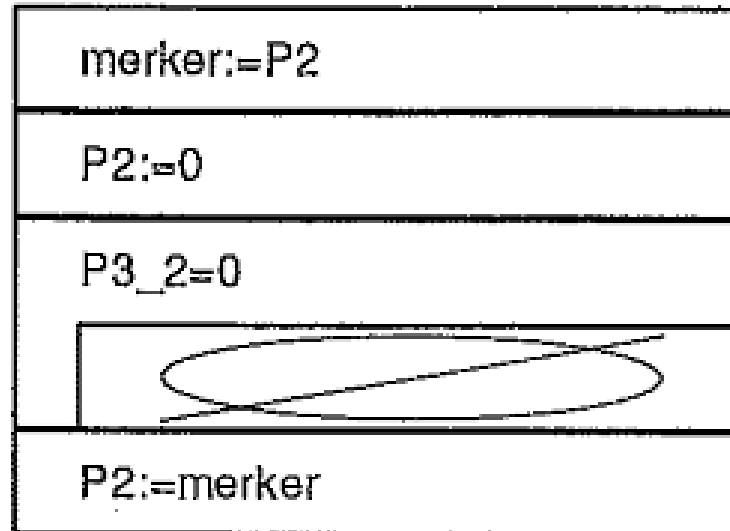


Musterlösung HP1920 A1

- 1.2.2.2 Entwickeln Sie den Algorithmus der Interrupt-Service-Routine und stellen Sie ihn als Programmablaufplan (PAP) oder als Struktogramm dar.

Halt_ISR

lokale Variable: unsigned char merker

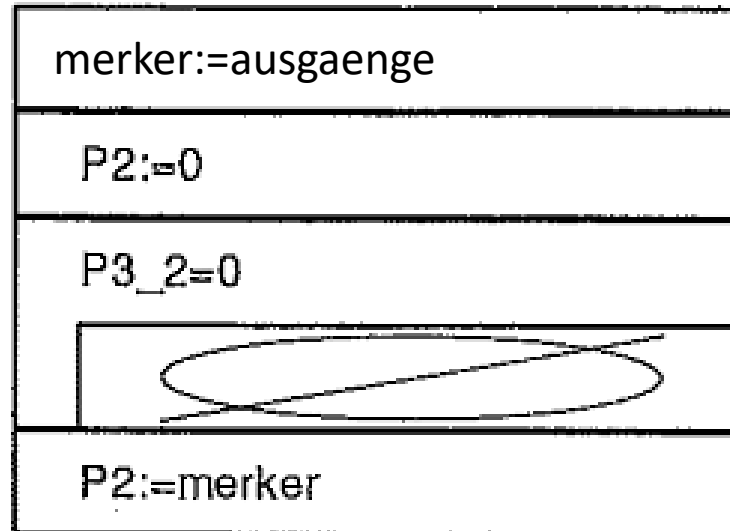


Musterlösung HP1920 A1

- 1.2.2.2 Entwickeln Sie den Algorithmus der Interrupt-Service-Routine und stellen Sie ihn als Programmablaufplan (PAP) oder als Struktogramm dar.

Halt_ISR

lokale Variable: unsigned char merker

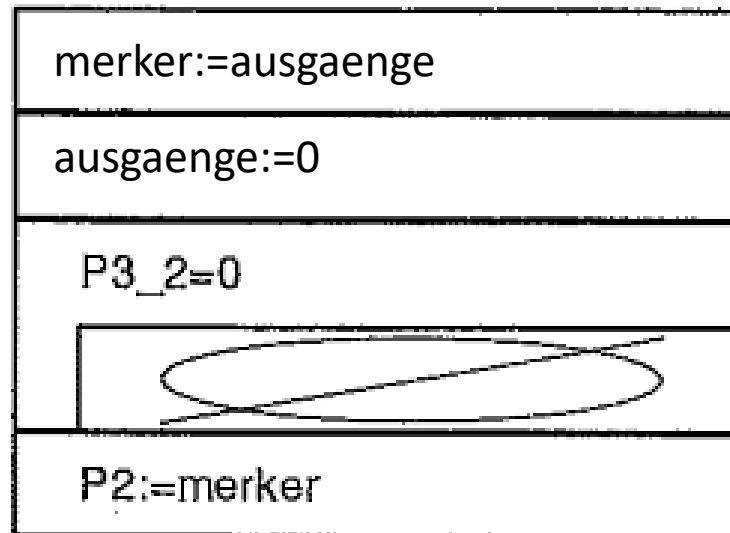


Musterlösung HP1920 A1

- 1.2.2.2 Entwickeln Sie den Algorithmus der Interrupt-Service-Routine und stellen Sie ihn als Programmablaufplan (PAP) oder als Struktogramm dar.

