Ouput-Pin

Logical Output Value	Output Signal		
	Open Drain No Pull	Open Drain + Pull Up	Push-Pull
0	Low	Low	Low
1	Z	High	High

Input-Pin

Input Signal	Logical Input		
	No Pull (Floating)	Pull Down	Pull Up
high	1	1	1
floating (Z)	?	0	1
low	0	0	0

GPIO cookbook for basic input / output configuration

- Initialization
  - Find pin numbers related to GPIOx or vice versa
  - Configure through configuration registers
    - ▸ GPIOx_MODER
    - ▸ GPIOx_OTYPER
    - ▸ GPIOx_OSPEEDR
    - ▸ GPIOx_PUPDR
- Use
  - Read / write data using data registers
    - ▸ GPIOx_IDR / GPIOx_ODR
      or
    - ▸ GPIOx_BSRR

Control registers
32-bit memory-mapped control registers for each GPIO
Configure up to 16 I/Os
GPIOx_MODER RM 8.4.1, pg. 278
- I/O direction (input, output, alternate function, analog)
GPIOx_OTYPER RM 8.4.2, pg. 279
- output type (push-pull or open-drain)
GPIOx_OSPEEDR RM 8.4.3, pg. 279
- speed (the I/O speed pins are directly connected to the corresponding GPIOx_OSPEEDR register bits whatever the I/O direction)
GPIOx_PUPDR RM 8.4.4, pg. 280
- pull-up/pull-down independent of programmed I/O direction

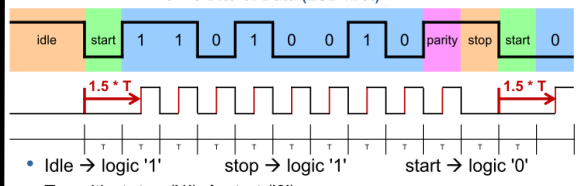
UART Characteristics

Synchronization: Each data item (5-8 bits) requires synchronization
Asynchronous data transfer: Mismatch of clock frequencies in TX and RX, Requires overhead for synchronization -> additional bits, Requires effort for synchronization ⍰ additional hardware **Advantage:** Clock does not have to be transmitted, Transmission delays are automatically compensated
On-board connections: Signal levels are 3V or 5V with reference to ground, Off-board connections require stronger output drivers (circuits)

UART Timing

e.g. 9'600 Baud = symbols / s
➔ T = (1 / 9600) s = 0.104 ms

5 – 8 Bits of Data (LSB first)



- Idle → logic '1' stop → logic '1' start → logic '0'
- Transition stop ('1') → start ('0')
  - Receiver detects edge at the start of each data block (5 to 8 bits)
  - Allows receiver to sample data "in middle of bits" → red edges
  - Clocks have to be accurate enough to allow sampling up to parity bit, i.e. max. +/- 0.5 bit times aggregated deviation until last bit

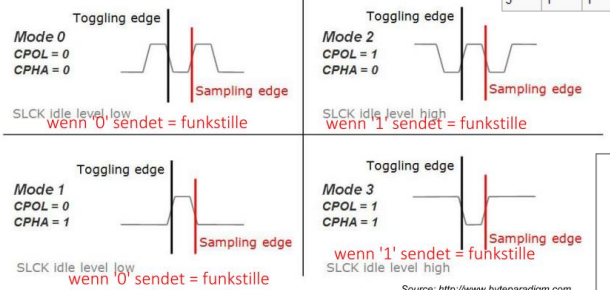
SPI – Serial Peripheral Interface

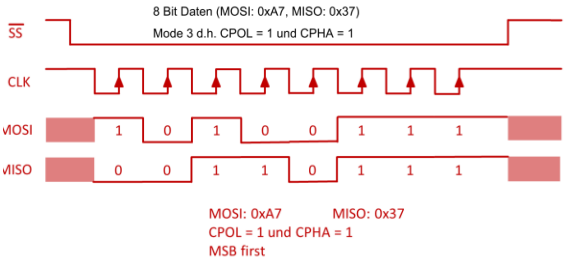
Serial bus for on-board connections: Used for short distance comm.
Connects microcontroller and external devices: Sensors, A/D converters, displays, flash memories, codecs, IOs, Real Time Clocks (RTC), wireless transceivers **Synchronous:** Master distributes the clock to slaves
Compared to a parallel bus: Saves board area Lowers pin count on both chips (TX and RX) ⍰ smaller, low-cost, Simplifies EMC (Electromagnetic compatibility)

Clock Polarity and Clock Phase

- TX provides data on 'Toggling Edge'
- RX takes over data with 'Sampling Edge'

