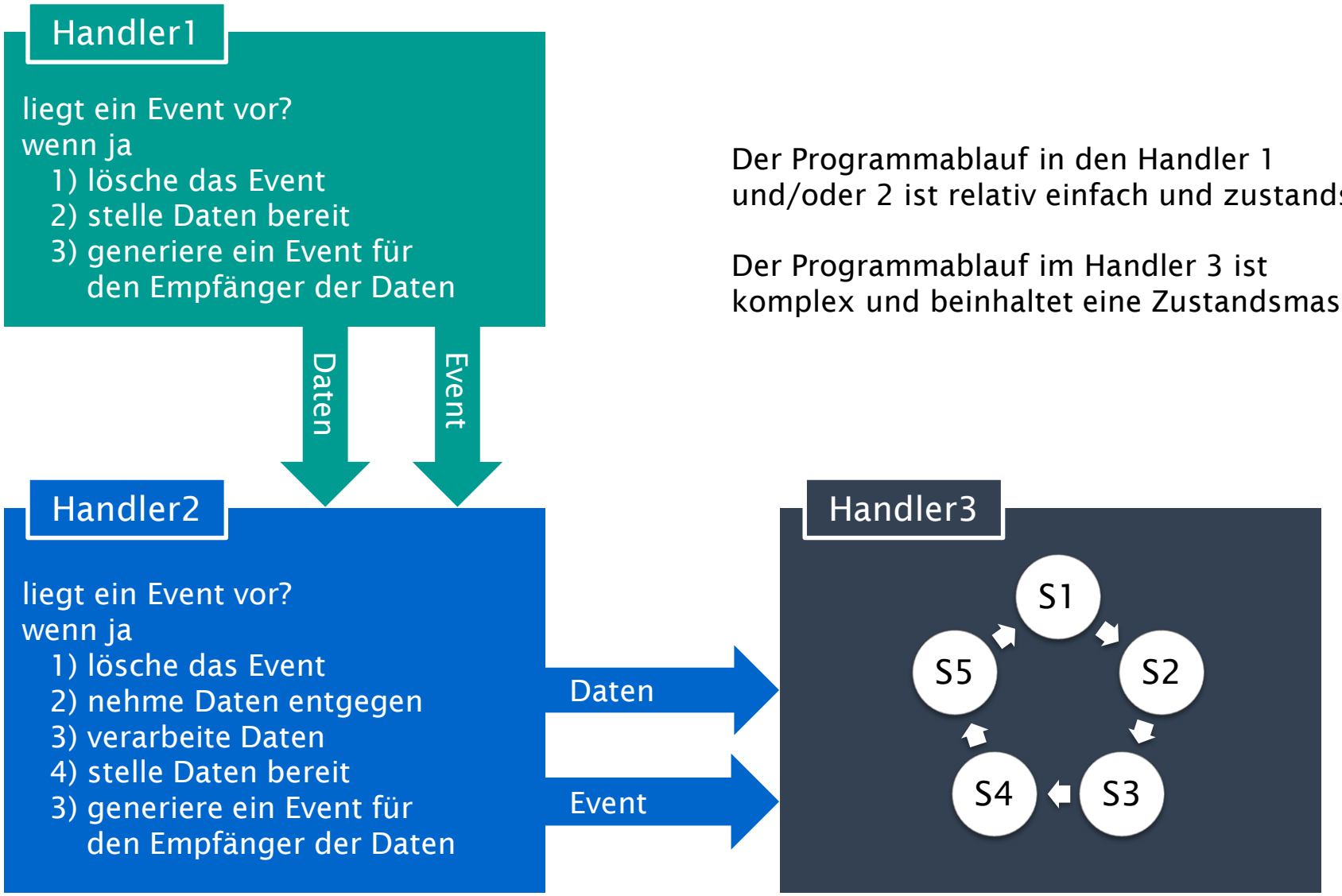
Kooperierende Prozesse/Handler



Der Programmablauf in den Handler 1 und/oder 2 ist relativ einfach und zustandslos.

Der Programmablauf im Handler 3 ist komplex und beinhaltet eine Zustandsmaschine

Kooperierende Prozesse/Handler

- Main-Funktion
  - Endlose while-Schleife mit der Warte-Anweisung auf Events und mit mehreren Handlern (Prozessen/Tasks)
  - Der Datenaustausch zwischen den Handlern erfolgt über gemeinsame Variablen/Speicherbereiche
  - Das Aktivieren eines Handlers erfolgt über Events

```
// Ausschnitt aus main.c

GLOBAL Void main(Void) {
   // Initialisierungen
   ...

   while(TRUE) {

      Event_wait();

      Handler1();
      Handler2();
      Handler3();
   }
}
```

Fallbeispiel: Timer

- Mit Hilfe eines Timers wird eine Ereignisquelle emuliert, die alle 50 ms periodisch ein Ereignis generiert

```
GLOBAL Void TA0_init(Void) {
   CLRBIT(TA0CTL, MC0 | MC1   // stop mode
                  | TAIE      // disable interrupt
                  | TAIFG);   // clear interrupt flag
   CLRBIT(TA0CCTL0, CM1 | CM0 // no capture mode
                  | CAP       // compare mode
                  | CCIE      // disable interrupt
                  | CCIFG);   // clear interrupt flag
   TA0CCR0  = TICK(50);       // set up Compare Register for 50 ms
   TA0EX0   = TAIDEX_7;       // set up expansion register /8
   TA0CTL   = TASSEL__ACLK    // 614.4 kHz
            | MC__UP          // Up Mode
            | ID__8           // /8
            | TACLR;          // clear and start Timer
   SETBIT(TA0CTL, TAIE);      // enable interrupt
}


#pragma vector = TIMER0_A1_VECTOR
__interrupt Void TIMER0_A1(Void) {
   TGLBIT(P2OUT, BIT7);
   CLRBIT(TA0CTL, TAIFG);           // clear interrupt flag
   Event_set(EVENT_RUN1);           // set up event
   __low_power_mode_off_on_exit();  // restore Active Mode on return
}
```

Fallbeispiel: Handler1

- Handler1 stellt Daten zur Verfügung (Producer)

Beispiel 7

```
// Application Note 4400
// Pseudo Random Number Generation Using Linear Feedback Shift Registers
// Quelle: https://www.maximintegrated.com/en/app-notes/index.mvp/id/4400

GLOBAL Int   rnd_value;
LOCAL  ULong lfsr32;
LOCAL  ULong lfsr31;

#pragma FUNC_ALWAYS_INLINE(Handler1_init)
GLOBAL Void Handler1_init(Void) {
   rnd_value = 0;
   lfsr32 = 0xABCDE;
   lfsr31 = 0x23456789;
}

LOCAL Void shift_lfsr(ULong *lfsr, ULong mask) {
   Int feedback = *lfsr BAND 0x1;
   *lfsr >>= 1;
   if (feedback EQ 1) {
      *lfsr ^= mask;
   }
}
```

```
LOCAL Int get_random(Void) {
   shift_lfsr(&lfsr32, POLY_MASK_32);
   shift_lfsr(&lfsr32, POLY_MASK_32);
   shift_lfsr(&lfsr31, POLY_MASK_31);
   return (lfsr32 XOR lfsr31) BAND 0xFFFF;
}
```

```
GLOBAL Void Handler1(Void) {
   if (NOT Event_tst(EVENT_RUN1))
      return;

   Event_clr(EVENT_RUN1);
   rnd_value = get_random();
   Event_set(EVENT_RUN2);
}
```

Fallbeispiel: Handler2

- Handler2 nimmt Daten vom Handler1 entgegen (Konsumer-Rolle), verarbeitet sie und leitet sie weiter (Producer-Rolle)

Beispiel 7

```
LOCAL  Char table[TABSIZE];
LOCAL  UInt idx;
GLOBAL Char * const tab = table;

#pragma FUNC_ALWAYS_INLINE(Handler2_init)
GLOBAL Void Handler2_init(Void) {
   idx = 0;
}
```

```
GLOBAL Void Handler2(Void) {
   if (Event_tst(EVENT_RUN2))
      return;

   Event_clr(EVENT_RUN2);
   Char ch = (Char)rnd_value;
   if (NOT between('A', ch, 'Z'))
      return;
   table[idx++] = ch;
   if (idx EQ TABSIZE) {
      idx = 0;
      Event_set(EVENT_RUN3);
   }
}
```

Fallbeispiel: Handler3 – Version 1

- Handler3 nimmt Daten vom Handler2 entgegen (Konsumer) und verarbeitet sie

Beispiel 7

```
#pragma FUNC_ALWAYS_INLINE(swap)
LOCAL Void swap(Char *arg1, Char *arg2) {
   Char ch = *arg2;
   *arg2 = *arg1;
   *arg1 = ch;
}

// Bubblesort
LOCAL Void  sort(Char * arr, Int n) {
   UInt i, j;
   for (i = 0; i LT n-1; i++) {
      for (j = 0; j LT n-i-1; j++) {
         if (arr[j] GT arr[j+1]) {
            swap(&arr[j], &arr[j+1]);
         }
      }
   }
}
```

```
// externe Tabelle -> Handler2
EXTERN Char * const tab;
```

```
GLOBAL Void Handler3(Void) {
   if (Event_tst(EVENT_RUN3)) {
      Event_clr(EVENT_RUN3);
      sort(tab, TABSIZE);
   }
}
```

Diese Implementierung hat leider den Nachteil, dass sie für andere Handler blockierend ist und so verhindert, dass Ereignisse im System rechtzeitig abgearbeitet werden können.

=> Ein Übergang auf eine Zustandsmaschine ist erforderlich

Fallbeispiel: Handler3

- Grundregel:
  - wenn ein Handler an mehreren Stellen auf Ereignisse wartet, oder (ineinander verschachtelte) Schleifen beinhaltet, oder zwar schleifenfrei ist aber dafür komplexe/rechenintensive Abläufe beinhaltet, so ist der Ablauf in so einem Handler mit Hilfe einer oder mehrerer Zustandsmaschinen zu implementiert.

- Der Übergang auf eine Implementierung mit einer Zustands-maschine kann u.U. anspruchsvoll sein. Es ist deshalb sinnvoll, die Implementierung in mehreren Schritten durchzuführen.

Fallbeispiel: Handler3 – Version 2

- Übergang von for-Schleifen auf while-Schleifen

Beispiel 7

```
for ( i=0; i < N; i++ ) {
   statement();
}
```

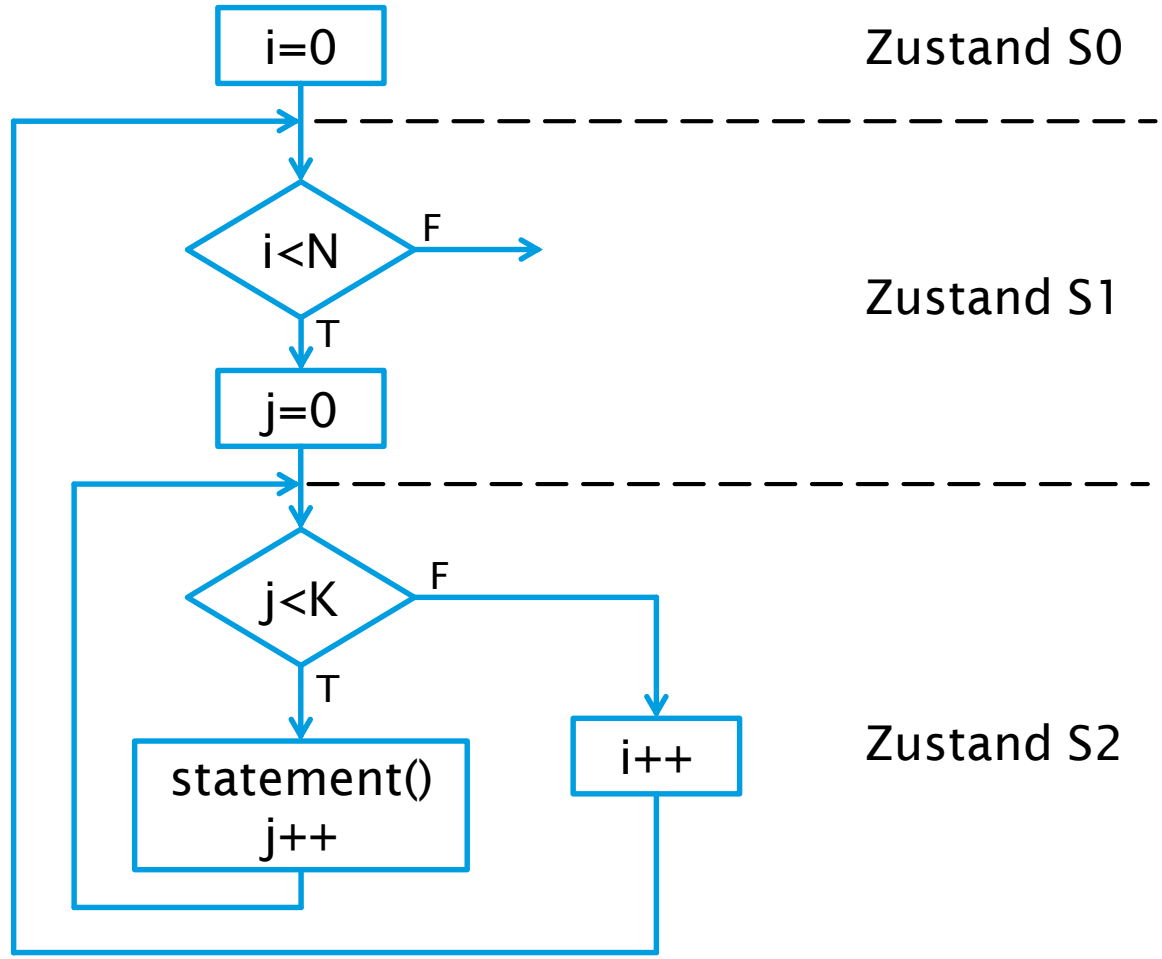
```
i=0;
while ( i < N ) {
   statement();
   i++;
}
```

```
LOCAL Void  sort(Char * arr, Int n) {
   UInt i = 0;
   while (i LT n-1) {
      UInt j = 0;
      while (j LT n-i-1) {
         if (arr[j] GT arr[j+1]) {
            swap(&arr[j], &arr[j+1]);
         }
         j++;
      }
      i++;
   }
}
```

