STM32 - Schieberegister, SPI

u23 2013

andy, florob, gordin, ike, meise, tobix, zakx

Chaos Computer Club Cologne e.V.
http://koeln.ccc.de

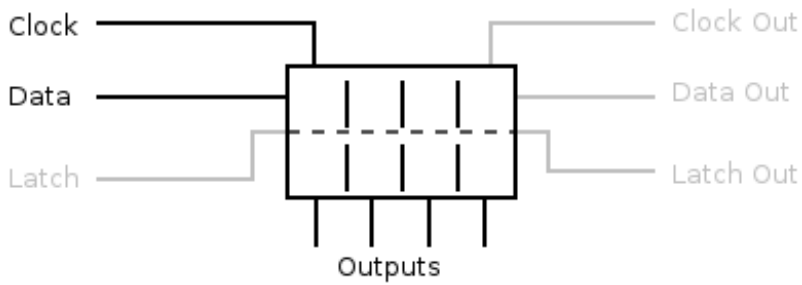
Cologne
2013-11-04

ike <ike@koeln.ccc.de>

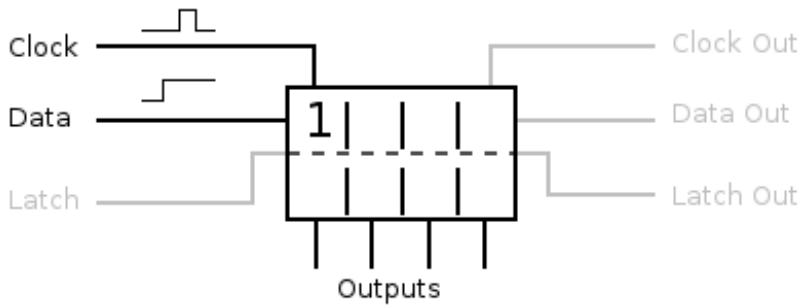
Was ist das?

- Register: Hält Daten
- Schieben: Die Daten werden hinein“geschoben”
- Vorteil: Viele Inputs/Outputs mit wenigen µC-Pins
- Nachteil: Höherer Programmieraufwand / Zeit- / Rechenbedarf

Funktionsweise

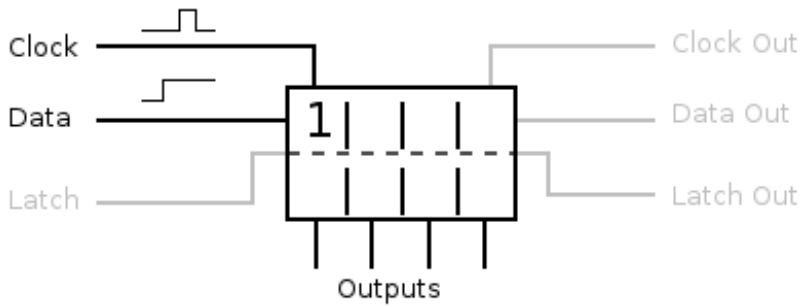


Funktionsweise



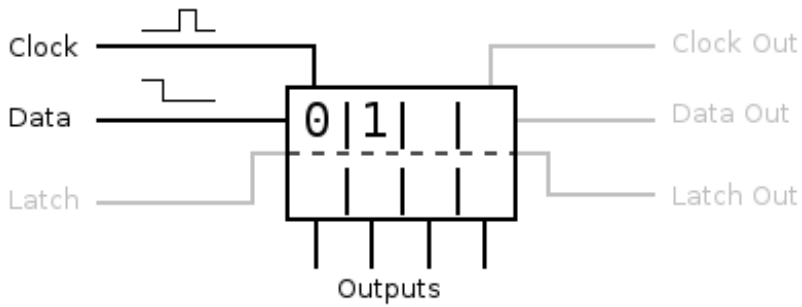
- Data: Eingehende Daten
- Clock: Bei steigender Flanke wird der Wert von Data in das temporäre Register übernommen

Funktionsweise



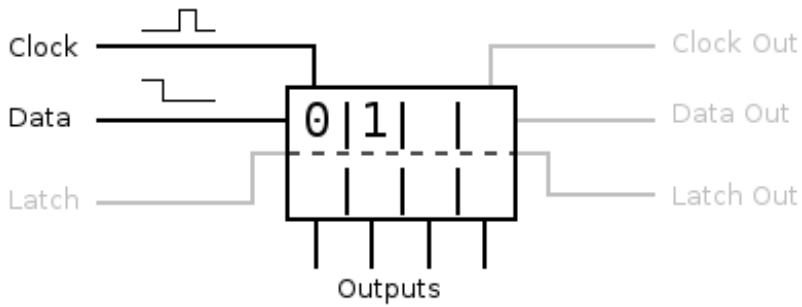
- Data: Eingehende Daten
- Clock: Bei steigender Flanke wird der Wert von Data in das temporäre Register übernommen

Funktionsweise



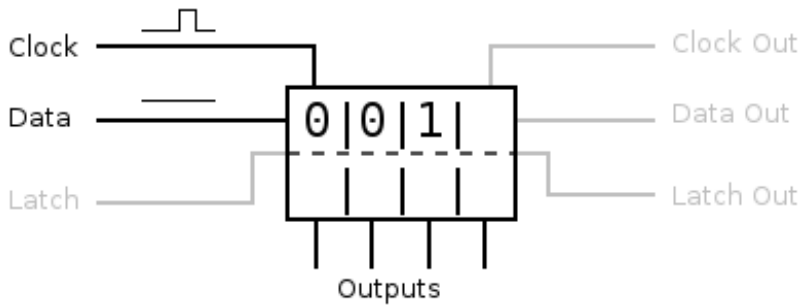
- Die Werte werden immer in das erste Feld übernommen
- Die vorhandenen Werte werden “durchgeschoben”