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

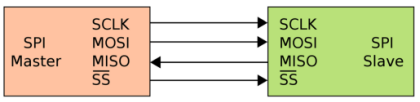




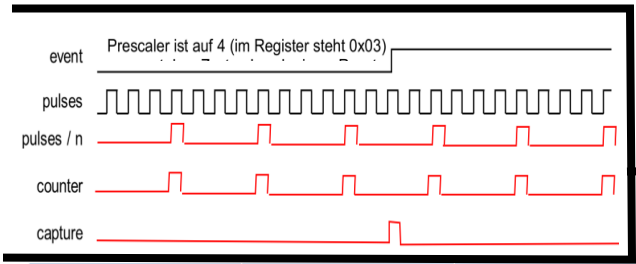
Register	Funktion(en)	Inhalt
Prescaler	Es wird nur jeder n-te Wert gezählt.	Divisor für Eingangssignal
Counter	Der Wert wird mit jedem n-ten Tick um eins erhöht oder erniedrigt	Aktueller Timerwert
Reload	Upcounter: Timer zählt bis zu diesem Wert, dann Überlauf Downcounter: Startwert für Timer	Wert für Timer-überlauf
Capture/Compare	Capture: Bei einem Event wird der Wert des Counters hier gespeichert Compare: Sobald der Wert vom Counter erreicht wird, wird ein Event ausgelöst.	Vergleichswert

Synchronous Serial Data Connection

- Full duplex



- Master (single master)
  - Generates clock (SCLK)
  - Initiates data transfer by setting $\overline{SS} = 0$ (Slave Select)
- MOSI – Master Out Slave In
  - Data from master to slave
- MISO – Master In Slave Out
  - Data from slave to master



Welche Taktabweichung in % von der Bit-Zeit (T) darf der Empfänger maximal aufweisen, damit die Zeichen noch fehlerfrei empfangen werden können, falls der Sender mit exakter Frequenz läuft?

fallende Flanke Start-Bit bis Mitte D6 = 7.5 Bits
Maximale Abweichung für die richtige Erkennung D6: 0.5 bits
Taktabweichung: 100% * 0.5 / 7.5 = ~ 6%

Wie viele Nutzdaten-Bytes kann man pro Sekunde übertragen, wenn die UART eingestellt ist auf 9600 baud (entspricht hier 9600 bit/Sek.), 8 Datenbits, 2 Stoppbits, 1 Paritybit.

Pro Nutzdate-Byte werden 1 Startbit, 8 Datenbits, 2 Stoppbits und 1 Paritybit, d.h. total 12 Bits übertragen

UART	SPI	I2C
serial ports (RS-232)	4-wire bus	2-wire bus
TX, RX opt. control signals	*MOSI, MISO, SCLK, SS*	*SCL, SDA*
point-to-point	point-to-multipoint	(multi-) point-to-multi-point
full-duplex	full-duplex	half-duplex
asynchronous	synchronous	synchronous
only higher layer addressing	slave selection through $\overline{SS}$ signal	7/10-bit slave address
parity bit possible	no error detection	no error detection
chip-to-chip, PC terminal program	chip-to-chip, on-board connections	chip-to-chip, board-to-board connections

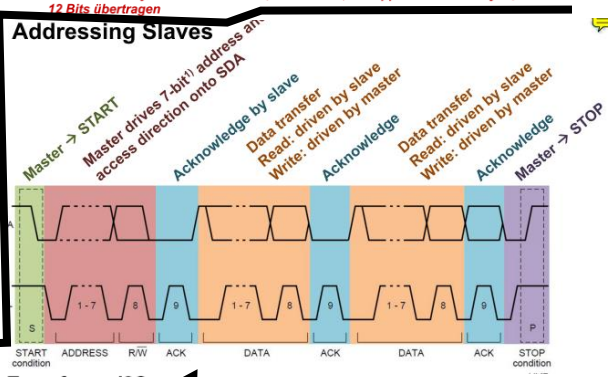
The three interfaces provide the lowest layer of communication and require higher level protocols to provide and interpret the transferred data.

SPI
- Four modes → clock polarity and clock phase
- No acknowledge, no error detection
- Selection of device through Slave Select ($\overline{SS}$)
- Synchronous full-duplex transmission (MOSI, MISO)
- Master/Slave
- Longer connections require line drivers → RS-232/RS-485

I2C
Synchronous half-duplex transmission (SCL, SDA), 7-bit slave addresses

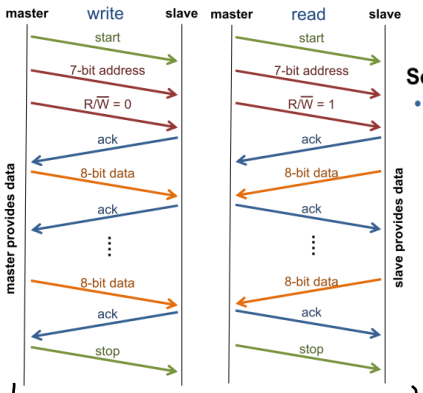
Asynchronous Serial Interface
- Transmitter and receiver use diverging clocks → overhead
- Synchronization using start/stop bits