Fallbeispiel: Handler3 – Version 3

- Übergang von while-Schleifen auf Zustände im Flußdiagramm

```
i = 0;

while (i < N) {
    j = 0;


    while (j < K) {
        statement();
        j++;
    }
    i++;
}
```

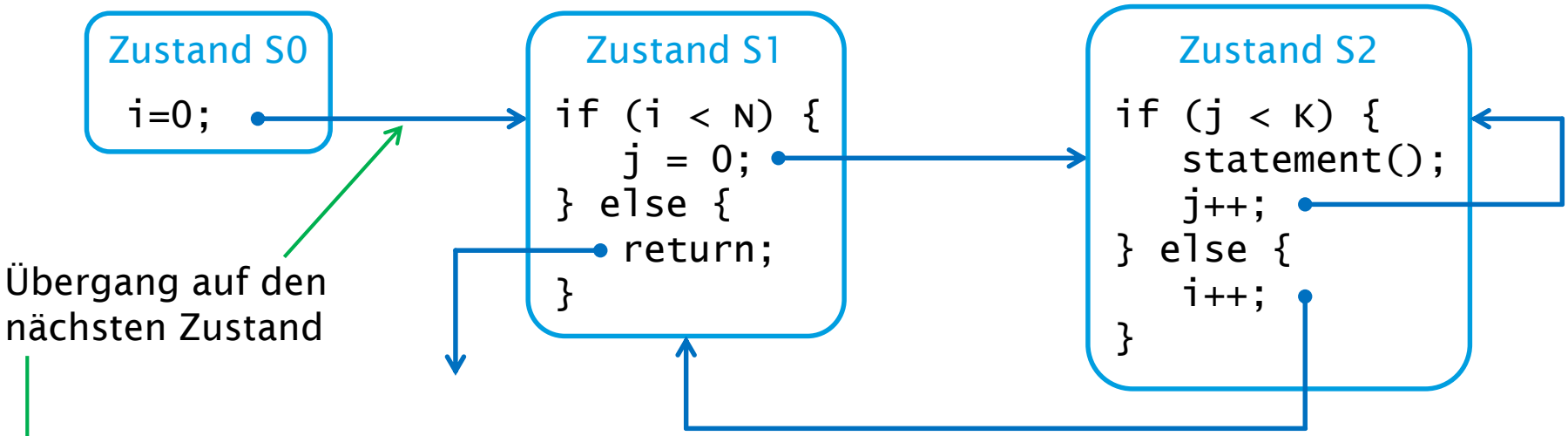


Fallbeispiel: Handler3 – Version 3

- Implementierung von Zustandsmaschinen als
  - verschachtelte switch-/case-Anweisungen
    - variable Zugriffszeit, abhängig vom Compiler (Tabelle/if-Bäume)
    - bei vielen Zuständen oder Abfragen oft unübersichtlich
  - Tabelle mit Funktionszeigern
    - konstante Zugriffszeit
    - relativ hoher Speicheraufwand
      - 10 Signale x 20 Zustände =>
        200 Funktionszeiger x 2 Byte/Zeiger => 400 Bytes
    - oft dünn besetzt => Komprimierung notwendig
  - Zustandsvariable mit Funktionszeigern
    - jedem Zustand wird eine Funktion zugeordnet
    - nur lokale switch-Anweisungen
    - leicht zu implementieren und übersichtlich
    - konstante Zugriffszeit

Fallbeispiel: Handler3 – Version 3

- Übergang vom Flussdiagramm auf Zustände im Zustandsgraphen




```
case S0:
  i = 0;
  state = S1;
  break;
```

```
case S1:
  if (i < N) {
    j = 0;
    state = S2;
  } else {
    state = S0;
    return;
  }
  break;
```

```
case S2:
  if (j < K) {
    statement();
    j++;
    state = S2;
  } else {
    i++;
    state = S1;
  }
  break;
```

Fallbeispiel: Handler3 – Version 3

- Übergang auf eine Zustandsmaschine

```
typedef enum {S0=0, S1, S2} TState;

LOCAL TState state = S0;

LOCAL Void  sort(Char * arr, Int n) {
   UInt i, j;

   while (1) {

      switch (state) {

         Zustände

         default:
            break;
      }
   }

}
```

Beispiel 7

```
case S0:
   i = 0;
   state = S1;
   break;

case S1:
   if (i LT n-1) {
      j = 0;
      state = S2;
   } else {
      state = S0;
      return;       // Ablaufende
   }
   break;

case S2:
   if (j LT n-i-1) {
      if (arr[j] GT arr[j+1]) {
         swap(&arr[j], &arr[j+1]);
      }
      j++;
      state = S2;
   } else {
      i++;
      state = S1;
   }
   break;
```

Fallbeispiel: Handler3 – Version 4

- Eliminierung der while-Schleife

```
typedef enum {S0=0, S1, S2} TState;

LOCAL TState state = S0;
LOCAL UInt i, j;

GLOBAL Void Handler3(Void) {
   switch (state) {

      Zustände

      default:
         break;
   }
}
```

Beispiel 7

```
case S0:
   if (Event_tst(EVENT_RUN3)) {
      i = 0;
      state = S1;
   }
   break;

case S1:
   if (i LT TABSIZE-1) {
      j = 0;
      state = S2;
   } else {
      Event_clr(EVENT_RUN3);
      state = S0;
   }
   break;

case S2:
   if (j LT TABSIZE-i-1) {
      if (tab[j] GT tab[j+1]) {
         swap(&tab[j], &tab[j+1]);
      }
      j++;
   } else {
      i++;
      state = S1;
   }
   break;
```

Fallbeispiel: Handler3 – Version 5

- Übergang auf eine Zustandstabelle mit konstanten Funktionspointern

```
// Datentyp eines konstanten Funktionspointers
typedef Void (* const VoidFunc)(Void);

typedef enum {S0=0, S1, S2} TState;

// lokale Zustandsvariable
LOCAL TState state = S0;
LOCAL UInt i, j;
```

Zustände als Funktionen

```
// Tabelle mit konstanten Funktionspointern
LOCAL const VoidFunc run[TRUNSIZE] = {State0,
                                      State1,
                                      State2};
```

```
GLOBAL Void Handler3(Void) {
   if (state LT TRUNSIZE) {
      run[state]();
   }
}
```

Beispiel 7

```
LOCAL Void State0(Void) {
   if (Event_tst(EVENT_RUN3)) {
      i = 0;
      state = S1;
   }
}

LOCAL Void State1(Void) {
   if (i LT TABSIZE-1) {
      j = 0;
      state = S2;
   } else {
      Event_clr(EVENT_RUN3);
      state = S0;
   }
}

LOCAL Void State2(Void) {
   if (j LT TABSIZE-i-1) {
      if (tab[j] GT tab[j+1]) {
         swap(&tab[j], &tab[j+1]);
      }
      j++;
   } else {
      i++;
      state = S1;
   }
}
```

Fallbeispiel: Handler3 – Version 6

- Übergang auf eine Lösung nur mit Funktionspointern

```
// Datentyp eines Funktionspointers
typedef Void (* VoidFunc)(Void);

LOCAL Void State0(Void);
LOCAL Void State1(Void);
LOCAL Void State2(Void);

// lokale Zustandsvariable
LOCAL VoidFunc state = State0;
LOCAL UInt i, j;
```

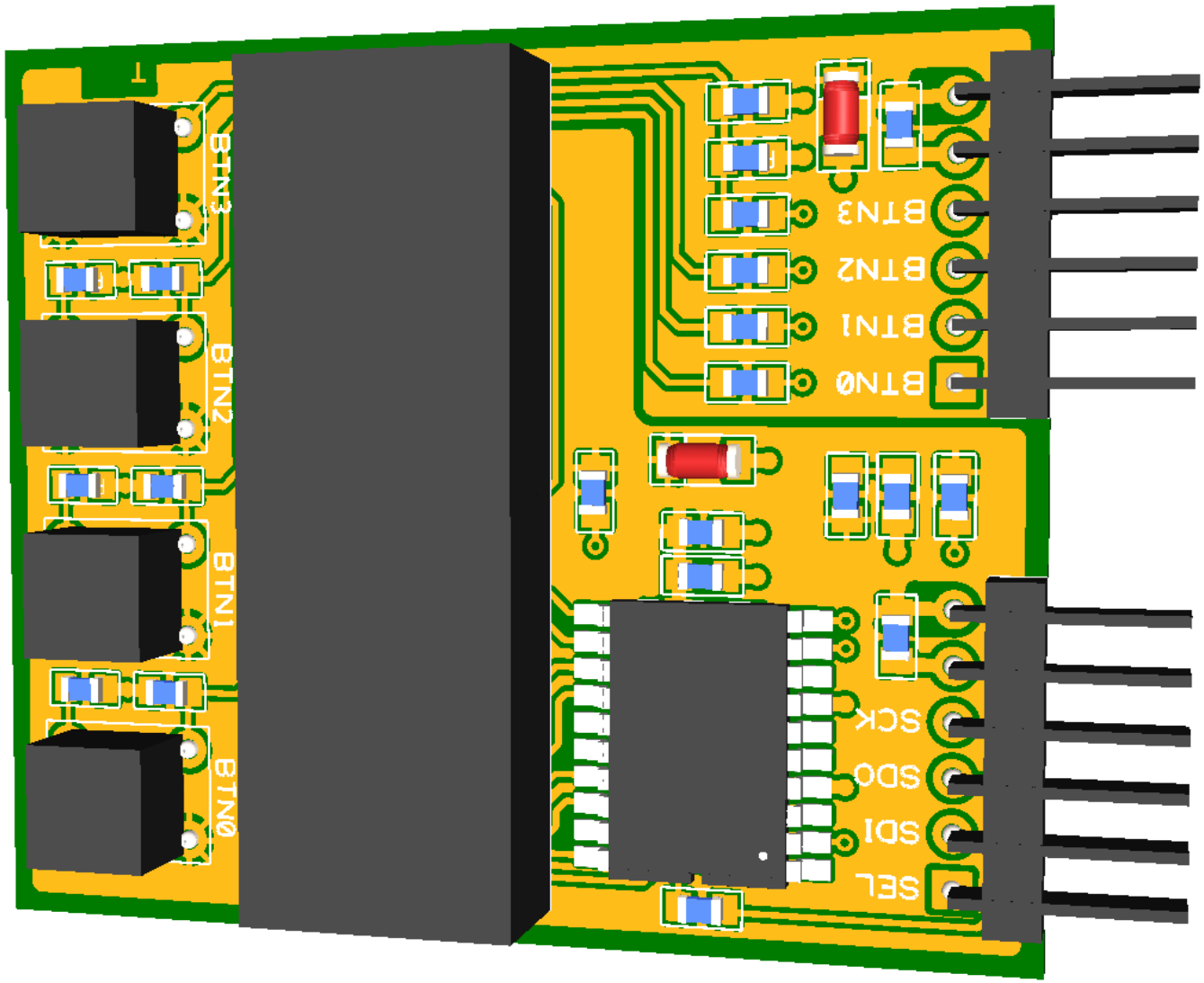
Zustände als Funktionen

```
GLOBAL Void Handler3(Void) {
   (*state)();
}
```