Funktionsweise



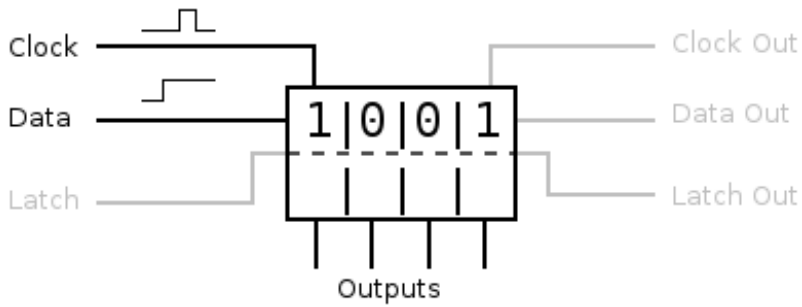
- Die Werte werden immer in das erste Feld übernommen
- Die vorhandenen Werte werden “durchgeschoben”

Funktionsweise



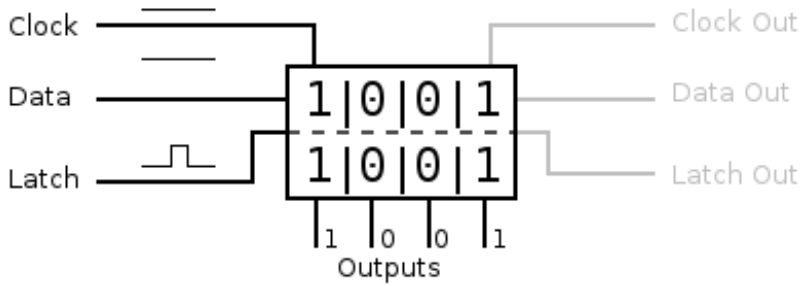
- Die Werte werden immer in das erste Feld übernommen
- Die vorhandenen Werte werden “durchgeschoben”

Funktionsweise



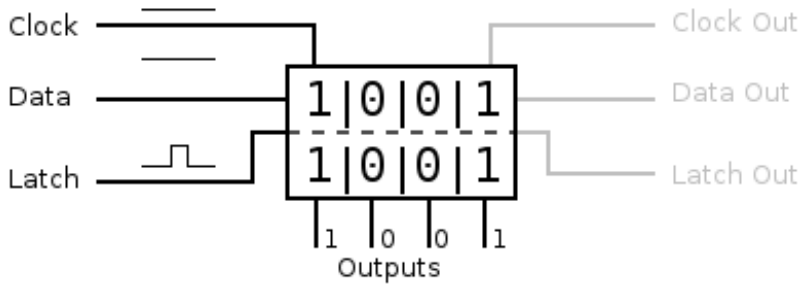
- Die Werte werden immer in das erste Feld übernommen
- Die vorhandenen Werte werden “durchgeschoben”

Funktionsweise



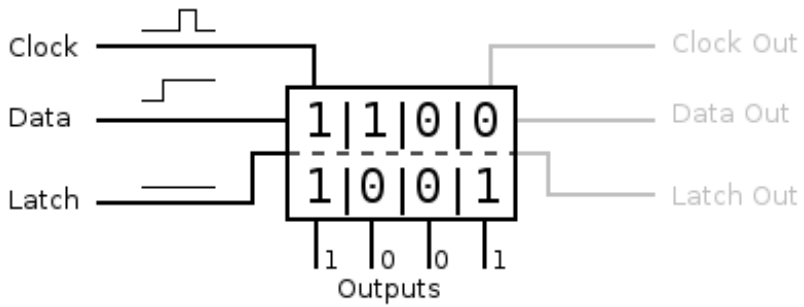
- Latch: Bei steigender Flanke werden die Werte des temporären Registers in das Output-Register übernommen
- Sonst werden diese Werte nie geändert!

Funktionsweise



- Latch: Bei steigender Flanke werden die Werte des temporären Registers in das Output-Register übernommen
- Sonst werden diese Werte nie geändert!

Funktionsweise



- Latch: Bei steigender Flanke werden die Werte des temporären Registers in das Output-Register übernommen
- Sonst werden diese Werte nie geändert!

Bitbanging

- Wie generiert man so ein Signal?
- Eine Möglichkeit: Manuelles Erzeugen mit GPIO's ("Bitbanging")
- Jedes Bit wird einzeln generiert

Bitbanging Shiftbrites - Output

```
// Write a single 32 bit integer out to SPI by bitbanging.
// Also toggles the corresponding LED pins.
void gpiospi_write_int32(uint32_t data)
{
  // Go through the bits, MSB first
  for (int i=31; i>=0; i--)
  {
    // Get current bit
    uint32_t bit = data & (0x1<<i);

    // Set Data
    GPIO_WriteBit(GPIOB, GPIO_Pin_15, bit ? SET : RESET); // Data
    GPIO_WriteBit(GPIOD, GPIO_Pin_15, bit ? SET : RESET); // LED
    wait_own(wait_data);

    // Set clock high
    GPIO_WriteBit(GPIOB, GPIO_Pin_13, SET); // Clock
    GPIO_WriteBit(GPIOD, GPIO_Pin_13, SET); // LED
    wait_own(wait_data);

    // Set clock low
    GPIO_WriteBit(GPIOB, GPIO_Pin_13, RESET); // Clock
    GPIO_WriteBit(GPIOD, GPIO_Pin_13, RESET); // LED
    wait_own(wait_data);
  }
}
```