Halt_ISR

lokale Variable: unsigned char merker

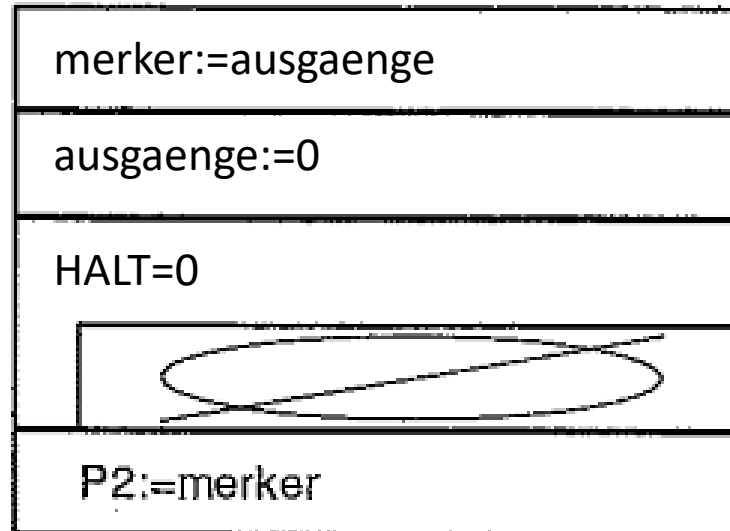


Musterlösung HP1920 A1

- 1.2.2.2 Entwickeln Sie den Algorithmus der Interrupt-Service-Routine und stellen Sie ihn als Programmablaufplan (PAP) oder als Struktogramm dar.

Halt_ISR

lokale Variable: unsigned char merker

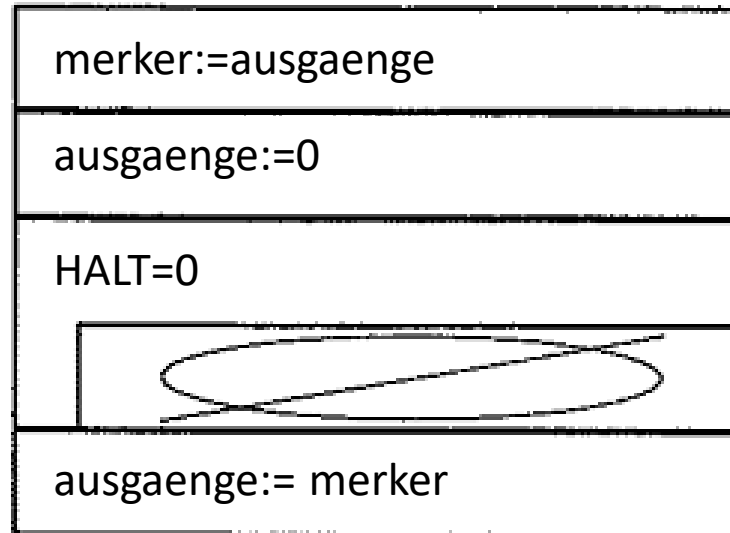


Musterlösung HP1920 A1

- 1.2.2.2 Entwickeln Sie den Algorithmus der Interrupt-Service-Routine und stellen Sie ihn als Programmablaufplan (PAP) oder als Struktogramm dar.