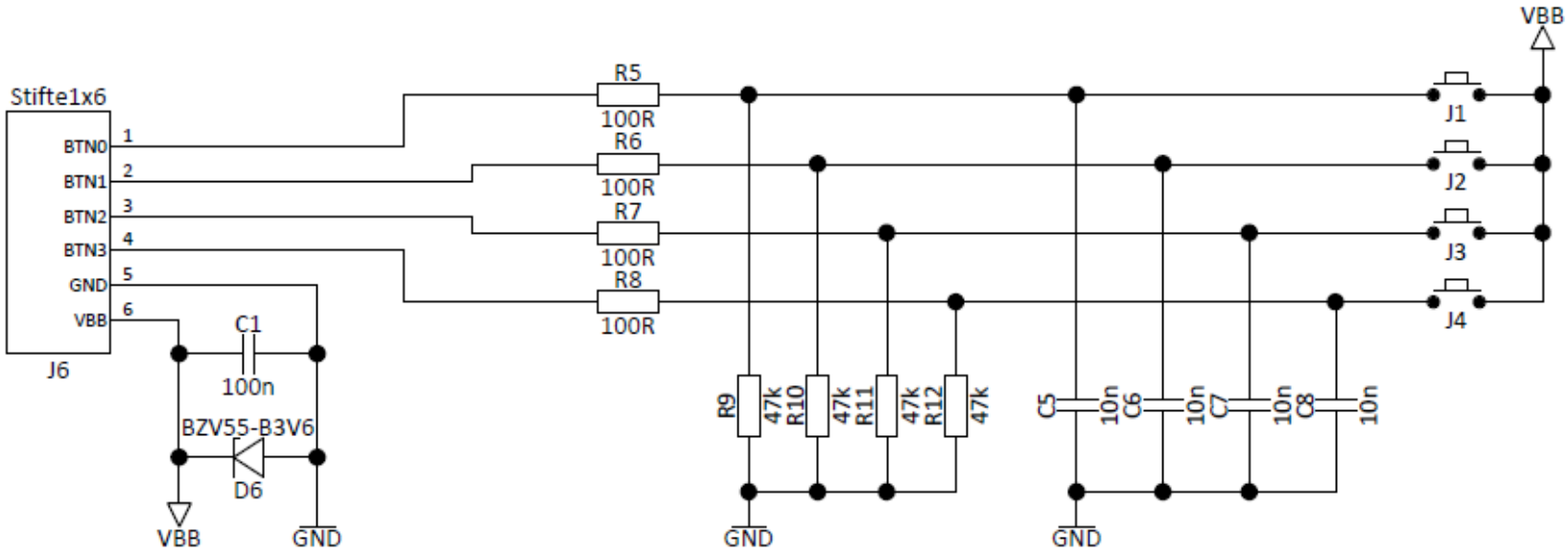
Beispiel 7

```
LOCAL Void State0(Void) {
   if (Event_tst(EVENT_RUN3)) {
      i = 0;
      state = State1;
   }
}

LOCAL Void State1(Void) {
   if (i LT TABSIZE-1) {
      j = 0;
      state = State2;
   } else {
      Event_clr(EVENT_RUN3);
      state = State0;
   }
}

LOCAL Void State2(Void) {
   if (j LT TABSIZE-i-1) {
      if (tab[j] GT tab[j+1]) {
         swap(&tab[j], &tab[j+1]);
      }
      j++;
   } else {
      i++;
      state = State1;
   }
}
```