Addressing Slaves



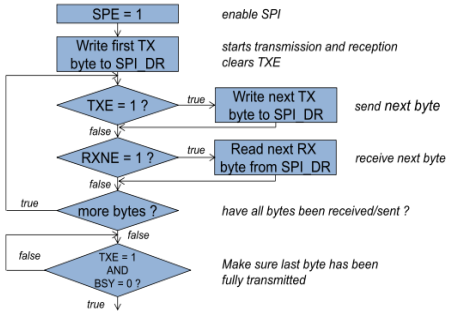
Data Transfer on I2C

- 8-bit oriented transfers
- Bit 9: Receiver acknowledges by driving SDA low
- Master defines number of 8-bit transfers (STOP)



Software: Handling Data Transmission and Reception

- Full duplex: Check TXE and RXNE bits

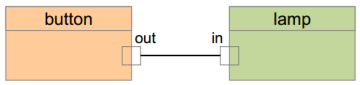


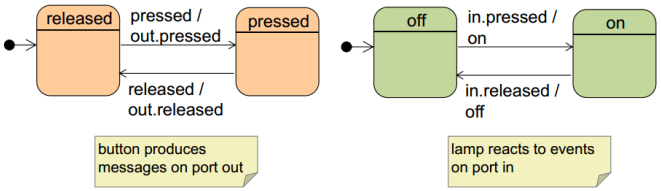
c) Die Zeit f ür den Timerüberlauf soll 200 ms betragen. Welche Werte müssen Sie in die Register PSC und ARR schreiben (Angabe hexadezimaler Werte)?
Hinweis: Es sind verschiedene richtige Lösungen möglich.

```
TIM3_PSC = 0x20CF   // (8400 - 1) entspricht 10 kHz
TIM3_ARR = 0x07CF   // (2000    - 1) entspricht 200 ms

TIM3_PSC = 0x0347   // (840 - 1) entspricht 100 kHz
TIM3_ARR = 0x4E20   // (20000    - 1) entspricht 200 ms

TIM3_PSC = 0x01A3   // (420 - 1) entspricht 200 kHz
TIM3_ARR = 0x9C3F   // (40000    - 1) entspricht 200 ms
```

State-event applications

- Process control
  - Washing machines
  - Vending machines
  - Heating systems





Für eine Autowaschanlage soll eine Software Steuerung entwickelt werden, welche wie folgt spezifiziert ist:

- Die Anlage soll im Ruhezustand auf das Drücken der Starttaste warten. Wurde die Starttaste gedrückt, sollen nacheinander die drei Arbeitsschritte 'Waschen', 'Spülen' und 'Trocknen' ausgeführt werden.
- Jeder der drei Arbeitsschritte soll gleich lange dauern. Für die Kontrolle der Zeitdauer steht ein Timer zur Verfügung.
- Beim Arbeitsschritt 'Waschen' soll Wasser und Shampoo eingeschaltet sein.
- Beim Arbeitsschritt 'Spülen' soll nur Wasser eingeschaltet sein.
- Beim Arbeitsschritt 'Trocknen' soll nur der Luftstrom eingeschaltet sein.
- Bei jedem Arbeitsschritt soll der Ablauf durch Drücken der Stopptaste abgebrochen werden können. Alle Aktoren sollen ausgeschaltet werden und die Steuerung soll in den Ruhezustand zurückkehren.

Folgende Ereignisse (Events) sind definiert:

start	Die Starttaste wurde gedrückt.
stop	Die Stoptaste wurde gedrückt.
time_out	Der Timer ist abgelaufen.

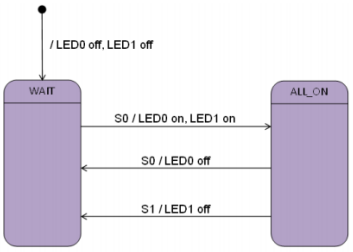
Die Aktoren und der Timer können mit den folgenden Meldungen angesteuert werden:

water_on	Wasser wird eingeschaltet.
water_off	Wasser wird ausgeschaltet
shampoo_on	Das Beimischen des Shampoos wird eingeschaltet.
shampooAus	Das Beimischen des Shampoos wird ausgeschaltet.
air_on	Der Luftstrom für das Trocknen wird eingeschaltet.
air_off	Der Luftstrom für das Trocknen wird ausgeschaltet.
timer_start	Startet den Timer neu.

Zeichnen Sie das State-Diagramm der Steuerung in UML, mit den oben definierten Ereignissen und Meldungen.