Halt_ISR

lokale Variable: unsigned char merker

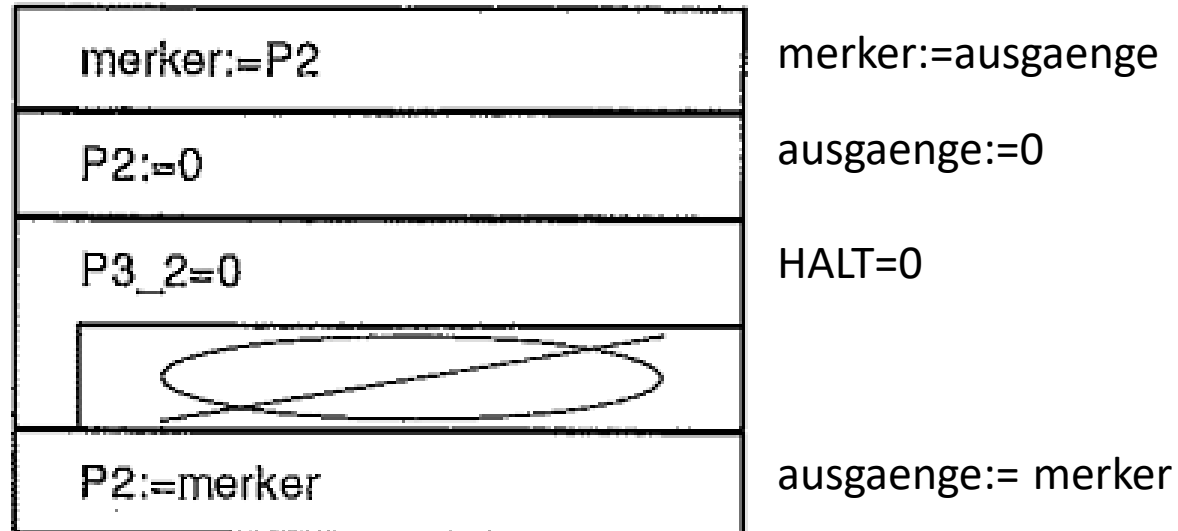


Musterlösung HP1920 A1

- 1.2.2.2 Entwickeln Sie den Algorithmus der Interrupt-Service-Routine und stellen Sie ihn als Programmablaufplan (PAP) oder als Struktogramm dar.

Halt_ISR

lokale Variable: unsigned char merker

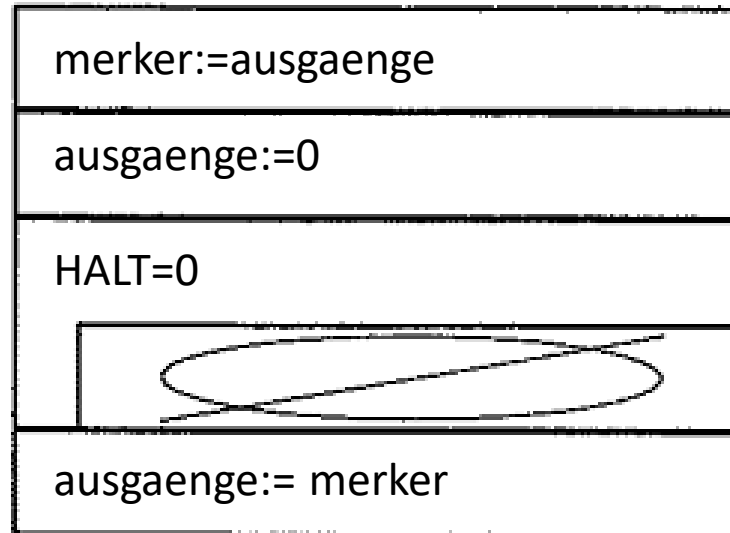


Musterlösung HP1920 A1

1.2.2.2 Entwickeln Sie den Algorithmus der Interrupt-Service-Routine und stellen Sie ihn als Programmablaufplan (PAP) oder als Struktogramm dar.