Bitbanging Shiftbrites - Latch

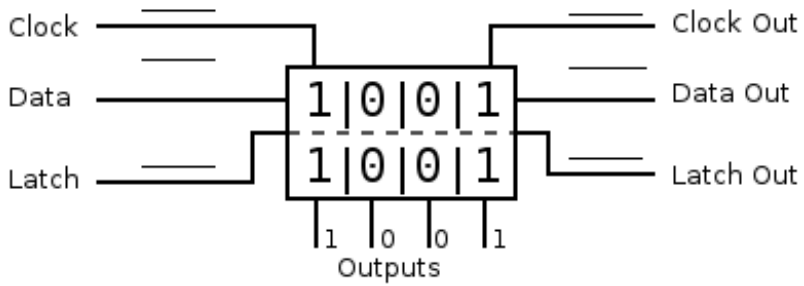
```
// Sets the latch pin (PIN 12) to high and then to low.
// At the same time also toggle the corresponding LED pins.
void latch(void)
{
  // Latch High
  GPIO_WriteBit(GPIOB, GPIO_Pin_12, SET); // Latch
  GPIO_WriteBit(GPIOD, GPIO_Pin_12, SET); // LED
  wait_own(wait_latch);

  // Latch Low
  GPIO_WriteBit(GPIOB, GPIO_Pin_12, RESET); // Latch
  GPIO_WriteBit(GPIOD, GPIO_Pin_12, RESET); // LED
  wait_own(wait_latch);
}
```

Bitbanging Shiftbrites

- Der Code ist synchron
- Das heißt, die CPU ist die ganze Zeit damit beschäftigt die Daten rauszupusten
- Das dauert lange und kostet Rechenzeit
- Bitbanging ist die einfachste, aber nicht die Beste Lösung
- (siehe später)

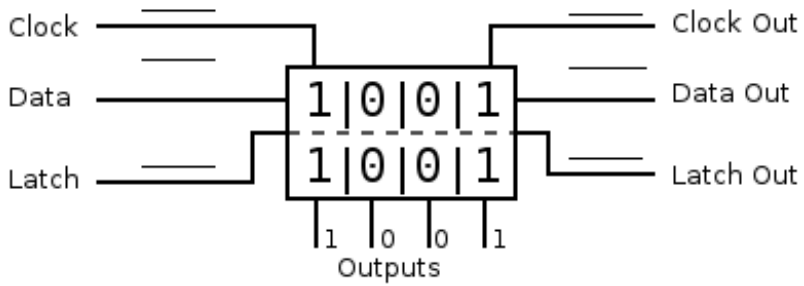
Logic Analyzer

Jetzt wird's lustig



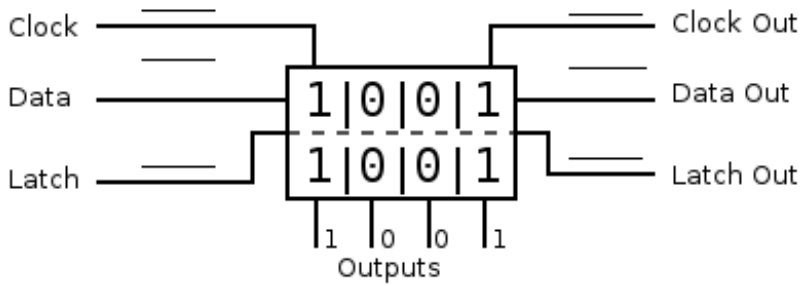
- Schieberegister haben noch mehr Funktionen:
  - Data Out: Hier liegt der Wert des hintersten Datenbits an
  - Clock Out: Das Clock Signal wird neu produziert / durchgeschleift
  - Latch Out: Das Latch Signal wird neu produziert / durchgeschleift
- Data Out und Latch Out sind nicht immer vorhanden

Jetzt wird's lustig



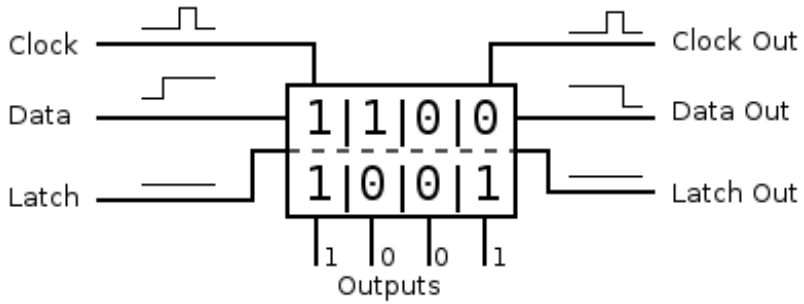
- Schieberegister haben noch mehr Funktionen:
  - Data Out: Hier liegt der Wert des hintersten Datenbits an
  - Clock Out: Das Clock Signal wird neu produziert / durchgeschleift
  - Latch Out: Das Latch Signal wird neu produziert / durchgeschleift
- Data Out und Latch Out sind nicht immer vorhanden