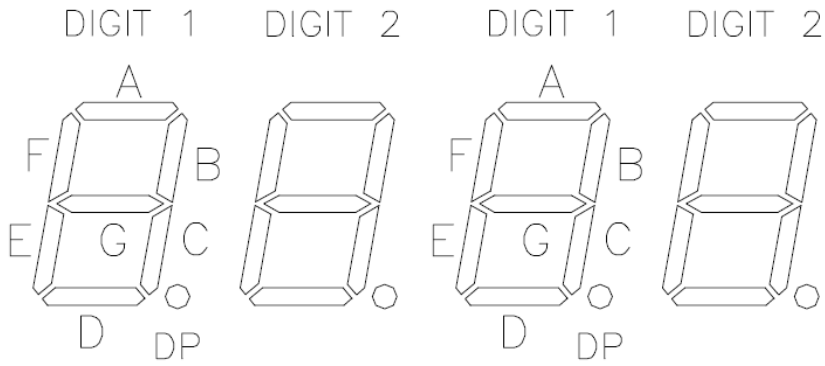
KEY2LED: Platine

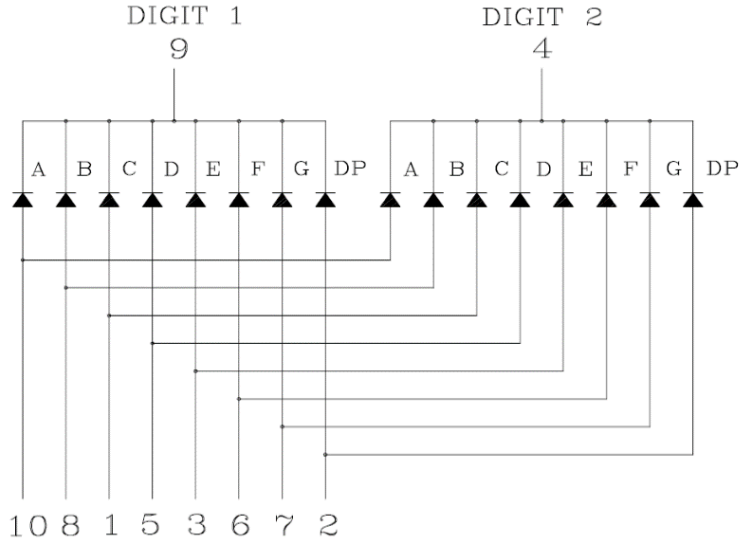


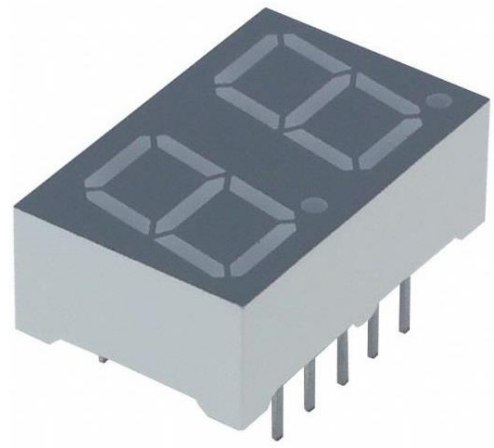
KEY2LED: Vier Taster



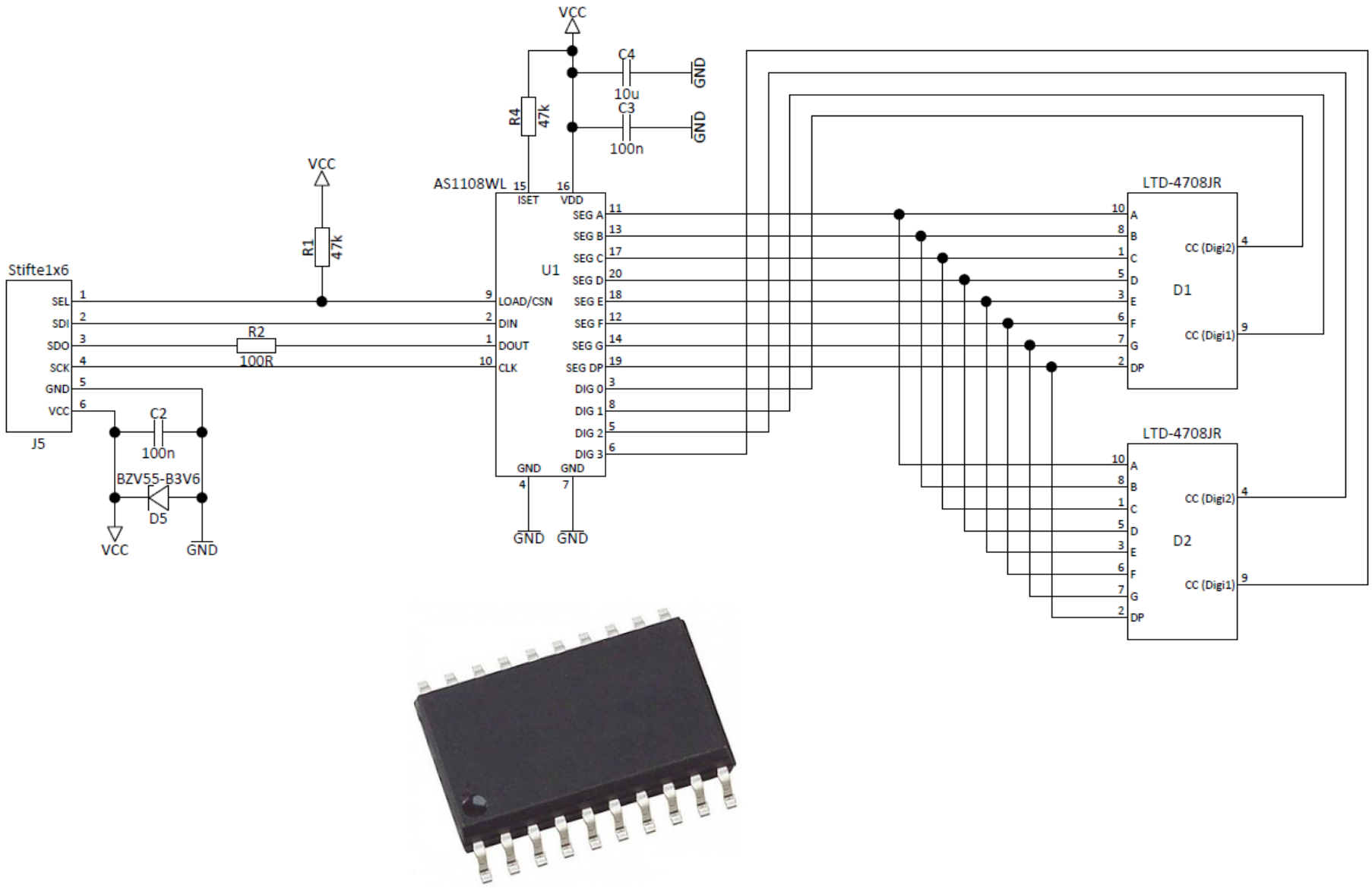
KEY2LED: 4 stellige 7-Segment-Anzeige



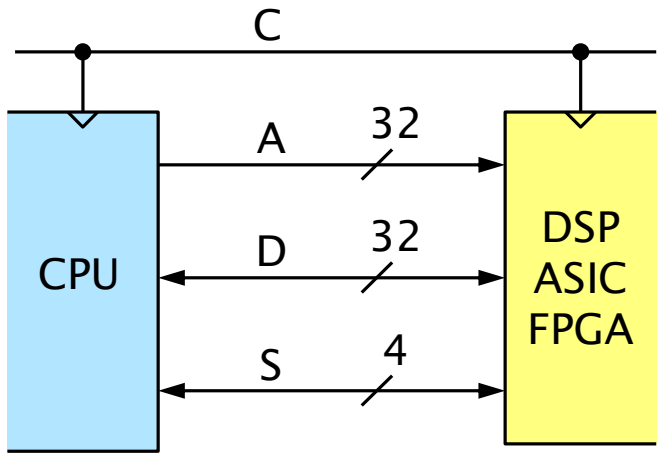


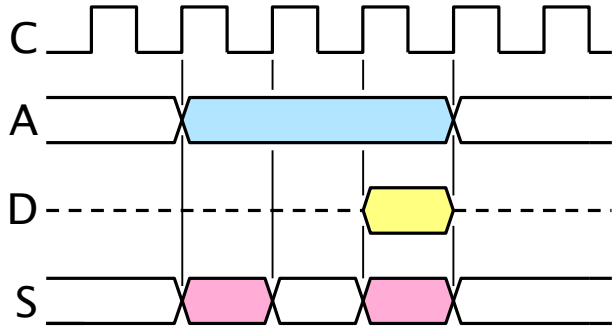


KEY2LED: Ansteuerung der 7-Segment-Anzeige



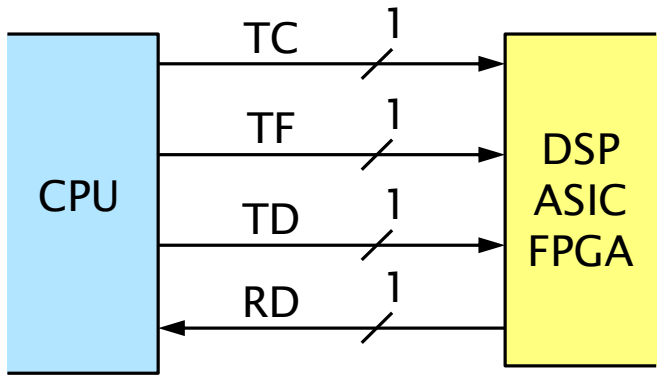
Parallele Datenübertragung

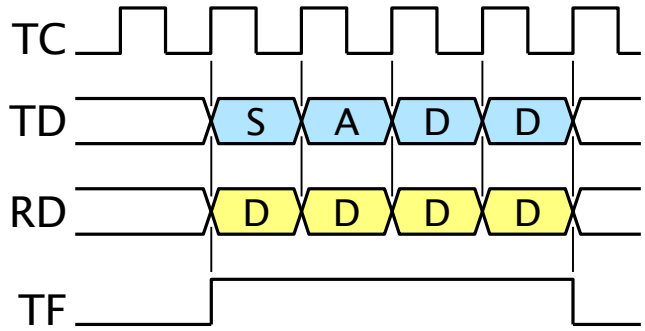




- Bus-/Übertragungsmerkmale:
  - viele parallele, bi-/unidirektionale Leitungen für Adressen (A), Daten (D) und Steuersignale (S)
  - Master-Slave-Betrieb mit P2P-Verbindung im Halbduplex-Betrieb
  - lineare (multimasterfähige) Multipunkt-Busarchitektur
- PCB-Entwurf:
  - komplizierte Entflechtung von Leiterplatten
  - großflächig, mehrlagig (4+)
  - Signalintegrität bei hohen Geschwindigkeiten
- Beispiele:
  - Prozessor-/CPU-Bus
  - PCI-Bus, VMEbus, SCSI, IEC

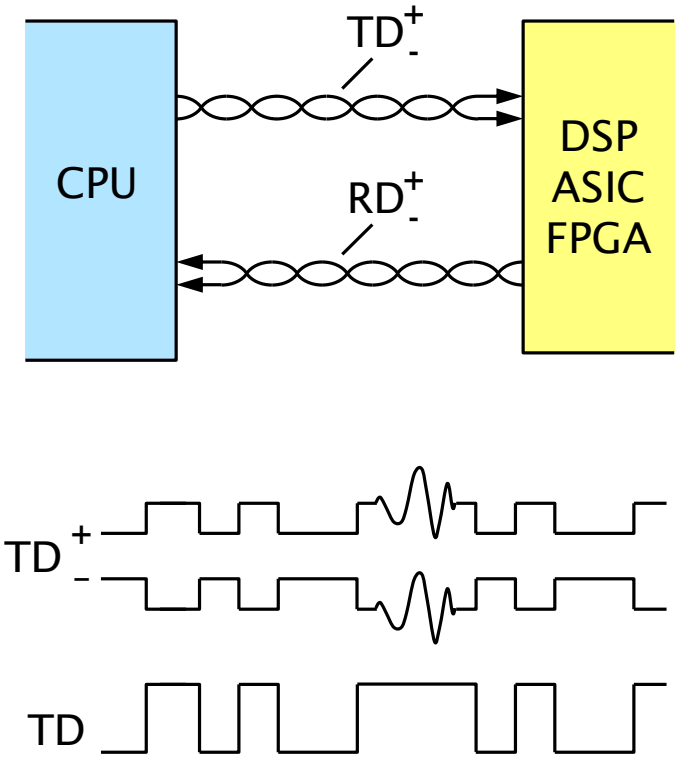
Bitserielle Datenübertragung





- Bus-/Übertragungsmerkmale:
  - vier unidirektionale Leitungen für Takt (TC), Rahmen (TF), und Daten (TD und RD)
  - Master-Slave-Betrieb mit P2P-Verbindung im Voll-/Halbduplex-Betrieb (Bus, Stern, Baum)
  - zeitsequentieller Transfer mit Übertragungsrahmen (Frame): Steuerfeld (S), Adressfeld (A), Datenfeld (D)
- PCB-Entwurf:
  - vereinfachte Entflechtung der Leiterplatte
  - weniger Probleme mit der Signalintegrität
- Beispiele:
  - SPI, I^2C, IDL

(A)synchron-serielle Datenübertragung

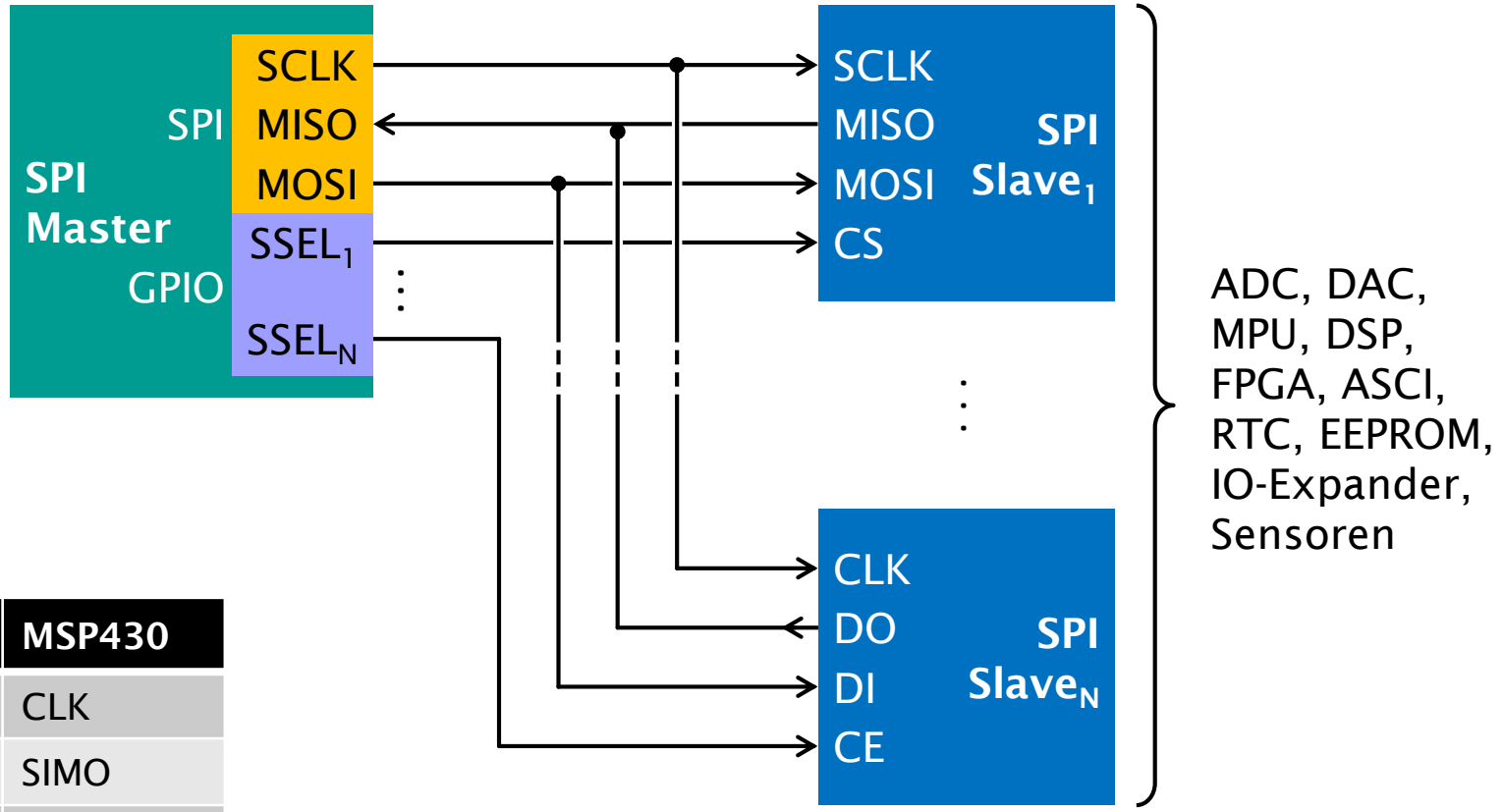


- Bus-/Übertragungsmerkmale:
  - zwei unidirektionale, differentielle Leitungspaare (TD+ und TD-) und (RD+ und RD-) mit symmetrischer Signalübertragung
  - Master-Slave-Betrieb mit P2P-Verbindung im Vollduplex-Betrieb
  - zeitsequentieller Transfer mit Übertragungsrahmen (Frame): Steuerfeld (S), Adressfeld (A), Datenfeld (D)
- Takt-/Rahmenkennung:
  - Start-Stopp-Verfahren
  - Leitungscode: Manchester-Codierung, 8B/10B-Codierung
- Beispiele:
  - FPD-Link, GVIF, USB, FireWire

SPI-Bus: wesentliche Merkmale

- entwickelt 1985 von Motorola (NXP Semiconductor)
- synchrone, serielle Schnittstelle für Kommunikation von Gerät zu Gerät über kurze Entfernung (<< 1 m)
- heute quasi Industriestandard geworden
- einfache, kostengünstige Schnittstelle mit geringem Overhead und vollständiger Flexibilität bei der Anzahl der übertragenen Bits
- Master-Slave-Modell mit einem Master und ggf. mit mehreren Slave-Geräten, die mit Taktgeschwindigkeiten von bis zu 50 MHz per Vollduplex-Datenübertragung bedient werden
- kein Standardprotokoll, überträgt nur Datenpakete, eignet sich ideal auch für die Übertragung langer Datenströme