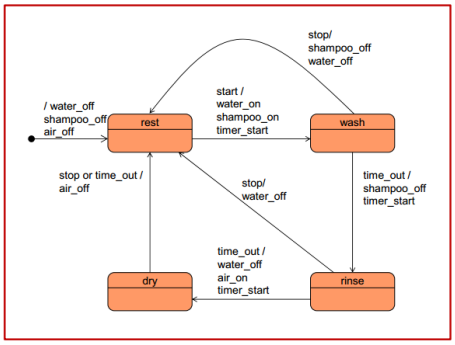
Example: LED control

```
int main(void)
{
    event_t event;

    fsm_init();
    while (1) {
        event = get_event();
        if (event != NO_SWITCH){
            fsm_handle_event(event);
        }
        wait();
    }
}
```



→ see code example

state = ?		
WAIT	ALL_ON	default
event = ?	event = ?	
S0 / default	S0 / S1 / default	
LED0 on LED1 on state = ALL_ON	LED0 off / LED1 off state = WAIT / state = WAIT	state = WAIT




```
switch (state) {
    case STATE_A:
        switch (event) {
            case S0:
                action();
                state = NEW_STATE;
                break;
            default:
                state = WAIT;
        }
        break;

    case STATE_B:
        switch (event)
```

Value = $(-1)^S \cdot (1 + F) \cdot 2^{(E - Bias)}$

- More exponent bits → wider range of numbers
- More fraction bits → higher precision

Exponent value = (E - Bias)

- Bias: 127 (float) / 1023 (double)
- Example (float):

0111'1010b = 122 → 122 – 127 = -5 → Value = 2^{-5}

Value = 2^{-5} → -5 + 127 = 122 = 0111'1010b

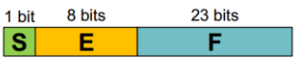
- Decimal: 0.00002796 = $2.796 \cdot 10^{-5}$
- Binary: 0.000010110111 = $1.0110111 \cdot 2^{-5}$

Fraction value = (1 + F)

- Example

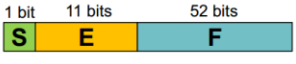
1.265625 d = 1.010001 b → F = 010001

Single precision (float)



- 32 bits
- Exponent bias: 127
- Dynamics: $-3.4 \cdot 10^{38}$ ... $-1.17 \cdot 10^{-38}$, 0, $1.17 \cdot 10^{-38}$ ... $3.4 \cdot 10^{38}$
- Resolution: $2^{-24} = 0.6 \cdot 10^{-7}$ → 7 digits

Double precision (double)



- 64 bits
- Exponent bias: 1023
- Dynamics: $-1.8 \cdot 10^{308}$ ... $-2.2 \cdot 10^{-308}$, 0, $2.2 \cdot 10^{-308}$ ... $1.8 \cdot 10^{308}$
- Resolution: $2^{-53} = 0.11 \cdot 10^{-15}$ → 15 digits

Decimal 7.5 d = ?

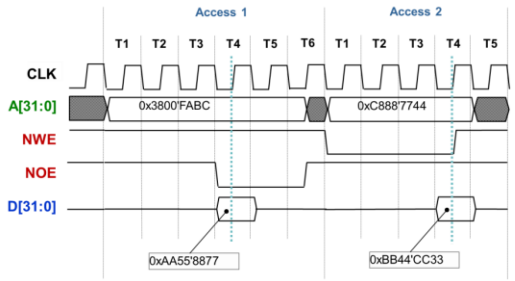
7.5 : 4 = 1 111.1
3.5 : 2 = 1
1.5 : 1 = 1
0.5 : 2^-1 = 1

1.111*2^2
EXP = 2
E = 2 + 127 = 129
= 1000'0001

0 1000'0001 111000... (so geschrieben reicht es)
In Hex = 40F0000h (muss man nicht)

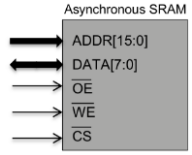
Binary 1011111'01010000'00000000'00000000 = ?

s = -1
E = 0111'1110
- 0111'1111
= -1
F = 101
Fraction = 1.101
Value = 1.101 * 2^-1
= 0.1101
= (-1) * 1/2+1/4+1/16 wegen sign

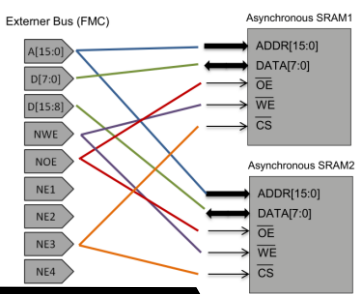


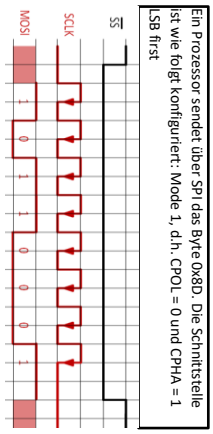
Access1: Lesen - 0xAA55'8877 von Adresse 0x3800'FABC

Access2: Schreiben - 0xBB44'CC33 nach Adresse 0xC888'7744



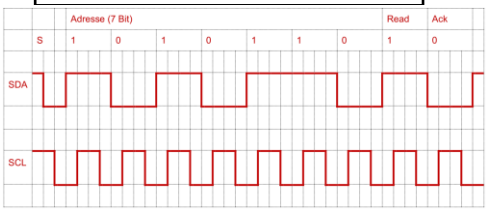
Was ist die Grösse y des Bausteines in KBytes (y K x 8bit) ? **y = 2^16 = 64 KBytes**



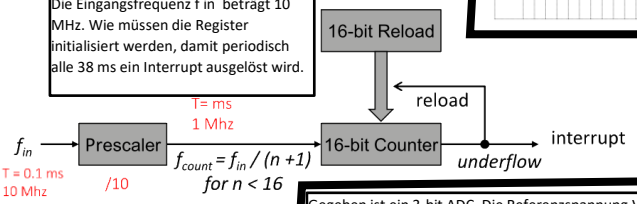


Adresse	Inhalt (Byte)	Nach schribzugriff
0xC888'7747	0xBB	
0xC888'7746	0x44	
0xC888'7745	0xCC	
0xC888'7744	0x33	

Mittels I²C soll ein Baustein zum Lesen adressiert werden. Die 7-Bit Adresse lautet 0x56. Zeichnen Sie die übertragenen Signale vom Start bis zum Acknowledge auf.