Jetzt wird's lustig



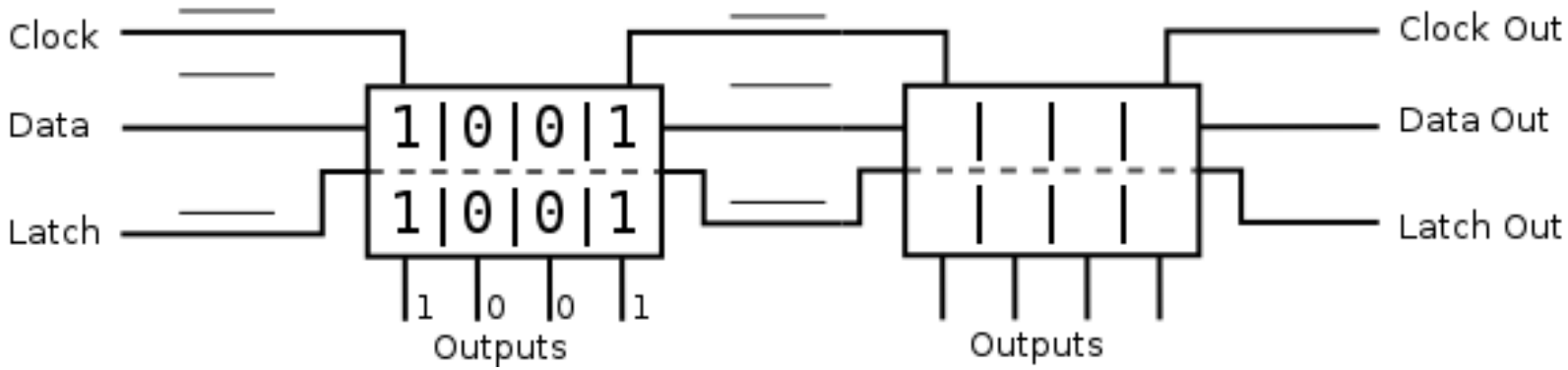
- Schieberegister haben noch mehr Funktionen:
  - Data Out: Hier liegt der Wert des hintersten Datenbits an
  - Clock Out: Das Clock Signal wird neu produziert / durchgeschleift
  - Latch Out: Das Latch Signal wird neu produziert / durchgeschleift
- Data Out und Latch Out sind nicht immer vorhanden

Jetzt wird's lustig



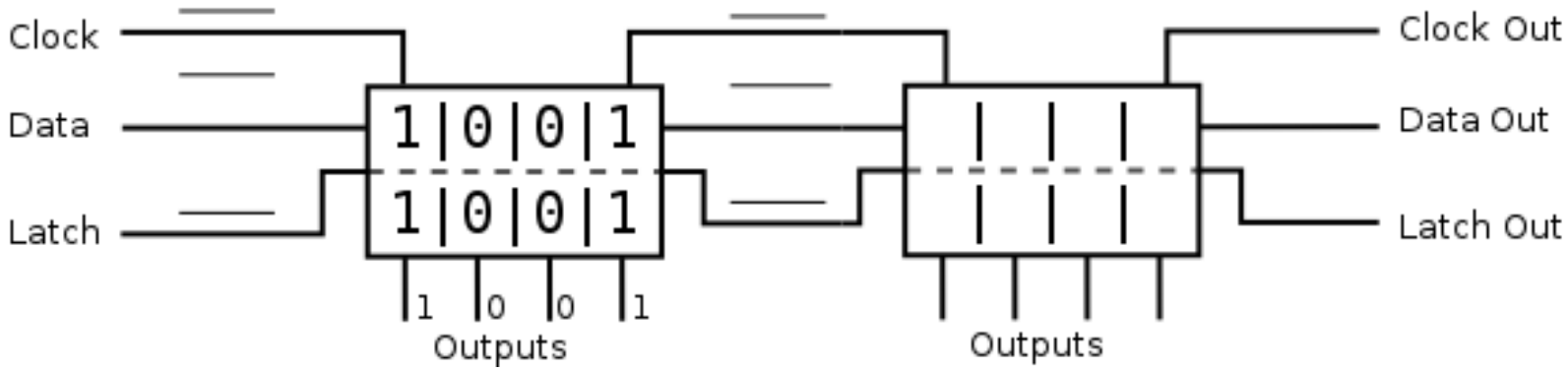
- Schieberegister haben noch mehr Funktionen:
  - Data Out: Hier liegt der Wert des hintersten Datenbits an
  - Clock Out: Das Clock Signal wird neu produziert / durchgeschleift
  - Latch Out: Das Latch Signal wird neu produziert / durchgeschleift
- Data Out und Latch Out sind nicht immer vorhanden

Chaining



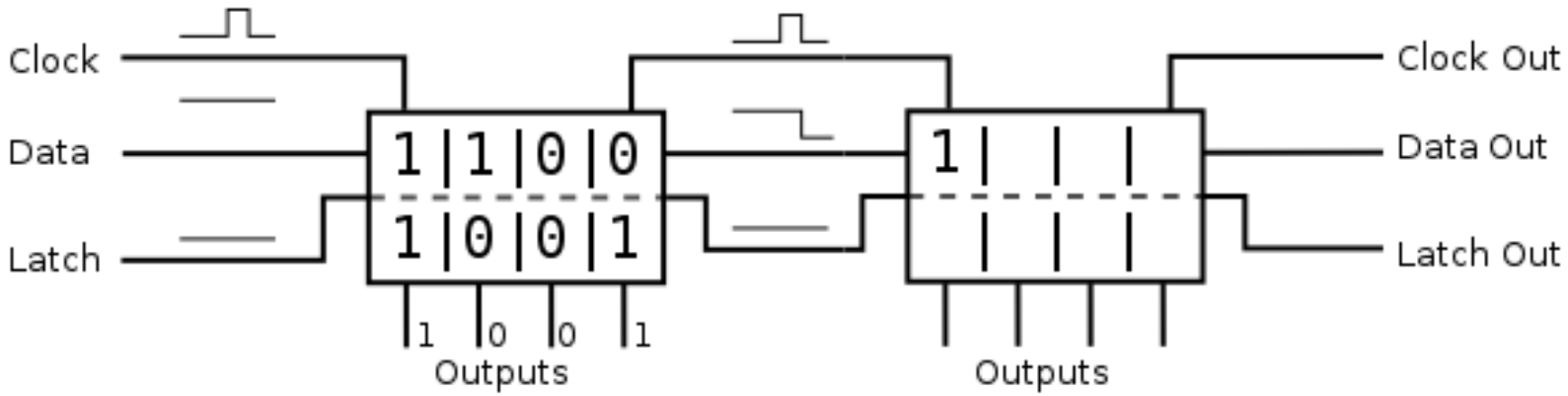
- Man kann so mehrere Schieberegister hinter einander hängen
- und braucht trotzdem nur 3 Pins