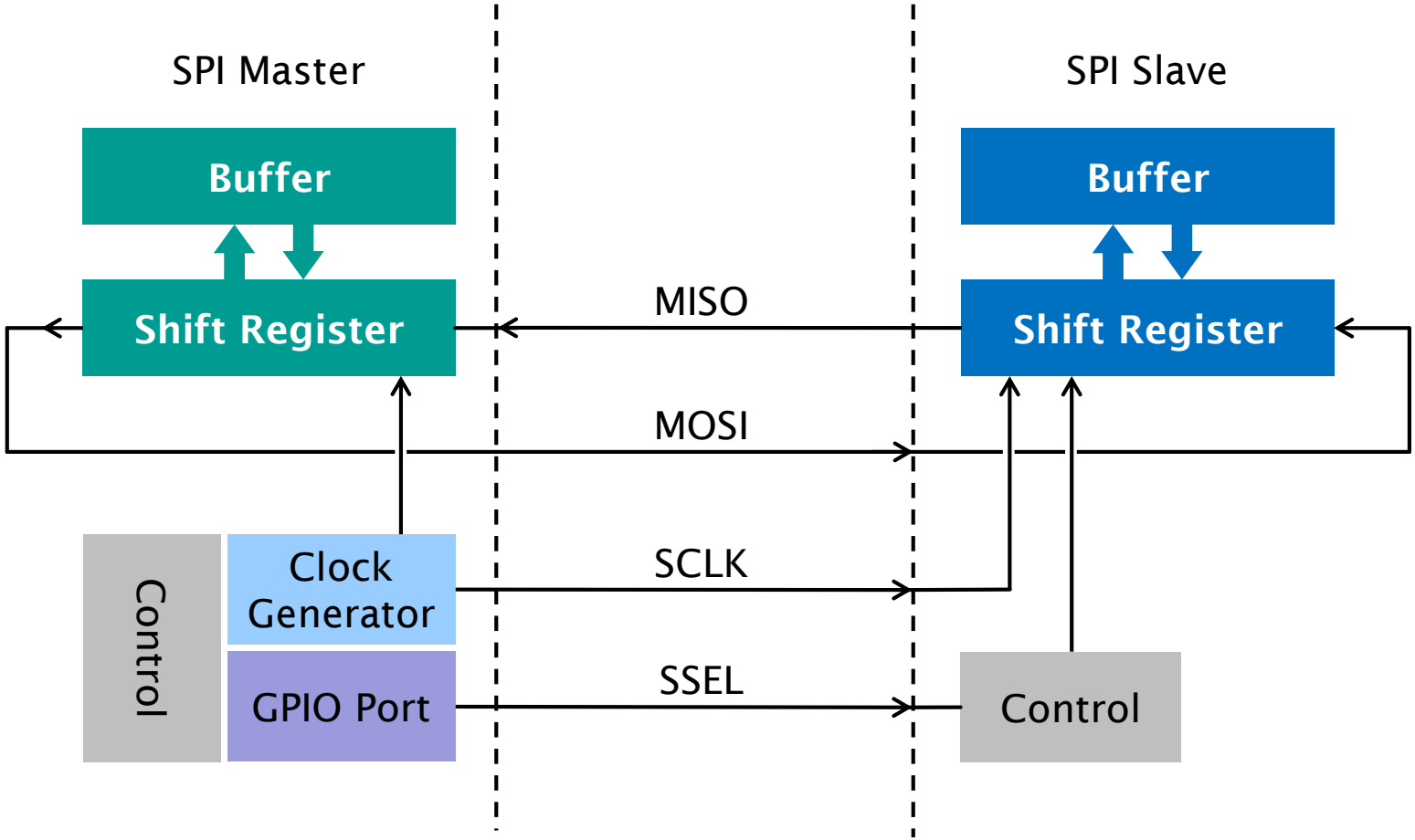
SPI-Bus: Master-Slave-Konfiguration



Motorola	MSP430
SCLK	CLK
MOSI	SIMO
MISO	SOMI
SS	STE

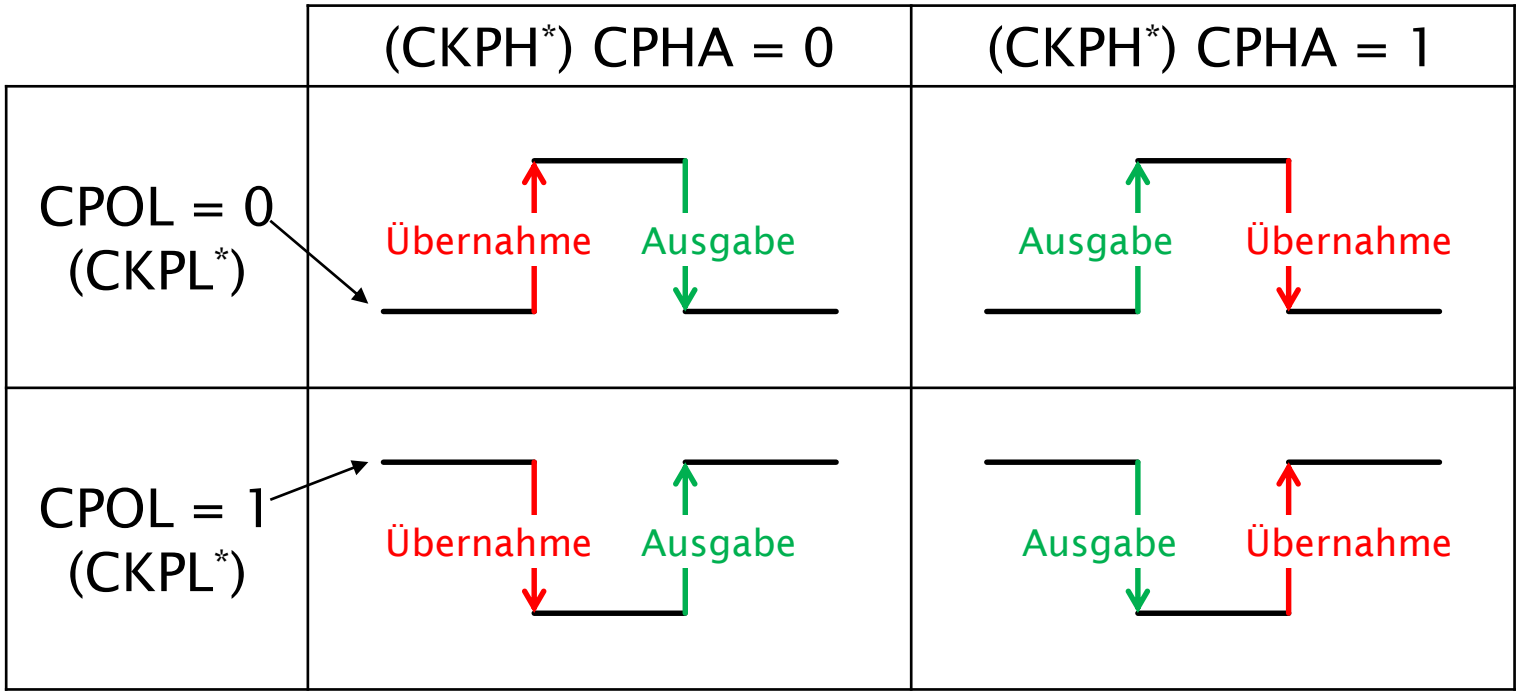
CS- und CE-Signale können mit
Low oder High aktiv sein,
anhängig von der SPI-Komponente

SPI-Registerstruktur



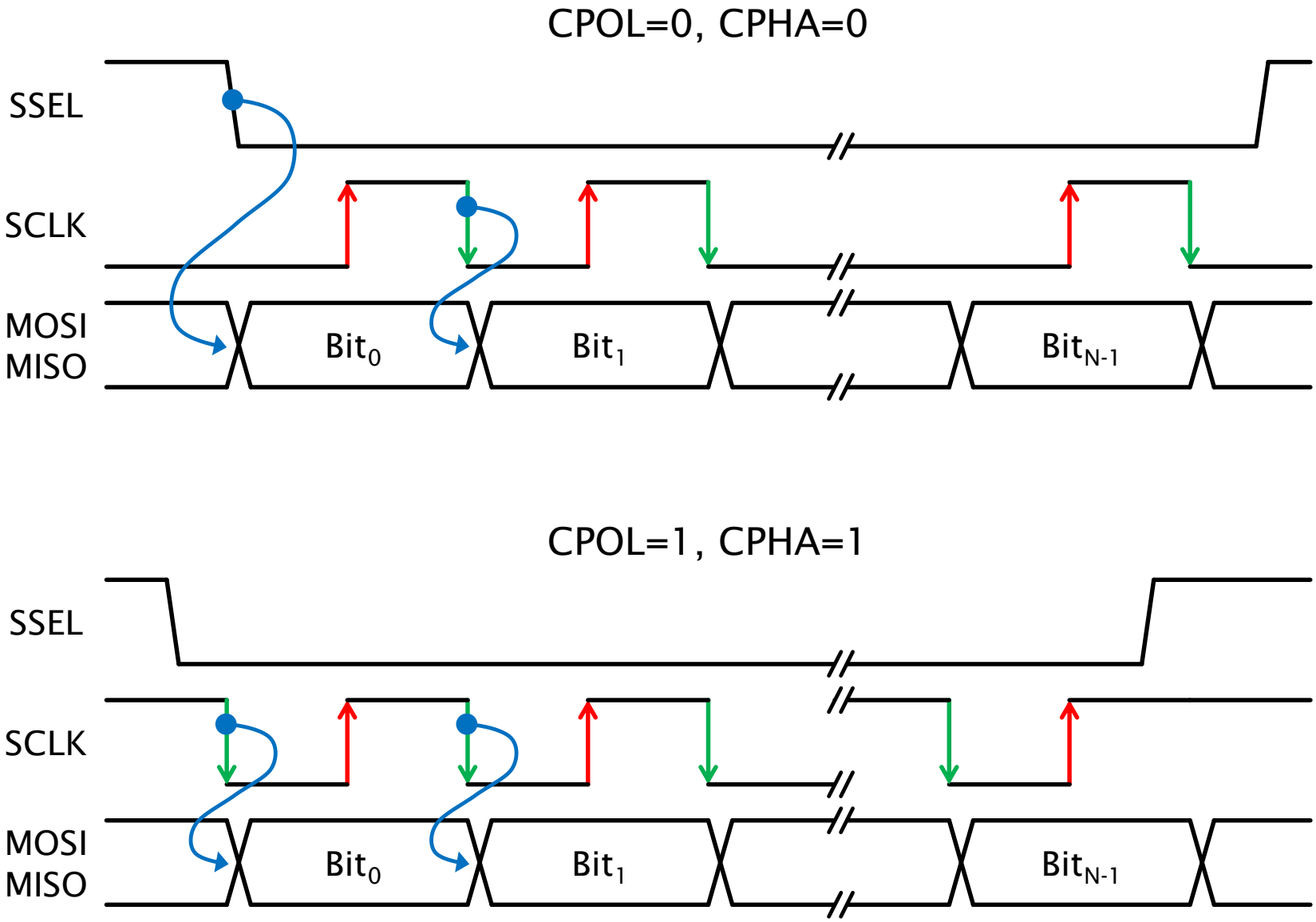
SPI-Taktmodi

- Der SPI-Master erzeugt und steuert den SPI-Takt
- Die Einstellung des SPI-Takts erfolgt durch zwei Taktattribute: Taktpolarität (CPOL) und Taktphase (CPHA)
  und legt die Taktflanke fest, mit der Daten abgetastet und übernommen werden.

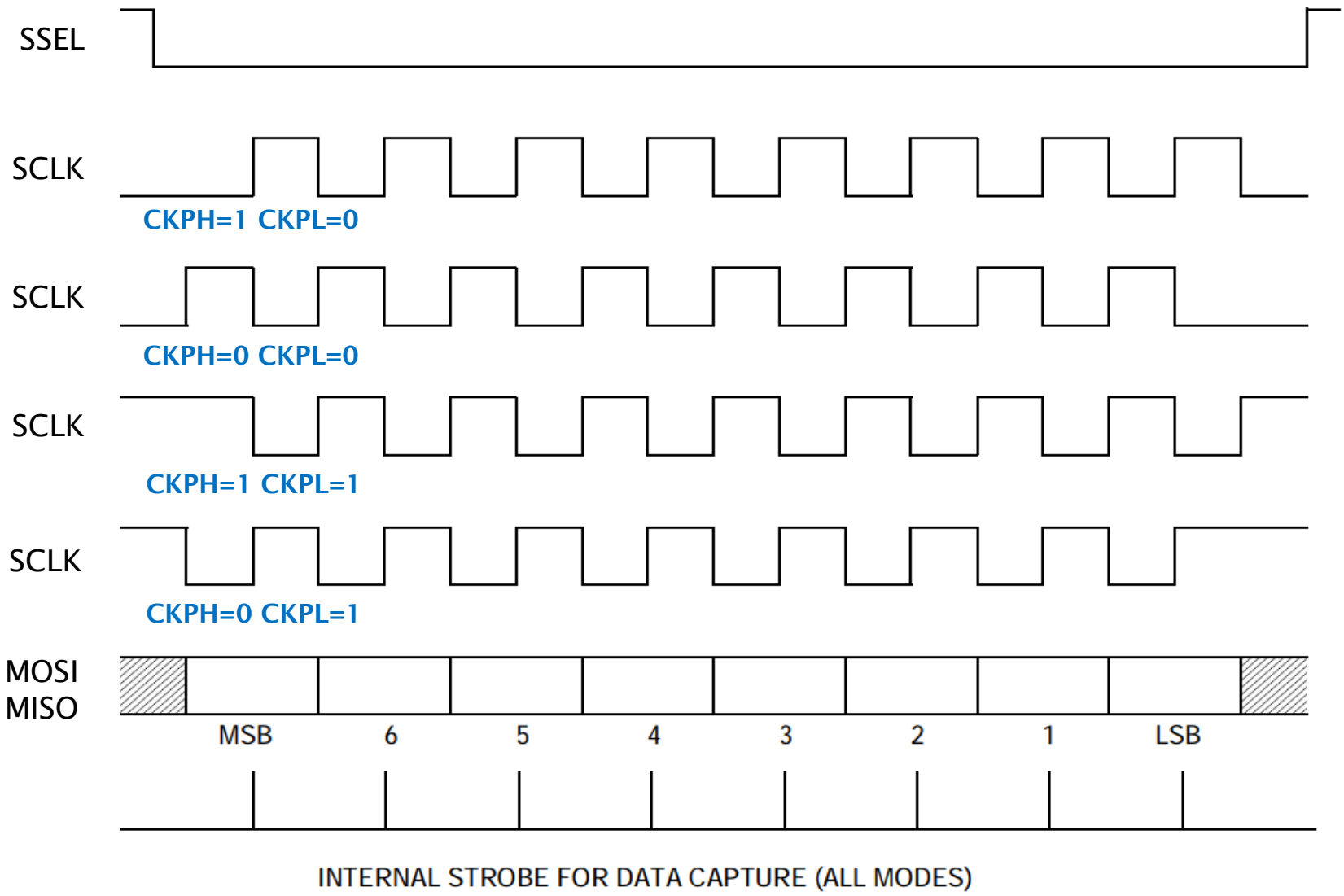


(*) Bezeichnungen bei MSP430

SPI-Timing


CPOL=0, CPHA=0
SSEL
SCLK
MOSI
MISO
Bit0
Bit1
BitN-1
CPOL=1, CPHA=1
SSEL
SCLK
MOSI
MISO
Bit0
Bit1
BitN-1

SPI-Timing


SSEL
SCLK
CKPH=1 CKPL=0
SCLK
CKPH=0 CKPL=0
SCLK
CKPH=1 CKPL=1
SCLK
CKPH=0 CKPL=1
MOSI
MISO
MSB
6
5
4
3
2
1
LSB
INTERNAL STROBE FOR DATA CAPTURE (ALL MODES)