Halt_ISR

lokale Variable: unsigned char merker

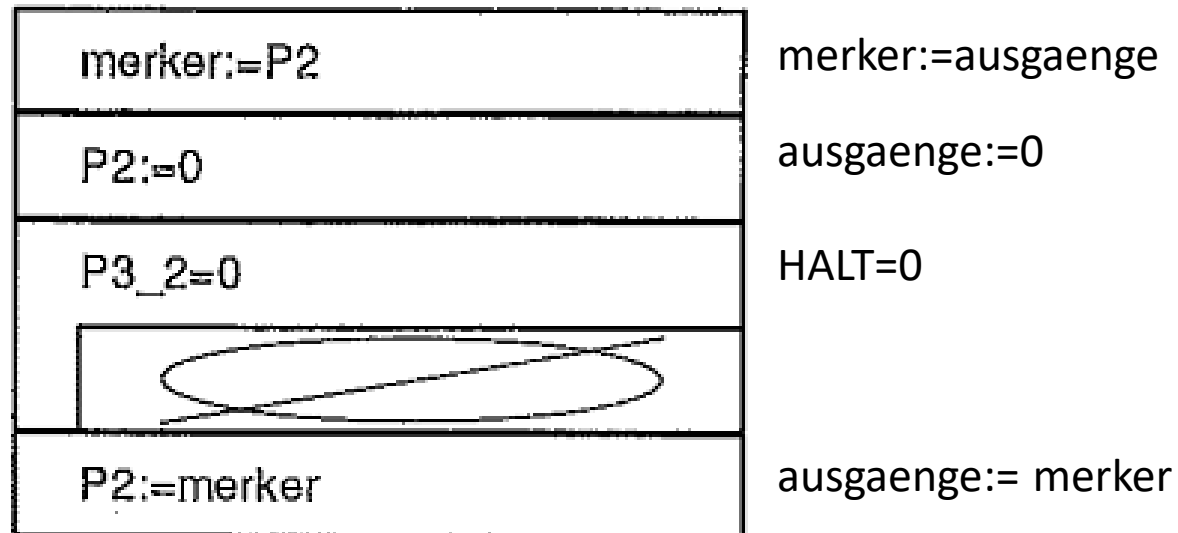


Musterlösung HP1920 A1

1.2.2.2 Entwickeln Sie den Algorithmus der Interrupt-Service-Routine und stellen Sie ihn als Programmablaufplan (PAP) oder als Struktogramm dar.

Halt_ISR

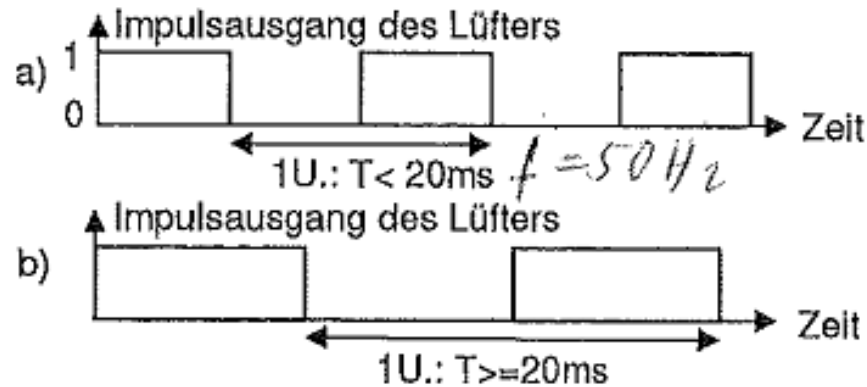
lokale Variable: unsigned char merker



Musterlösung HP1920 A1

1.2.3 Lüfterüberwachung

Für die optimale Trocknung des Getreides, ist es erforderlich, dass der Lüfter mit einer Minstdrehzahl läuft. Zur Drehzahlmessung verfügt der Lüfter über einen Impulsausgang der bei jeder Umdrehung einen Rechteckimpuls ausgibt.



In a) dauert eine Umdrehung des Lüfterrads weniger als 20ms => Drehzahl ok, Warnlampe ausschalten ($P2.4=0$).

In b) dauert eine Umdrehung des Lüfterrads länger als 20ms => Drehzahl zu niedrig, Warnlampe einschalten ($P2.4=1$).

Abbildung 3: Lüfterimpulse

Die Warnlampe wird als „Aus“ initialisiert ($P2.4=0$).

Hinweis: Der Fall, dass der Lüfter schlagartig keine Impulse mehr liefert (z.B. blockiert oder Kabelbruch), muss nicht berücksichtigt werden.

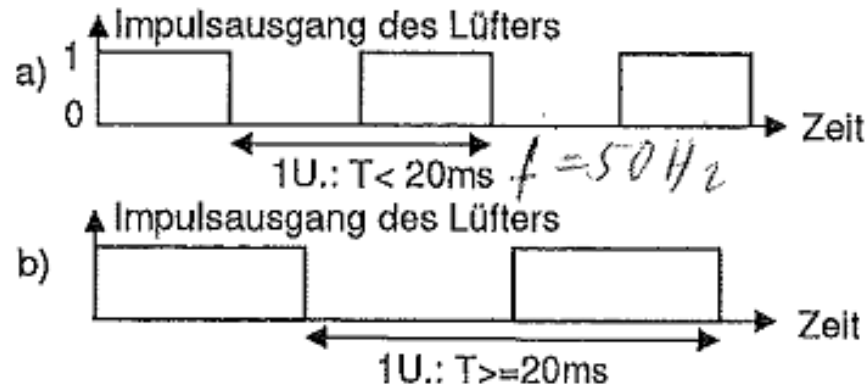
- 1.2.3.1 Bei jeder Umdrehung soll der Lüfter einen externen Interrupt auslösen. Nennen Sie einen geeigneten Port bei Ihrem Mikrocontroller für den Anschluss des Impulsausgangs des Lüfters.