Chaining



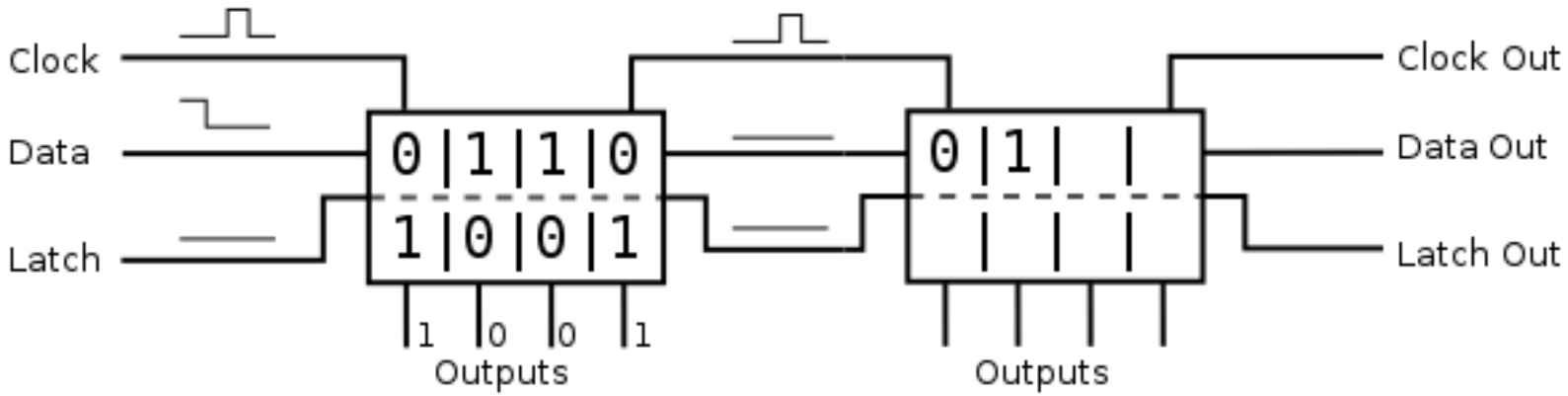
- Man kann so mehrere Schieberegister hinter einander hängen
- und braucht trotzdem nur 3 Pins

Chaining



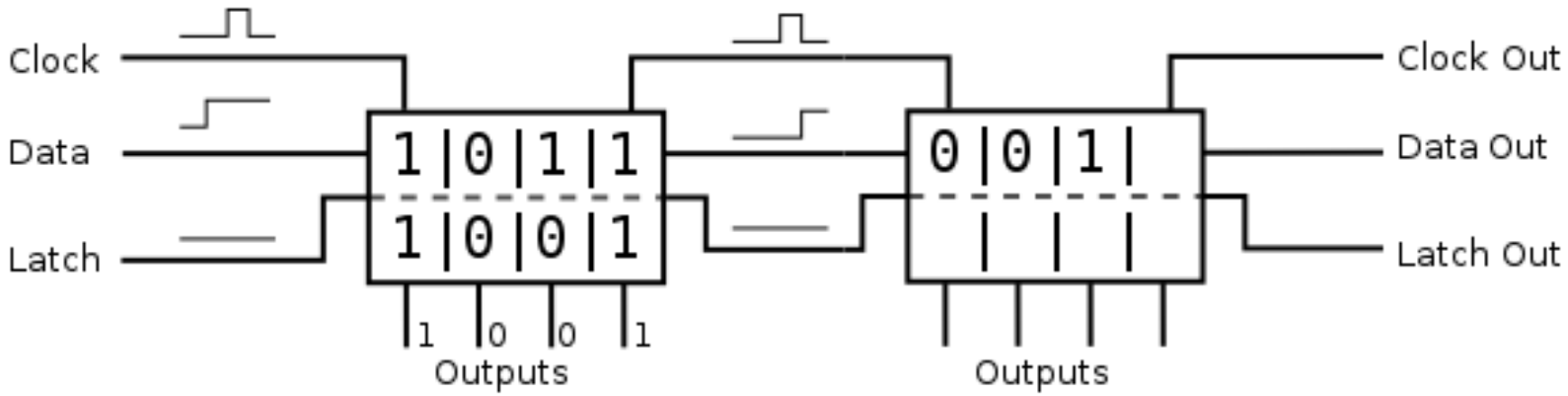
- Man kann so mehrere Schieberegister hinter einander hängen
- und braucht trotzdem nur 3 Pins

Chaining



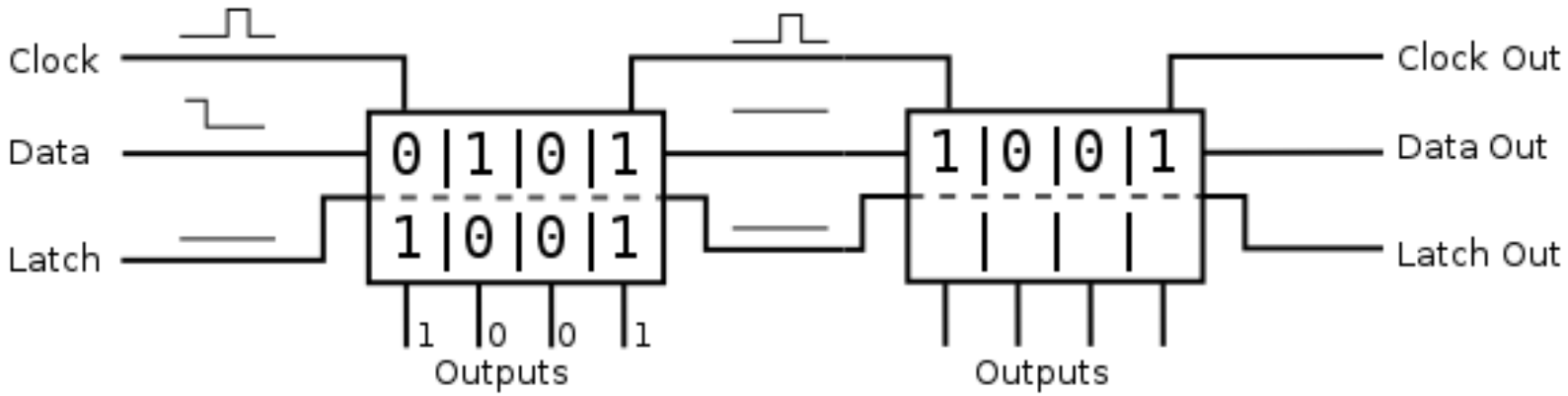
- Man kann so mehrere Schieberegister hinter einander hängen
- und braucht trotzdem nur 3 Pins