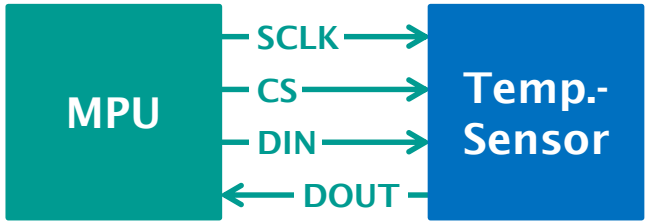
SPI-Timing

- Es gibt zwei typische SPI-Transfer-Konstellationen:
- SPI-Slave (z.B. Sensoren, AD-Wandler, Receiver im UART, EEPROM im Lesemodus) gibt Daten aus, die im SPI-Master (MCU) übernommen werden.
  - Beispiel: Werden Daten im SPI-Slave mit fallender SCLK-Flanke ausgegeben, dann erfolgt die Datenübernahme im SPI-Master mit steigender SCLK-Flanke.
- SPI-Master (MCU) gibt Daten aus, die im SPI-Slave (z.B. Aktoren, DA-Wandler, Transmitter im UART, EEPROM im Schreibmodus) übernommen werden.
  - Beispiel: Werden Daten im SPI-Slave mit steigender SCLK-Flanke übernommen, dann erfolgt die Datenausgabe im SPI-Master mit fallender SCLK-Flanke.

SPI-Timing

- Hinweise zur Einstellung von CPOL und CPHA
  - Datenblätter derjenigen (Peripherie-)Komponenten, die als SPI-Slave arbeiten sollten, sorgfältig studieren.
  - Man findet dort Informationen über die Taktpolarität und die Taktphase oft direkt im Klartext.
  - Timing-Diagramme analysieren: dort lassen sich oft Zusatzinformationen erkennen:
    - der Pegel von SCLK im Idle-Zustand
    - die SCLK-Flanke, mit der Daten ausgegeben bzw. übernommen werden
    - Zeitabhängigkeiten zwischen den einzelnen Phasen beachten.
  - Einstellungen von CPOL und CPHA beziehen sich immer auf den SPI-Takt im Mikrocontroller, basieren aber auf den Timing-Informationen einer Peripheriekomponente.

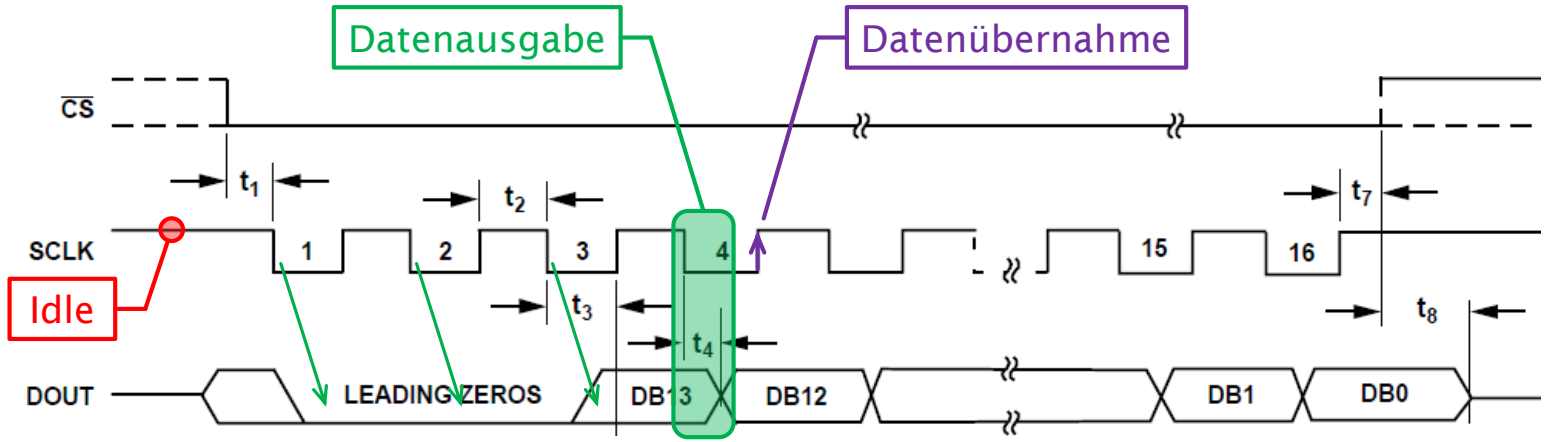
SPI-Timing: ADT7301 (14-Bit Temp.-Sensor)



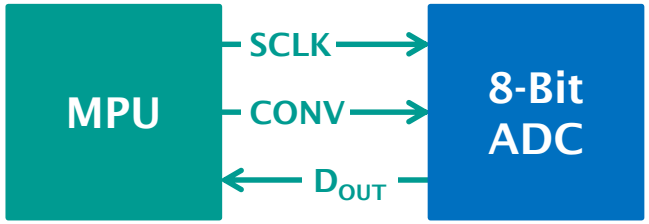
SCLK-Einstellung:
CPOL=1, CPHA=1

Parameter	Limit	Unit
t_1	5	ns (min)
t_2	25	ns (min)
t_3	25	ns (min)
t_4	35	ns (max)
t_7	5	ns (min)
t_8	40	ns (max)

Idle von SCLK ist HIGH => CPOL=1
Die Datenausgabe auf DOUT erfolgt mit fallender SCLK-Flanke nach t_4 => die Datenübernahme im MCU muss mit steigender SCLK-Flanke erfolgen => CPHA=1

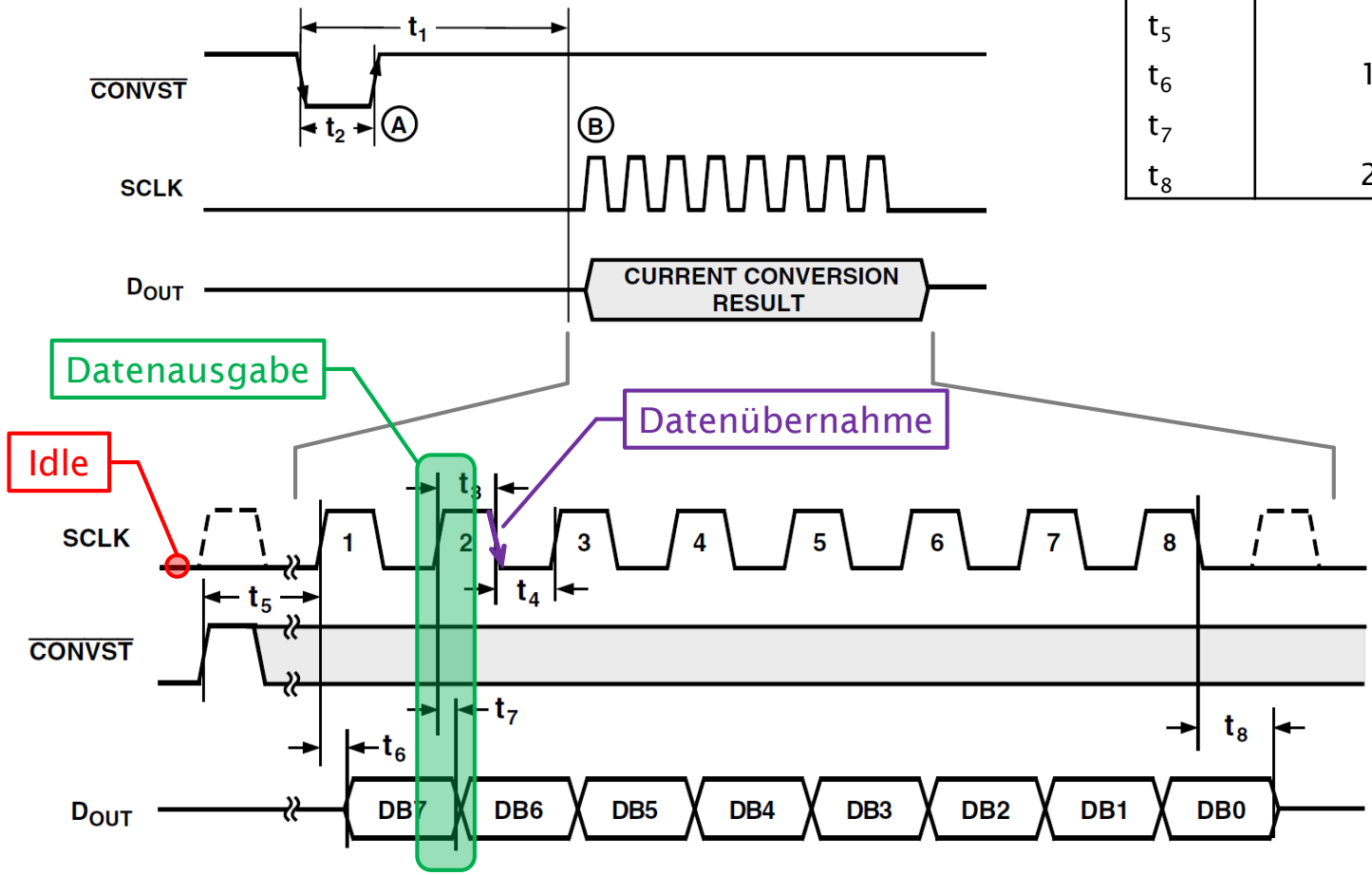


SPI-Timing: AD7823 (8-Bit-ADC)

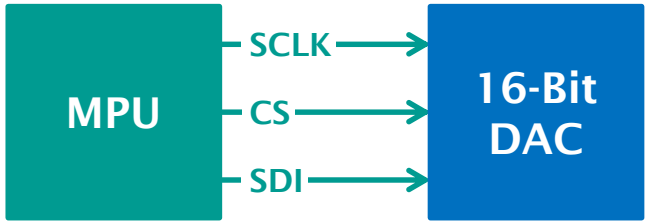


SCLK-Einstellung:
CPOL=0, CPHA=1

Param.	V_{DD} = +3V	Unit
t_1	5	µs (max)
t_2	20	ns (min)
t_3	25	ns (min)
t_4	25	ns (min)
t_5	5	ns (min)
t_6	10	ns (min)
t_7	5	ns (min)
t_8	20	ns (min)