Die Eingangsfrequenz f in beträgt 10 MHz. Wie müssen die Register initialisiert werden, damit periodisch alle 38 ms ein Interrupt ausgelöst wird.



Tin = 1 / fin = 1 / (10 MHz) = 100 ns

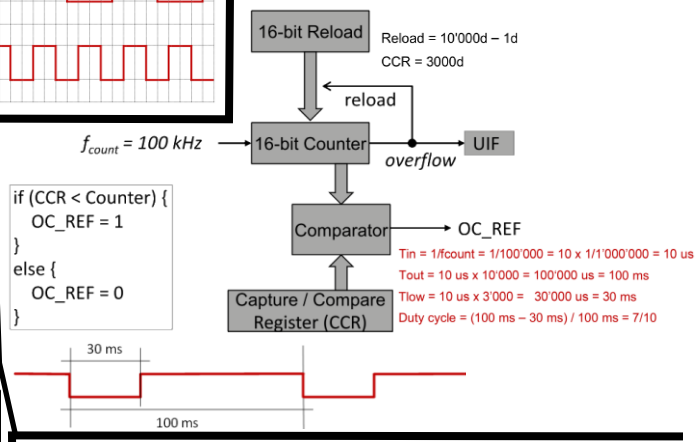
Tout = 38'000'000 ns

Tout/Tin = 380'000 = z.B. 10 x 38'000

Prescaler n = 10 – 1 = 9 → Tcount = 1 us

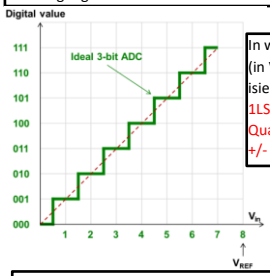
Reload = Counter = 38'000d – 1d

38'000 * 1 us = 38 ms



Gegeben ist ein 3-bit ADC. Die Referenzspannung VREF ist auf 8V festgelegt.

a) Nehmen Sie an, der ADC ist „idea|“. Zeichnen Sie die Übertragungsfunktion!



In welchem Spannungsbereich (in V) bewegt sich der Quantisierungsfehler?

1LSB =vREF/2“ =sv/2=*= 1v

Quantisierungsfehler:

+/- 0.5 LSB = +/- 0.5V

Gegeben ist ein realer 3-bit ADC. Er hat keinen Offsetfehler (offset error = O LSB). Ab einer Spannung von 5V gibt der ADC den digitalen Wert von „111“ aus. Wie gross ist der Verstarkungsfehler (Gain error)? Der Losungsweg muss nachvollziehbar sein.

Full scale range FSR = VREF -1 LSB = 8V — 1V = 7V

111 sollte bei 6.5V erreioht werden (siehe Ubertragungsfunktion).

-> Gain error = 1.5V = 1.5LSB

Geben Sie an, wie -4.75d im Single Precision-Format nach IEEE Standard 754/ 854 abgelegt wird. Ihr Lésungsweg muss nachvollziehbar sein.

Dezimal -> Binär: -4.75d = -100.11b

Normalisieren: 100.11b = 1.0011b * 22

Mantisse: 0011b

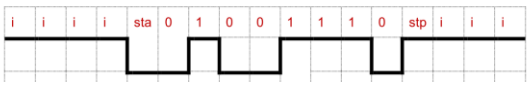
Exponent: E = 2d + 127d = 129d = 1000'0001b

Sign: -4.75<0 -> V = 1

Packen:

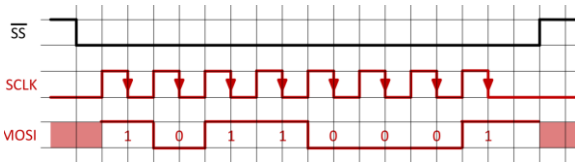
1000000'10011000'00000000'00000000

Auf einer seriellen Datenleitung (UART) messen Sie den folgenden zeitlichen Verlauf. Die Übertragung verwendet ein Startbit, ein Stoppbit, 8 Datenbits ohne Parity-bit bei 4'800 baud.