Chaining



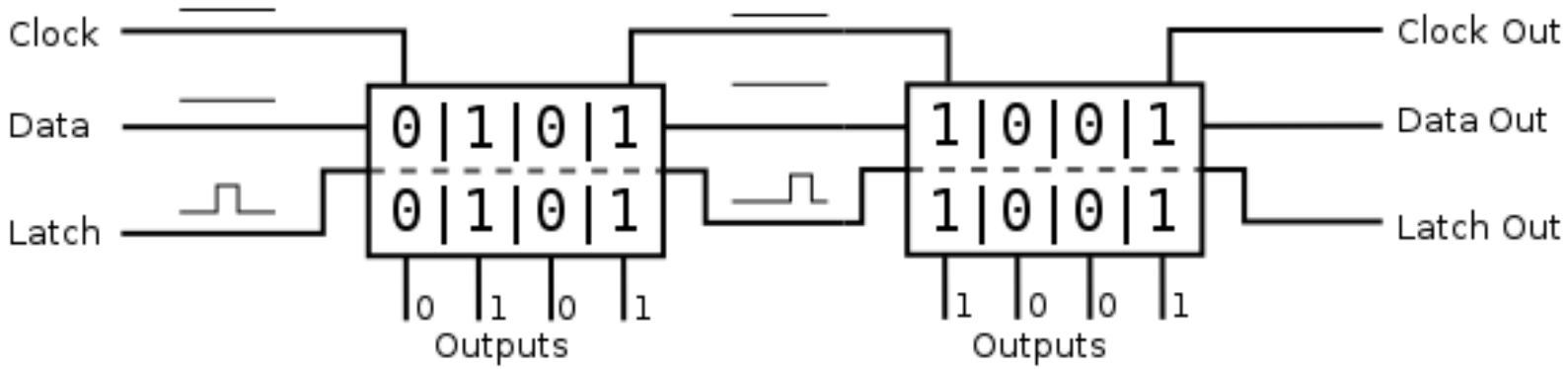
- Man kann so mehrere Schieberegister hinter einander hängen
- und braucht trotzdem nur 3 Pins

Chaining



- Man kann so mehrere Schieberegister hinter einander hängen
- und braucht trotzdem nur 3 Pins

Chaining



- Man kann so mehrere Schieberegister hinter einander hängen
- und braucht trotzdem nur 3 Pins

Was ist das?

- SPI = **S**erial **P**eripheral **I**nterface
- Ein low-level Datenübertragunsprotokoll
- Seriell: Daten(bits) werden seriell (nacheinander) übertragen

Protokoll

- Ähnlich zu Schieberegistern, allerdings:
- Es gibt keinen Latch, die Übertragung ist nach fester Bitlänge abgeschlossen
- Der Master gibt dem Slave ein Signal, dass dieser Slave angesprochen wird und die Übertragung beginnt. (sog. Chip Select)
- Es gibt einen Rückkanal
- Es wird immer gleichzeitig empfangen und gesendet
- Will man nur lesen, sendet man zB 0x00

Pins

- CLK: Clock
- MOSI: Master Out Slave In
- MISO: Master In Slave Out
- CS: Chip Select

Initialisierung - Pins

```
// SPI (SCK PB13, MISO PB14, MOSI PB15)
GPIO_Init(GPIOB, &(GPIO_InitTypeDef){
    .GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15,
    .GPIO_Mode = GPIO_Mode_AF,
    .GPIO_Speed = GPIO_Speed_50MHz,
  });

// Configure pins to be used by the SPI hardware (alternate function)
GPIO_PinAFConfig(GPIOB, GPIO_PinSource13, GPIO_AF_SPI2);
GPIO_PinAFConfig(GPIOB, GPIO_PinSource14, GPIO_AF_SPI2);
GPIO_PinAFConfig(GPIOB, GPIO_PinSource15, GPIO_AF_SPI2);
```

Initialisierung - SPI

```
// Init SPI
SPI_Init(SPI2, &(SPI_InitTypeDef){
    .SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_64, // Configure Data speed
    .SPI_CPHA = SPI_CPHA_1Edge, // Sample data on rising edge
    .SPI_CPOL = SPI_CPOL_Low, // Clock is default low
    .SPI_CRCPolynomial = 1, // Don't use CRC
    .SPI_DataSize = SPI_DataSize_8b, // Send 8 bit at a time
    .SPI_Direction = SPI_Direction_2Lines_FullDuplex, // Enable Sending / Receiving
    .SPI_FirstBit = SPI_FirstBit_MSB, // Most Significant Bit first
    .SPI_Mode = SPI_Mode_Master, // STM32 is the master
    .SPI_NSS = SPI_NSS_Soft, // Don't use automatic chip select
  });

// Enable SPI hardware
SPI_Cmd(SPI2, ENABLE);
```