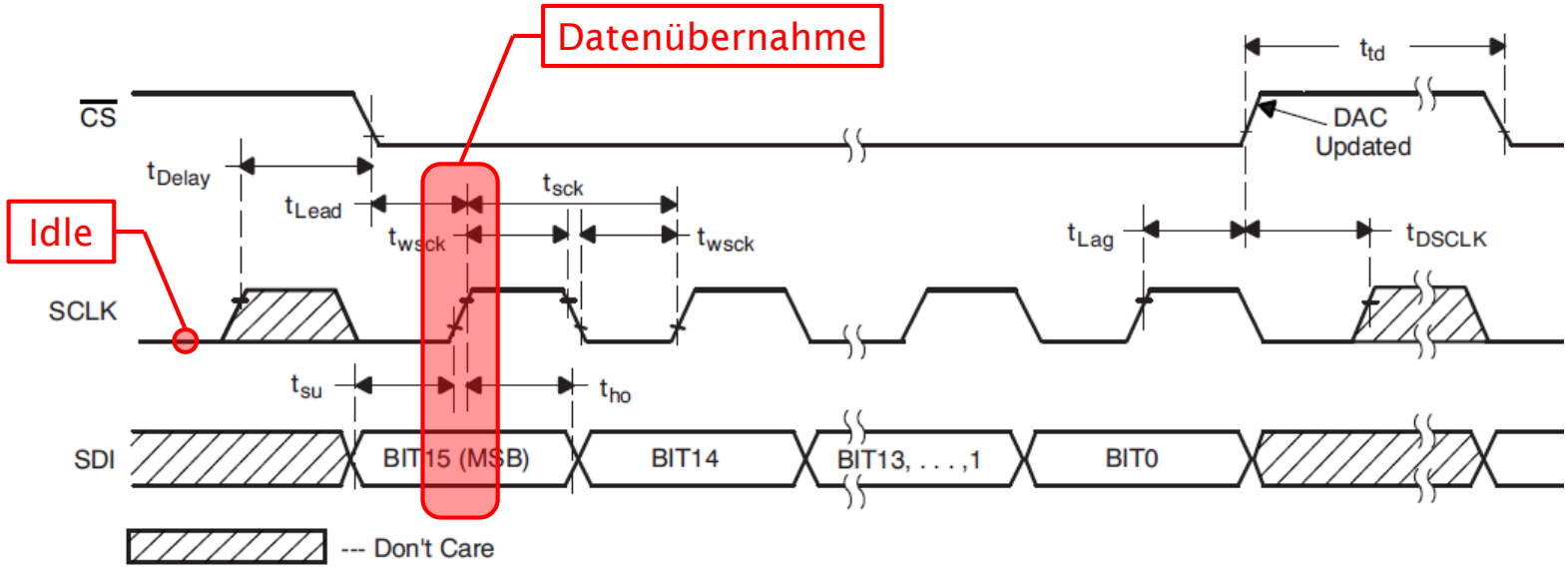
SPI-Timing: DAC8830 (16-Bit DAC)



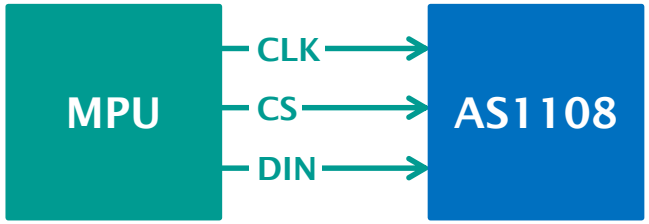
SCLK-Einstellung:
CPOL=0, CPHA=0

Param.	Min	Max	Unit
t_{SCK}	20	-	ns
t_{WSCK}	10	-	ns
t_{Delay}	10	-	ns
t_{Lead}	10	-	ns
t_{Lag}	10	-	ns
t_{DSCLK}	10	-	ns
t_{td}	30	-	ns
t_{su}	10	-	ns
t_{ho}	0	-	ns

Idle von SCLK ist LOW => CPOL=0
Die Datenübernahme im DAC erfolgt mit steigender SCLK-Flanke => die Datenausgabe im MCU muss mit fallender SCLK-Flanke erfolgen => CPHA=0



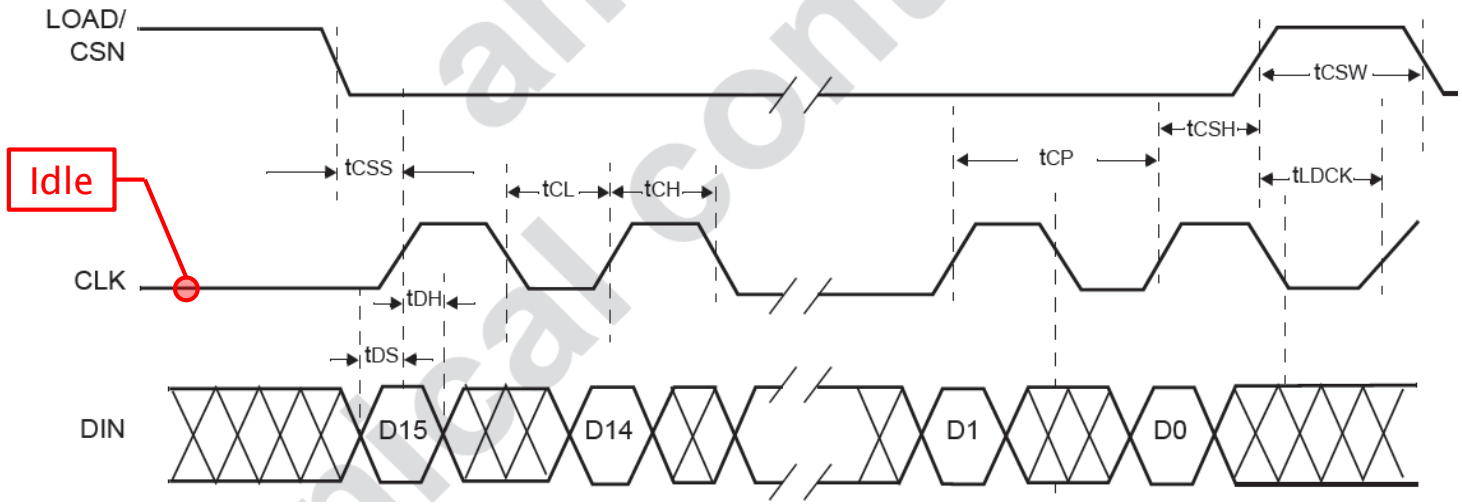
SPI-Timing: AS1108 (4-Digit LED Treiber)



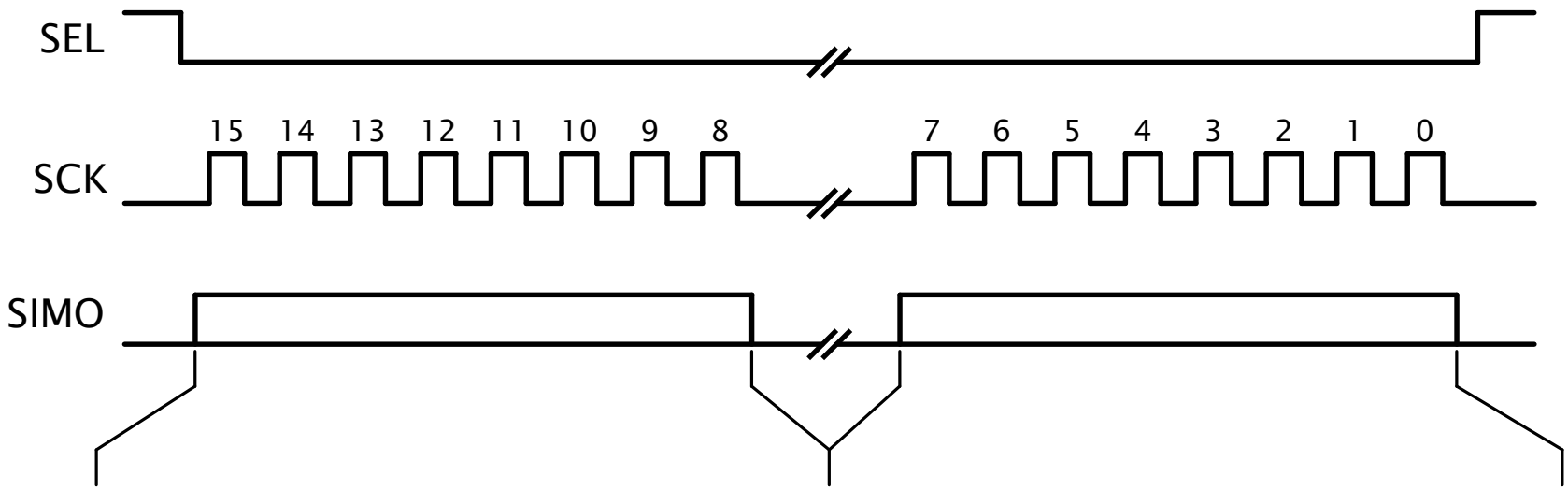
Symbol	Conditions	Min	Typ	Max	Unit
tCP		100			ns
tCH		50			ns
tCL		50			ns
tCSS		25			ns
tCSH		0			ns
tDS		25			ns
tDH		0			ns
tDO	CLOAD = 50pF			25	ns
tLDCK		50			ns
tCSW		50			ns
tDSPD				2.25	ms

Idle von CLK ist LOW => CPOL=0
Die Datenübernahme im AS1108 erfolgt mit steigender CLK-Flanke => die Datenausgabe im MCU muss mit fallender CLK-Flanke erfolgen => CPHA=0

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	Register Adresse				MSB			Data				LSB

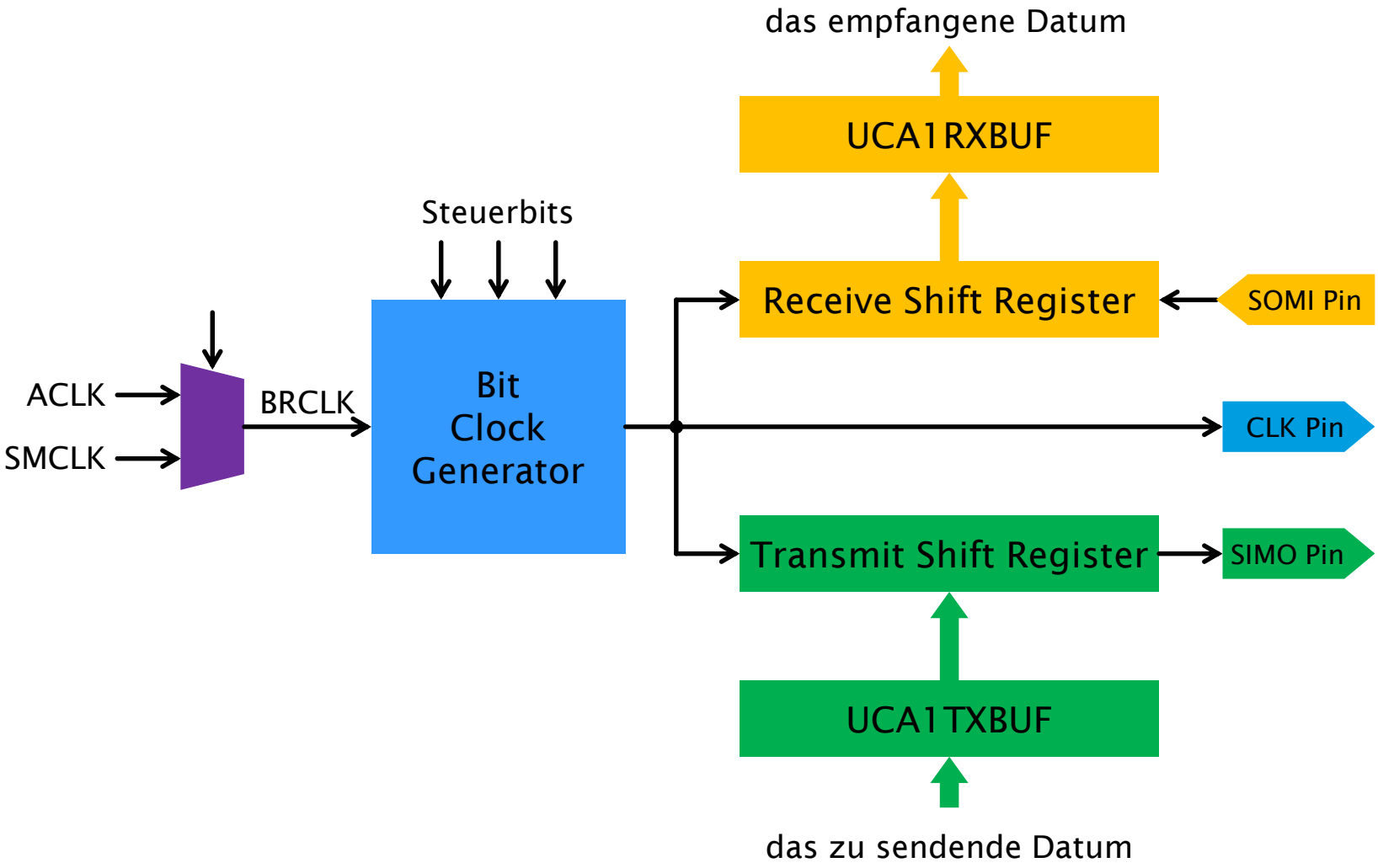


SPI-Timing: AS1108 (4-Digit LED Treiber)



D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	Register Adresse				MSB			Data				LSB

UCA-Aufbau


das empfangene Datum
UCA1RXBUF
Steuerbits
Receive Shift Register
SOMI Pin
ACLK
SMCLK
BRCLK
Bit
Clock
Generator
CLK Pin
Transmit Shift Register
SIMO Pin
UCA1TXBUF
das zu sendende Datum

UCA-Einstellungen

- Die Initialisierung eines eUSCI-Moduls für die synchron-serielle Datenübertragung erfolgt in folgenden Schritten: (am Beispiel der UCA1-Komponente)
  1. Pin 3 am Port 2 (Chip-Select für AS1108) als Output konfigurieren.
  2. Pins 4, 5 und 6 am Port 2 entsprechend als CLK, SIMO und SOMI konfigurieren.
  3. Das Bit UCSWRST (software reset enable) im Steuerregister UCA1CTLW0 wird auf 1 gesetzt. Das führt zu einem Reset-Vorgang im UCA-Modul.
  4. Bei UCSWRST=1 werden restliche Steuerregister des UCA1-Moduls mit geeigneten Werten beladen.
  5. Das Bit UCSWRST im Steuerregister UCA1CTLW0 wird zurückgesetzt.
  6. Sofern erforderlich wird das Interrupt-Enable-Flag UCRXIE im Statusregister UCA1IE auf 1 gesetzt.