Ein Prozessor sendet über SPI das Byte 0x8D. Die Schnittstelle ist wie folgt konfiguriert:

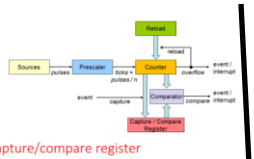
- Mode 1, d.h. CPOL = 0 und CPHA = 1
- LSB first



Capture:
Bei einem Event wird der Inhalt des Counter Registers in das Capture / Compare Register kopiert. Der Counter I läuft weiter.
Compare:
Sobald der Counter den Wert des Capture / Compare Register erreicht hat, wird ein Event oder ein Interrupt ausgelost. Der Counter läuft weiter.

Compare example

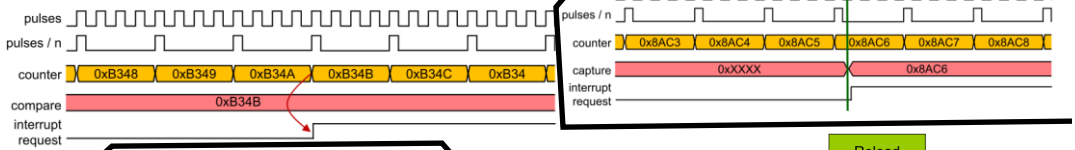
- Raise an alarm when specified count is reached or exceeded
- Continuously compare counter value to a reference value



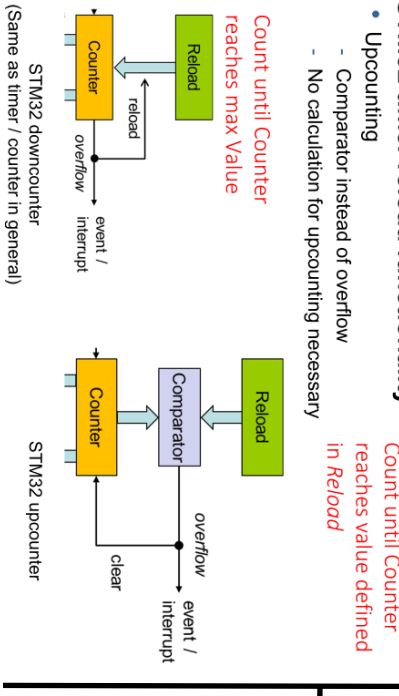
Event is triggered when counter reaches capture/compare register

Capture example

- Stop watch
- At which moment in time does the user push the button?
  - Time of event is captured
  - Count continues



STM32 timer reload functionality



- Upcounting
  - Comparator instead of overflow
  - No calculation for upcounting necessary

Timer Configuration

Register address = Base address + Offset

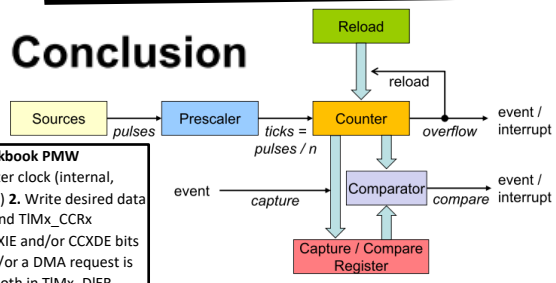
- Offset address is given for each register in Reference Manual
- Base address defined in memory map
  -> Reference Manual pg. 64 ff

Boundary address	Peripheral
0x4000 0C00 - 0x4000 0FFF	TIM5
0x4000 0800 - 0x4000 0BFF	TIM4
0x4000 0400 - 0x4000 07FF	TIM3
0x4000 0000 - 0x4000 03FF	TIM2
0x4002 3800 - 0x4002 3BFF	RCC

RCC: Reset and Clock Configuration

Programming cookbook PMW
1. Select the counter clock (internal, external, prescaler) **2.** Write desired data in the TIMx_ARR and TIMx_CCRx registers **3.** Set CCXIE and/or CCXDE bits if an interrupt and/or a DMA request is to be generated (both in TIMx_DIER register) **4.** Select the output mode (registers CCMRX / CCER) **5.** Enable counter by setting the CEN bit in the TIMx_CR1 register

Conclusion



Reset value is 0x0 as long as nothing else is given

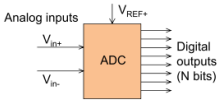
Input signals

- Differential inputs
- V_{in+} signal to convert (non-inverting input)
- V_{in-} signal to convert (inverting input)

$V_{in} = V_{in+} - V_{in-}$

Single ended mode

- Only V_{in+} used
- V_{in-} is grounded



Reference voltage V_{REF+}

- Internal or external stable voltage
- Needed to weight input voltage

$V_{in} = (digital\ value) * V_{REF+} / (2^N)$

Resolution

- Number of bits N
- Size of digital word

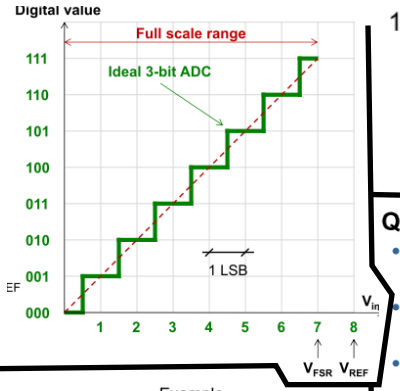
LSB[1]