Initialisierung - Interrupts

```
// Enable SPI interrupt
NVIC_Init(&(NVIC_InitTypeDef){
    .NVIC_IRQChannel = SPI2_IRQn,
    .NVIC_IRQChannelPreemptionPriority = 0,
    .NVIC_IRQChannelSubPriority = 0,
    .NVIC_IRQChannelCmd = ENABLE,
  });
```

Daten Senden - Non-Buffered

```
// Enable RXNE interrupt
SPI_I2S_ITConfig(SPI2, SPI_I2S_IT_RXNE, ENABLE);

// Start Data transfer
SPI_I2S_SendData(SPI2, data);
```

- Die SPI Hardware läuft nun Asynchron
- In der Zeit kann und wird die CPU etwas anderes tun (und wenn es nur warten ist)
- Zum Beispiel: Warten bis ein Byte komplett empfangen wurde

```
while (SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_RXNE) == SET);
```

Daten Senden - Interrupts

- Besser mit Interrupts:
- Wenn die Übertragung vorbei ist und ein Byte empfangen wurde wird der RXNE (Receive Buffer Not Empty) interrupt ausgelöst

```
void SPI2_IRQHandler(void) {
  // Check for interrupt cause
  if(SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_RXNE) == SET)
  {
    data = SPI_I2S_ReceiveData(SPI2);
  }
}
```

- Der Interrupt kann verschiedene Gründe haben, deswegen müssen wir prüfen ob es wirklich RXNE war

Daten Senden - Buffered

- Wenn man mehr als ein Byte senden will, ist es übersichtlicher Buffer zu benutzen:

```
// Data structures
uint8_t buffer[3];
int RX_counter = 0;
int TX_counter = 0;

// Start transfer
SPI_I2S_ITConfig(SPI2, SPI_I2S_IT_RXNE, ENABLE);
SPI_I2S_ITConfig(SPI2, SPI_I2S_IT_TXE, ENABLE);
```

- Wir benutzen einen weiteren SPI-Interrupt: TXE (Transmit Buffer Empty)
- Dieser wird aufgerufen sobald der interne Buffer der SPI leer ist
- In unserem Fall (SPI läuft nicht) passiert das sofort

Daten Senden - Interrupts

- Wird TXE ausgelöst schreiben wir ein neues Byte
- Wenn wir alle geschrieben haben, deaktivieren wir TXE-Int
- Die gelesenen Bytes schreiben wir zurück in den Buffer (bei RXNE)

```
void SPI2_IRQHandler(void) {
  // Check for interrupt cause
  if(SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_RXNE) == SET)
  {
    buffer[RX_counter++] = SPI_I2S_ReceiveData(SPI2);
    if (RX_counter >= 3)
      SPI_I2S_ITConfig(SPI2, SPI_I2S_IT_RXNE, DISABLE);
  }

  else if(SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_TXE) == SET)
  {
    SPI_I2S_SendData(SPI2, buffer[TX_counter++]);
    if (TX_counter >= 3)
      SPI_I2S_ITConfig(SPI2, SPI_I2S_IT_TXE, DISABLE);
  }
}
```

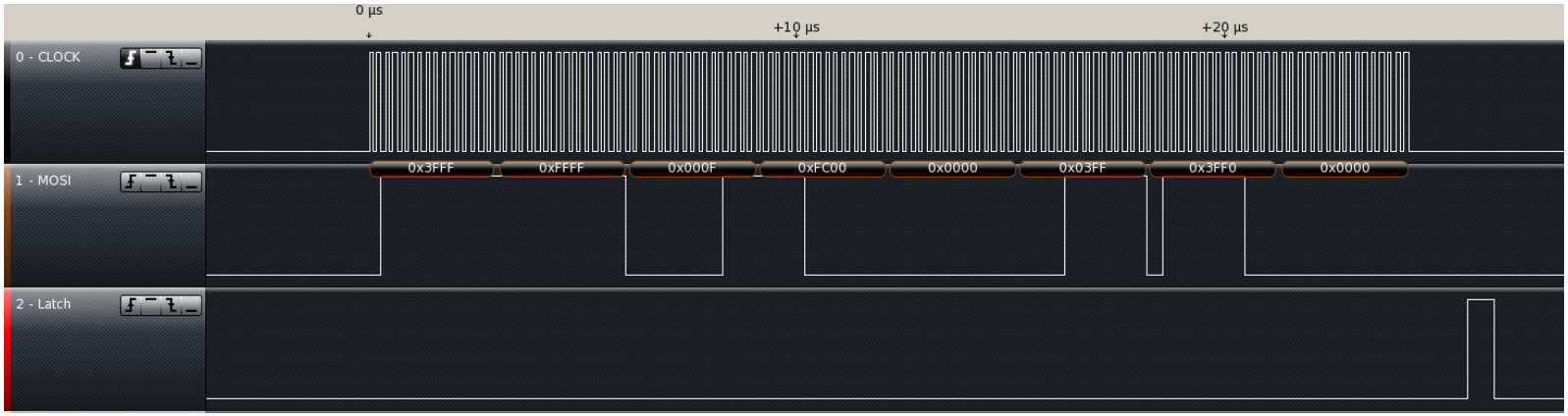
Schieberegister per SPI

- SPI sieht fast so aus wie das Schieberegisterprotokoll
- Es fehlt nur der Latch
- Wir können Schieberegister also besser bedienen
- Der Latch muss allerdings ausgelöst werden, nachdem alle Daten geschrieben sind
- Wir müssen also warten bis die SPI fertig ist
- Dafür gibt es keinen Interrupt
- Man kann aber RXNE benutzen oder manuell warten
- Man sollte aber aufpassen mit dem Warten während eines Interrupts