Musterlösung HP1920 A1

1.2.3 Lüfterüberwachung

Für die optimale Trocknung des Getreides, ist es erforderlich, dass der Lüfter mit einer Minstdrehzahl läuft. Zur Drehzahlmessung verfügt der Lüfter über einen Impulsausgang der bei jeder Umdrehung einen Rechteckimpuls ausgibt.



In a) dauert eine Umdrehung des Lüfterrads weniger als 20ms => Drehzahl ok, Warnlampe ausschalten (**P2.4=0**).

In b) dauert eine Umdrehung des Lüfterrads länger als 20ms => Drehzahl zu niedrig, Warnlampe einschalten (**P2.4=1**).

Abbildung 3: Lüfterimpulse

Die Warnlampe wird als „Aus“ initialisiert (**P2.4=0**).

Hinweis: Der Fall, dass der Lüfter schlagartig keine Impulse mehr liefert (z.B. blockiert oder Kabelbruch), muss nicht berücksichtigt werden.

- 1.2.3.1 Bei jeder Umdrehung soll der Lüfter einen externen Interrupt auslösen. Nennen Sie einen geeigneten Port bei Ihrem Mikrocontroller für den Anschluss des Impulsausgangs des Lüfters.

z.B.

`InterruptIn impuls(PA_1);`



Musterlösung HP1920 A1

1.2.3.2 Entwickeln Sie das Programm für die Drehzahlüberwachung unter Verwendung von externem Interrupt und Timer. Gefordert sind die Initialisierung und die Interrupt-Service-Routine(n).

Lösungsidee: Lüfterfrequenz messen.

$T=20\text{ms} \Leftrightarrow f=1/T = 50\text{Hz}$

Lüfterimpuls bewirkt externen Interrupt => ISR zählt 'globale Variable z hoch

Timer mit zyklischen Interrupts alle 1s

TimerISR: falls $z > 50$ warnlampe aus, sonst warnlampe an

z auf 0 zurücksetzen

Pending und Überlauf zurücksetzen



Musterlösung HP1920 A1

1.2.3.2 Entwickeln Sie das Programm für die Drehzahlüberwachung unter Verwendung von externem Interrupt und Timer. Gefordert sind die Initialisierung und die Interrupt-Service-Routine(n).

```
DigitalOut warnlampe(PC_7);  
InterruptIn impuls(PA_1);  
volatile int z=0;
```



Musterlösung HP1920 A1

1.2.3.2 Entwickeln Sie das Programm für die Drehzahlüberwachung unter Verwendung von externem Interrupt und Timer. Gefordert sind die Initialisierung und die Interrupt-Service-Routine(n).

```
void impulsISR()  
{  
    z++;  
    //test=1;  
};
```