- 1 LSB ≙ $V_{REF} / (2^N)$

$V_{out} = (digital\ value) * V_{REF} / (2^N)$

$V_{in} = (digital\ value) * V_{REF+} / (2^N)$

1 LSB ≙ $V_{REF} / (2^N)$



Full Scale Range (FSR)

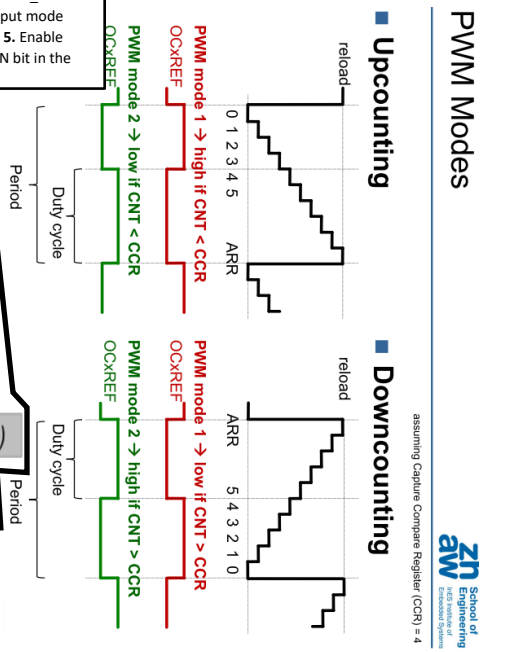
- Range between analog levels of minimum and maximum digital codes
- V_{FSR} is one LSB less than V_{REF}

Example

V_{REF} = 8 V, N = 3 bits

→ 1 LSB = 8 V / 8 = 1 V
→ FSR from 0 V to 7 V

PWM Modes

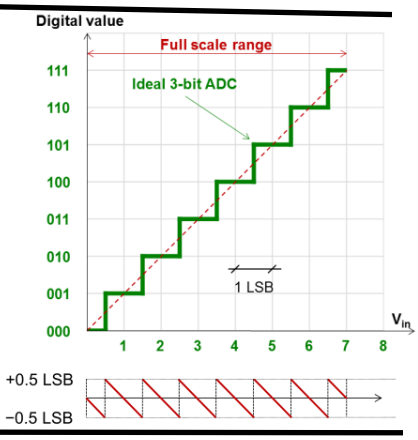


Quantization error

- Analog input is continuous
  - Infinite number of states
- Digital output is discrete
  - Finite number of states
- Introduces an error between −0.5 LSB and +0.5 LSB

Quantization error can be reduced by reducing LSB, e.g. either by increasing number of bits (resolution) or by reducing V_{REF}

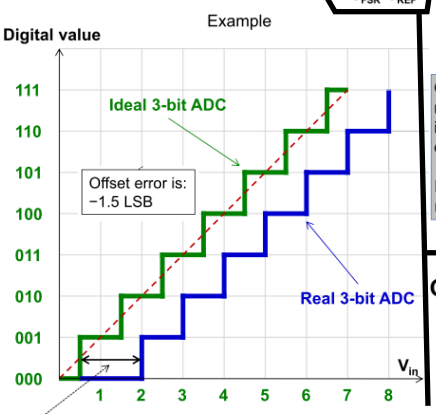
Reducing V_{REF} also reduces full scale range



Offset error

- Also called zero-scale error
- Deviation of real N-bit ADC from ideal N-bit ADC at input point zero
- For an ideal N-bit ADC, the first transition occurs at 0.5 LSB above zero
- Can be corrected using the microcontroller

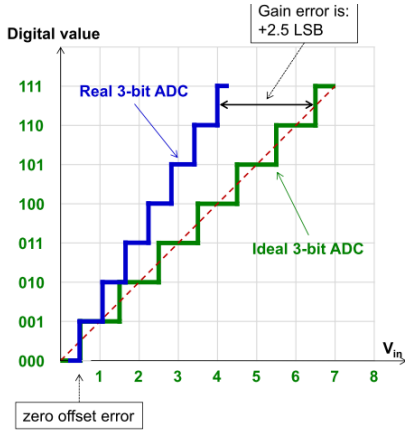
Measuring the offset error: Zero-scale voltage is applied to analog input and is increased until first transition occurs



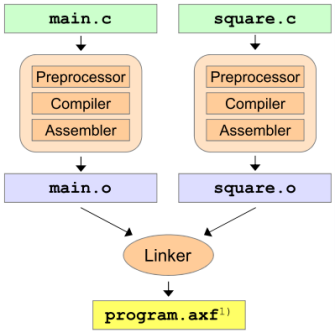
Gain error

- Indicates how well the slope of an actual transfer function matches the slope of the ideal transfer function
- Expressed in LSB or as a percent of full-scale range (%FSR)
- Calibration with hardware or software possible

full-scale error = offset error + gain error



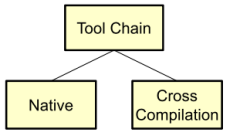
From source code to executable program



Topic	Benefits
Enable working in teams	Multiple developers working on the same source repository
Useful partitioning and structuring of the programs	Eases reusing of modules
Individual verification of each module	Benefits all users of the module
Providing libraries of types and functions	For reuse instead of reinvention
Mixing of modules that are programmed in various languages	E.g. mix C and assembly language modules
Only compile the changed modules	Speeds up compilation time