Latch

```
void SPI2_IRQHandler(void) {
  // Check for interrupt cause
  if(SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_RXNE) == SET)
  {
    buffer[RX_counter++] = SPI_I2S_ReceiveData(SPI2);
    if (RX_counter >= 3)
    {
      while (SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_BSY) == SET);
      SPI_I2S_ITConfig(SPI2, SPI_I2S_IT_RXNE, DISABLE);
      latch();
    }
  }
  // [...]
}
```

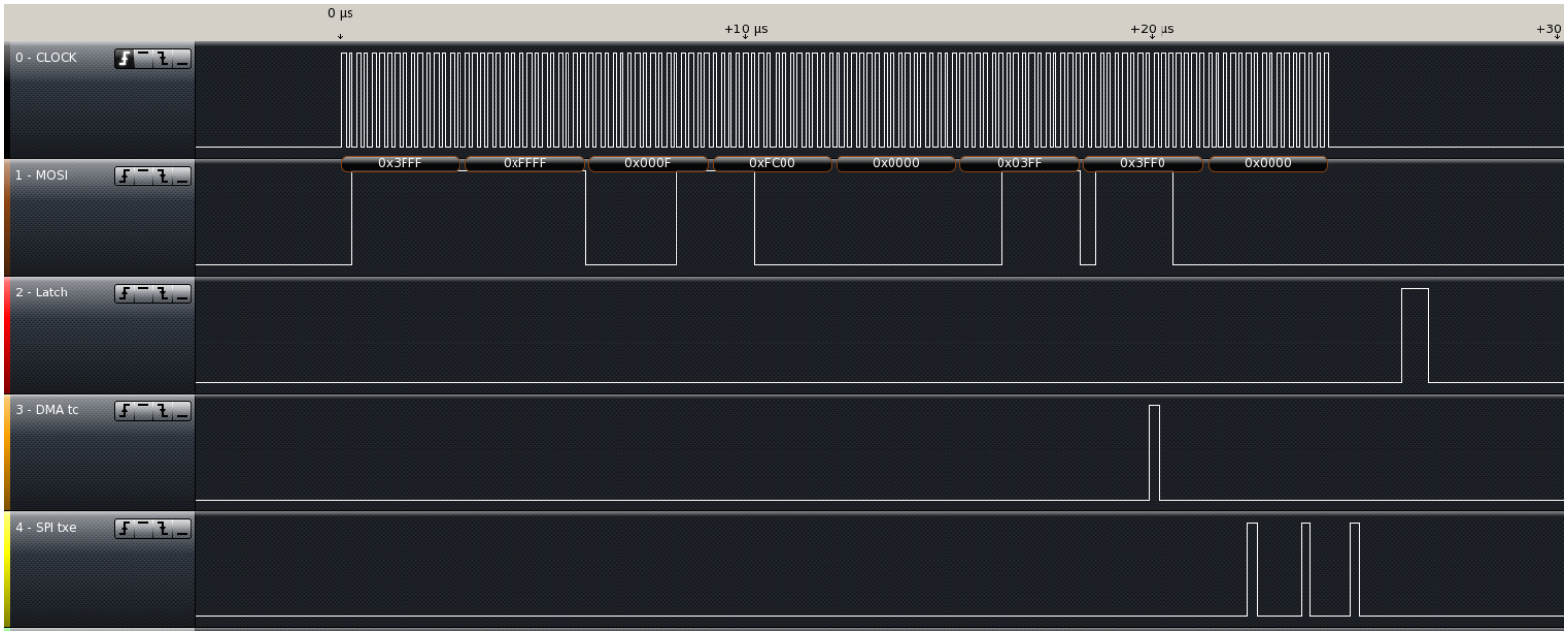
Logic Analyzer



Ein Wort zu SPI Interrupts

- TXE
  - Wird ausgelöst wenn der Transmit Buffer leer ist
  - Die SPI Hardware hat einen internen Buffer (8/16 bits)
  - Und einen Schiebebuffer, aus dem die Daten herausgeschoben werden
  - Das heißt die ersten beiden TXE-Ints kommen sehr schnell
  - Und: Bei längerer Übertragung tritt der letzte TXE-Int beim vorletzten Byte auf
- RXNE
  - Tritt Nach jedem gelesenen Byte auf
  - Liest man den RX-Buffer nicht aus, wird der Interrupt nochmal gefeuert!
  - Wird der RXNE-Int blockiert (zB anderer Int), können Daten verloren gehen

Logic Analyzer



SPI-Varianten

- Es gibt viele Varianten von SPI
- Man kann vieles einstellen
- Siehe Datenblätter von Bauteilen
- und STM-Doku zur SPI-Initialisierung

SPI vs Shiftreg

- Schieberegister brauchen nur 3 Pins, dafür Prozessorzeit
- SPI-Bausteine sind schneller zu bedienen, verbrauchen aber n+2 Pins
- Wahlweise kann man den Chip-Select mit Shiftregs bauen :)
  $\rightarrow$ 5 Pins

Beispiele

- Es gibt Beispiele
- libs/libsystem/{inc,src}/Spi.{c,h}
- bare_metal/0*_spi_*
- Shiftbriteansteuerung per Bitbanging, SPI, und SPI+DMA
- DMA ist ganz abgefahrenes Zeug, braucht ihr nicht
- Wenn doch / ihr Interesse habt / Masochisten seid: fragt uns

Beispiele

- Es gibt Beispiele
- libs/libsystem/{inc,src}/Spi.{c,h}
- bare_metal/0*_spi_*
- Shiftbriteansteuerung per Bitbanging, SPI, und SPI+DMA
- DMA ist ganz abgefahrenes Zeug, braucht ihr nicht
- Wenn doch / ihr Interesse habt / Masochisten seid: fragt uns