Musterlösung HP1920 A1

1.2.3.2 Entwickeln Sie das Programm für die Drehzahlüberwachung unter Verwendung von externem Interrupt und Timer. Gefordert sind die Initialisierung und die Interrupt-Service-Routine(n).

```
void isrTIM6(){  
    test=!test;  
    if (z<50) { warnlampe=1; }  
    else { warnlampe=0; }  
    z=0;  
    TIM6->SR=0;    //Überlaufflag zurücksetzen  
    HAL_NVIC_ClearPendingIRQ(TIM6_IRQn);    //Pendingbit NVIC zurücksetzen  
}
```



```
void init() {  
    ...  
    //Lüfterüberwachung  
    impuls.fall(&impulsISR);  
    impuls.enable_irq();  
    //Timer TIM6  
    RCC->APB1ENR |= 0b10000; //TIM6 mit Takt versorgen  
    TIM6->PSC=31999; //Prescaler für 1ms  
    TIM6->CNT=0;    //Zähler bei 0 starten  
    TIM6->ARR=999;  //Autoreload für 1s (0.999 = 1000 Zählakte  
    TIM6->SR=0;    //Überlaufflag zurücksetzen  
    TIM6->DIER=1;   //Interrupt freigeben Timer  
    NVIC_SetVector(TIM6_IRQn, (uint32_t)&isrTIM6); //ISR zuordnen  
    HAL_NVIC_EnableIRQ(TIM6_IRQn); //Interrupt freigeben NVIC  
    TIM6->CR1=1;   //Timer starten  
}
```



```
void init() {  
    ...  
    //Lüfterüberwachung  
    impuls.fall(&impulsISR);  
    impuls.enable_irq();  
    //Timer TIM6  
    RCC->APB1ENR |= 0b10000; //TIM6 mit Takt versorgen  
    TIM6->PSC=31999; //Prescaler für 1ms  
    TIM6->CNT=0;    //Zähler bei 0 starten  
    TIM6->ARR=999;  //Autoreload für 1s (0.999 = 1000 Zählakte  
    TIM6->SR=0;    //Überlaufflag zurücksetzen  
    TIM6->DIER=1;   //Interrupt freigeben Timer  
    NVIC_SetVector(TIM6_IRQn, (uint32_t)&isrTIM6); //ISR zuordnen  
    HAL_NVIC_EnableIRQ(TIM6_IRQn); //Interrupt freigeben NVIC  
    TIM6->CR1=1;   //Timer starten  
}
```



```
void init() {  
    ...  
    //Lüfterüberwachung  
    impuls.fall(&impulsISR);  
    impuls.enable_irq();  
    //Timer TIM6  
    RCC->APB1ENR |= 0b10000; //TIM6 mit Takt versorgen  
    TIM6->PSC=31999; //Prescaler für 1ms  
    TIM6->CNT=0;    //Zähler bei 0 starten  
    TIM6->ARR=999;  //Autoreload für 1s (0.999 = 1000 Zählakte  
    TIM6->SR=0;    //Überlaufflag zurücksetzen  
    TIM6->DIER=1;   //Interrupt freigeben Timer  
    NVIC_SetVector(TIM6_IRQn, (uint32_t)&isrTIM6); //ISR zuordnen  
    HAL_NVIC_EnableIRQ(TIM6_IRQn); //Interrupt freigeben NVIC  
    TIM6->CR1=1;   //Timer starten  
}
```



```
void init() {  
    ...  
    //Lüfterüberwachung  
    impuls.fall(&impulsISR);  
    impuls.enable_irq();  
    //Timer TIM6  
    RCC->APB1ENR |= 0b10000; //TIM6 mit Takt versorgen  
    TIM6->PSC=31999; //Prescaler für 1ms  
    TIM6->CNT=0;    //Zähler bei 0 starten  
    TIM6->ARR=999;  //Autoreload für 1s (0.999 = 1000 Zählakte  
    TIM6->SR=0;    //Überlaufflag zurücksetzen  
    TIM6->DIER=1;   //Interrupt freigeben Timer  
    NVIC_SetVector(TIM6_IRQn, (uint32_t)&isrTIM6); //ISR zuordnen  
    HAL_NVIC_EnableIRQ(TIM6_IRQn); //Interrupt freigeben NVIC  
    TIM6->CR1=1;   //Timer starten  
}
```



```
void init() {  
    ...  
    //Lüfterüberwachung  
    impuls.fall(&impulsISR);  
    impuls.enable_irq();  
    //Timer TIM6  
    RCC->APB1ENR |= 0b10000; //TIM6 mit Takt versorgen  
    TIM6->PSC=31999; //Prescaler für 1ms  
    TIM6->CNT=0;    //Zähler bei 0 starten  
    TIM6->ARR=999;  //Autoreload für 1s (0.999 = 1000 Zählakte  
    TIM6->SR=0;    //Überlaufflag zurücksetzen  