Table 19-3. UCAxCTLW0 Register Description

Bit	Field	Type	Reset	Description
15	UCCKPH	RW	0h	Clock phase select 0b = Data is changed on the first UCLK edge and captured on the following edge. 1b = Data is captured on the first UCLK edge and changed on the following edge.
14	UCCKPL	RW	0h	Clock polarity select 0b = The inactive state is low. 1b = The inactive state is high.
13	UCMSB	RW	0h	MSB first select. Controls the direction of the receive and transmit shift register. 0b = LSB first 1b = MSB first
12	UC7BIT	RW	0h	Character length. Selects 7-bit or 8-bit character length. 0b = 8-bit data 1b = 7-bit data
11	UCMST	RW	0h	Master mode select 0b = Slave mode 1b = Master mode
10-9	UCMODEx	RW	0h	eUSCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. 00b = 3-pin SPI 01b = 4-pin SPI with UCxSTE active high: Slave enabled when UCxSTE = 1 10b = 4-pin SPI with UCxSTE active low: Slave enabled when UCxSTE = 0 11b = Reserved
8	UCSYNC	RW	0h	Synchronous mode enable 0b = Asynchronous mode 1b = Synchronous mode
7-6	UCSSELx	RW	0h	eUSCI clock source select. These bits select the BRCLK source clock in master mode. UCxCLK is always used in slave mode. 00b = Reserved 01b = ACLK 10b = SMCLK 11b = SMCLK

Anwendungsbeispiel

- Eine Applikation auf einem MSP430 soll einen freilaufenden Zähler periodisch mit der Frequenz von 1 Hz inkrementieren und seinen Inhalt auf einer vierstelligen Siebensegmentanzeige darstellen.
- Die vierstelligen Siebensegmentanzeige wird mit einem Treiberbaustein AS1108 gesteuert.
- Die Schnittstelle zwischen dem MSP430 und dem AS1108 ist eine P2P-SPI-Verbindung. Laut dem Datenblatt für AS1108 beträgt die SPI-Taktperiode mindestens 100 ns, was einer Taktfrequenz von 10 MHz entspricht. In dieser Applikation soll der SPI-Takt allerdings mit nur ca. 100 kHz laufen.

Anwendungsbeispiel

- Auszug auf der Initialisierungssequenz von UCA

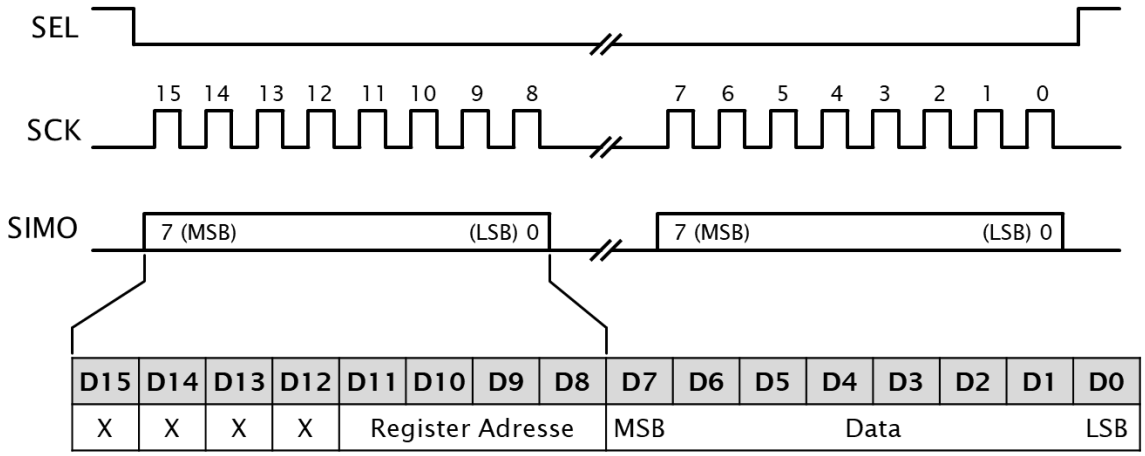
Beispiel 8.1

```
// Port 2: Pin 3          => SPI.CS output, idle High
// Port 2: Pin 4, 5 and 6 => SPI
...

#pragma FUNC_ALWAYS_INLINE(UCA1_init)
GLOBAL Void UCA1_init(Void) {
// set up Universal Serial Communication Interface A
   SETBIT(UCA1CTLW0, UCSWRST);  // UCA1 software reset
   UCA1BRW = 6;                 // prescaler

// in Übereinstimmung mit dem SPI-Timing-Diagramm von AS1108
   UCA1CTLW0 = UCCKPH           // 15: clock phase select: rising edge
             | 0                // 14: clock polarity: inactive low
             | UCMSB            // 13: MSB first
             | 0                // 12: 8-bit data
             | UCMST            // 11: SPI master mode
             | UCMODE_0         // 10-9: mode select: 3-pin SPI
             | UCSYNC           // 8:  synchronous mode enable
             | UCSSEL__ACLK     // 7-6: clock source select
             | 0;               // 0: release the UCA0 for operation

// ggf. weitere lokale Variablen initialisieren

}
```

Anwendungsbeispiel

- Schreibzugriffe auf den AS1108 basierend auf dem Timing

Beispiel 8.1



D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	Register Adresse				MSB	Data						LSB

```
Void UCA1_emit(const UChar adr, const UChar val) {
   UChar ch = UCA1RXBUF;          // RXBUF auslesen, UCRXIFG := 0, UCOE := 0
   CLRBIT(P2OUT, BIT3);           // Select aktivieren
   UCA1TXBUF = adr;               // Ausgabe einer Registeradresse

   while (NOT TSTBIT(UCA1IFG, UCRXIFG)) ;

   ch = UCA1RXBUF;
   UCA1TXBUF = val;               // Ausgabe eines Datums

   while (NOT TSTBIT(UCA1IFG, UCRXIFG)) ;

   ch = UCA1RXBUF;
   SETBIT(P2OUT, BIT3);           // Select deaktivieren
}
```

Anwendungsbeispiel

- Initialisierungssequenz für den AS1108

Beispiel 8.1

```
#define INITSIZE 10

typedef struct {
   UChar adr;
   UChar val;
} TFrame;

LOCAL const TFrame init[INITSIZE] = {
   { 0x0E, 0x0C },  // internal oscillator, enable B/HEX decoding, enable SPI
   { 0x0C, 0x81 },  // shutdown register := normal mode
   { 0x0F, 0x00 },  // normal mode
   { 0x01, 0x00 },
   { 0x02, 0x00 },
   { 0x03, 0x00 },
   { 0x04, 0x00 },
   { 0x09, 0xFF },  //
   { 0x0A, 0x03 },  // intensity 7/32
   { 0x0B, 0x03 }   // display all numbers
};

...

UInt i;
for(i=0; i LT INITSIZE; i++) {
   UCA1_emit(init[i].adr, init[i].val);
}
```

Anwendungsbeispiel

- Verarbeitung der Daten in einer Timer-ISR

Beispiel 8.1

```
LOCAL UInt counter;

#pragma vector = TIMER0_A1_VECTOR
__interrupt Void TIMER0_A1(Void) {
   UInt i;
   UInt tmp = counter;
   CLRBIT(TA0CTL, TAIFG); // clear interrupt flag
   for(i=1; i LE 4; i++) {
      UChar ch = 0x0F BAND tmp;
      ch += '0';
      UCA1_emit(i, ch);
      tmp >>= 4;
   }
   counter += 1;
}
```

- Durch die for-Schleife und die beiden while-Schleifen in der UCA1_emit-Funktion blockiert diese ISR für 632 us andere (auch höher priorisierte) ISR sowie andere Handler.

Anwendungsbeispiel

Beispiel 8.2

- Der Timer inkrementiert einen Zähler und erzeugt Events, die an einen Handler (in der main-Funktion) delegiert werden.
- Der Handler reagiert auf diese Evens, formatiert die Ausgabe und schreibt die Daten über die SPI-Schnittstelle in den Registersatz von AS1108 rein.

```
GLOBAL UInt counter;
...

#pragma vector = TIMER0_A1_VECTOR
__interrupt Void TIMER0_A1(Void) {
   CLRBIT(TA0CTL, TAIFG);
   counter += 1;
   Event_set(EVENT_UPDATE);
   __low_power_mode_off_on_exit();
}
```

```
GLOBAL Void main(Void) {
   CS_init();
   GPIO_init();
   Event_init();
   UCA1_init();
   TA0_init();

   while(TRUE) {
      Event_wait();
      Handler1();
      if (Event_err()) {
         SETBIT(P1OUT, BIT2);
      }
   }
}
```

```
LOCAL Void Handler1(Void) {
   UInt i;
   if (Event_tst(EVENT_UPDATE)) {
      Event_clr(EVENT_UPDATE);
      UInt tmp = counter;
      for(i=1; i LE 4; i++) {
         UChar ch = 0x0F BAND tmp;
         ch += '0';
         UCA1_emit(i, ch);
         tmp >>= 4;
      }
   }
}
```

- Die ISR selbst ist nicht mehr blockierend, aber der Handler!

Anwendungsbeispiel

- Die while-Schleifen aus der Schreibfunktion müssen durch eine Interrupt-gesteuerte Ausgabe implementiert werden.

Beispiel 8.3

```
#define DATASIZE 2
LOCAL UInt  idx;              // Index
LOCAL UChar data[DATASIZE];   // Datenfeld

GLOBAL Void UCA1_emit(const UChar adr, const UChar val) {
   idx  = 0;
   data[0] = adr;
   data[1] = val;
   SETBIT(UCA1IFG, UCRXIFG); // indirekter Aufruf der ISR
}

#pragma vector = USCI_A1_VECTOR
__interrupt Void UCA1_ISR(Void) {
   UChar ch = UCA1RXBUF;          // RXBUF auslesen, UCRXIFG := 0, UCOE := 0
   CLRBIT(P2OUT, BIT3);           // Select aktivieren
   if (idx GE DATASIZE) {
      SETBIT(P2OUT,  BIT3);       // Select deaktivieren
      Event_set(EVENT_DONE);      // Event senden
      __low_power_mode_off_on_exit();
   } else {
      UCA1TXBUF = data[idx++];    // nächstes Byte ausgeben
   }
}
```

Anwendungsbeispiel

- Die for-Schleife aus dem Handler ist mit Hilfe einer Zustandsmaschine zu implementieren.

Beispiel 8.3

```
// Datentyp eines Funktionspointers
typedef Void (* VoidFunc)(Void);

LOCAL Void State0(Void);
LOCAL Void State1(Void);

// lokale Zustandsvariablen
LOCAL VoidFunc state;
LOCAL UInt idx;
LOCAL UInt tmp;

#pragma FUNC_ALWAYS_INLINE(Handler1_init)
GLOBAL Void Handler1_init(Void) {
   state = State0;
}


#pragma FUNC_ALWAYS_INLINE(Handler1)
GLOBAL Void Handler1(Void) {
   (*state)();
}
```

```
LOCAL Void State0(Void) {
   if (Event_tst(EVENT_UPDATE)) {
      Event_clr(EVENT_UPDATE);
      tmp = counter;
      idx = 1;
      state = State1;
      Event_set(EVENT_DONE);
   }
}

LOCAL Void State1(Void) {
   if (Event_tst(EVENT_DONE)) {
      Event_clr(EVENT_DONE);
      if (idx LE 4) {
         UChar ch = 0x0F BAND tmp;
         ch += '0';
         UCA1_emit(idx, ch);
         tmp >>= 4;
         idx++;
      } else {
         state = State0;
      }
   }
}
```