    TIM6->DIER=1;   //Interrupt freigeben Timer  
    NVIC_SetVector(TIM6_IRQn, (uint32_t)&isrTIM6); //ISR zuordnen  
    HAL_NVIC_EnableIRQ(TIM6_IRQn); //Interrupt freigeben NVIC  
    TIM6->CR1=1;    //Timer starten  
}
```



```
void init() {  
    ...  
    //Lüfterüberwachung  
    impuls.fall(&impulsISR);  
    impuls.enable_irq();  
    //Timer TIM6  
    RCC->APB1ENR |= 0b10000; //TIM6 mit Takt versorgen  
    TIM6->PSC=31999; //Prescaler für 1ms  
    TIM6->CNT=0;    //Zähler bei 0 starten  
    TIM6->ARR=999;  //Autoreload für 1s (0.999 = 1000 Zählakte  
    TIM6->SR=0;    //Überlaufflag zurücksetzen  
    TIM6->DIER=1;   //Interrupt freigeben Timer  
    NVIC_SetVector(TIM6_IRQn, (uint32_t)&isrTIM6); //ISR zuordnen  
    HAL_NVIC_EnableIRQ(TIM6_IRQn); //Interrupt freigeben NVIC  
    TIM6->CR1=1;   //Timer starten  
}
```




```
void init() {  
    ...  
    //Lüfterüberwachung  
    impuls.fall(&impulsISR);  
    impuls.enable_irq();  
    //Timer TIM6  
    RCC->APB1ENR |= 0b10000; //TIM6 mit Takt versorgen  
    TIM6->PSC=31999; //Prescaler für 1ms  
    TIM6->CNT=0;    //Zähler bei 0 starten  
    TIM6->ARR=999;  //Autoreload für 1s (0.999 = 1000 Zählakte  
    TIM6->SR=0;    //Überlaufflag zurücksetzen  
    TIM6->DIER=1;   //Interrupt freigeben Timer  
    NVIC_SetVector(TIM6_IRQn, (uint32_t)&isrTIM6); //ISR zuordnen  
    HAL_NVIC_EnableIRQ(TIM6_IRQn); //Interrupt freigeben NVIC  
    TIM6->CR1=1;   //Timer starten  
}
```



```
void init() {  
    ...  
    //Lüfterüberwachung  
    impuls.fall(&impulsISR);  
    impuls.enable_irq();  
    //Timer TIM6  
    RCC->APB1ENR |= 0b10000; //TIM6 mit Takt versorgen  
    TIM6->PSC=31999; //Prescaler für 1ms  
    TIM6->CNT=0;    //Zähler bei 0 starten  
    TIM6->ARR=999;  //Autoreload für 1s (0.999 = 1000 Zählтакты  
    TIM6->SR=0;    //Überlaufflag zurücksetzen  
    TIM6->DIER=1;   //Interrupt freigeben Timer  
    NVIC_SetVector(TIM6_IRQn, (uint32_t)&isrTIM6); //ISR zuordnen  
    HAL_NVIC_EnableIRQ(TIM6_IRQn); //Interrupt freigeben NVIC  
    TIM6->CR1=1;    //Timer starten  
}
```



```
void init() {  
    ...  
    //Lüfterüberwachung  
    impuls.fall(&impulsISR);  
    impuls.enable_irq();  
    //Timer TIM6  
    RCC->APB1ENR |= 0b10000; //TIM6 mit Takt versorgen  
    TIM6->PSC=31999; //Prescaler für 1ms  
    TIM6->CNT=0;    //Zähler bei 0 starten  
    TIM6->ARR=999;  //Autoreload für 1s (0.999 = 1000 Zählakte  
    TIM6->SR=0;    //Überlaufflag zurücksetzen  
    TIM6->DIER=1;   //Interrupt freigeben Timer  
    NVIC_SetVector(TIM6_IRQn, (uint32_t)&isrTIM6); //ISR zuordnen  
    HAL_NVIC_EnableIRQ(TIM6_IRQn); //Interrupt freigeben NVIC  
    TIM6->CR1=1;   //Timer starten  
}
```



```
void init() {  
    ...  
    //Lüfterüberwachung  
    impuls.fall(&impulsISR);  
    impuls.enable_irq();  
    //Timer TIM6  
    RCC->APB1ENR |= 0b10000; //TIM6 mit Takt versorgen  
    TIM6->PSC=31999; //Prescaler für 1ms  
    TIM6->CNT=0;    //Zähler bei 0 starten  
    TIM6->ARR=999;  //Autoreload für 1s (0.999 = 1000 Zählakte  
    TIM6->SR=0;    //Überlaufflag zurücksetzen  
    TIM6->DIER=1;   //Interrupt freigeben Timer  
    NVIC_SetVector(TIM6_IRQn, (uint32_t)&isrTIM6); //ISR zuordnen  
    HAL_NVIC_EnableIRQ(TIM6_IRQn); //Interrupt freigeben NVIC  
    TIM6->CR1=1;   //Timer starten  
}
```