Tool chain

- Minimal view
  - The set of tools that is required to create from source code an executable for a given environment
- Native tool chain
  - Builds for the same architecture where it runs on
- Cross compiler tool chain
  - Builds for another architecture than the one it runs on
  - E.g. build in KEIL (on Windows) for the CT Board (bare-metal ARM)
- Professional view: there is more than the “compiler & linker”:
  - Editing tools (IDE), revision control tools, documentation tools, testing tools, build tools, deployment tools, issue tracking tools, …




```c
// square.c
...
uint32_t square(uint32_t v) {
  return v*v;
}
```

square = external linkage

Code	
`int square(int v) { return v * v; }`	External
`int square(int v) { int res = v * v; return res; }`	No
`static int square(int v) { return v * v; }`	Internal

Daten mit einer Rate von 16 kbit/s. im Schnitt 100 Clockzyklen.

Quantifizieren Sie den Einfluss des Interrupts auf das System. D.h. welchen Anteil in Prozent der Gesamtrechenzeit verbringt das System mit der Behandlung der Interrupts?

Interruptfrequenz = (16 kbit/s) / 32 bit = 500 Hz
interrupt service time = 100 * 1/(1 MHz) = 100 us
Impact = Interruptfrequenz * interrupt service time * 100 % =500Hz*100us*100%=5%

Bei welcher Datenrate der Schnittstelle wiirde der Prozessor 100% der Rechenzeit mit der Behandlung von Interrupts verbringen?
(X / 32 bit)*1OO µs *100 % =1OO % ->
X = 32 * (1/100) Mbit/s = 320 kBit/s

```c
// main.c
#include "square.h"
static uint32_t a = 5;
static uint32_t b = 7;
int main(void) {
  uint32_t res;
  res = square(a) + b;
  ...
}
```

a = internal linkage
b = internal linkage
main = external linkage
res = no linkage
square = external linkage[1)]

Tasks of a Linker

- **Merge object file data sections**
  - Place all data sections of the individual object files into one data section of the executable file
- **Merge object file code sections**
  - Place all code sections of the individual object files into one code section of the executable file
- **Resolve used external symbols**
  - Search missing addresses of used external symbols
- **Relocate addresses**
  - Adjust used addresses since merging the sections invalidated the original addresses[1)]

Linker-Task	Situation
1) Merge code sections	a) Modul A ruft eine Funktion aus Module B auf
2) Merge data sections	b) Modul A und Modul B enthalten Instruktionen
3) Resolve referenced symbols	c) Modul A und Modul B enthalten globale Daten
4) Relocate addresses	d) Verwendete Referenzen im Code müssen an die neue Lage der Symbole angepasst werden

Impact on system performance

Prüfung

- Percentage of CPU time used to service interrupts

$$\text{Impact} = f_{INT} * t_{ISR} * 100\ \%$$

fint = wie oft kommt er vor
fisr = zeit benötigt zum bearbeiten

- Example keyboard

$f_{INT} = 20\ Hz = 20\ \frac{1}{s}$
$t_{ISR} = 6\ us$ [1)]
Impact = 20 Hz * 6 us * 100 % = 0.012 %

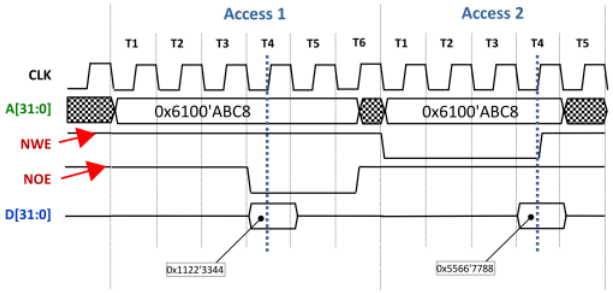
- Example serial interface with 230'400 Baud

$f_{INT} = 230'400 / 8 = 28'800\ Hz$
$t_{ISR} = 6\ us$ [1)]
Impact = 28'800 Hz * 6 us * 100 % = 17.3 %

- **t_{ISR} > "Time between two interrupt events"**
  - Some interrupt events will not be serviced (lost)
    - Data will be lost
  - f_{INT} as well as t_{ISR} may vary over time
    - Average may be ok, but individual interrupt events may still be lost

wenn interrupt länger braucht als zeit bis zum nächsten interrupt gehen daten verloren

- **Caution in case of several interrupt sources**
  - Interrupts can occur simultaneously
  - Computing power required for both ISRs



access 1: ***read von Adresse 0x6100'ABC8 mit Wert 0x1122'3344***
access 2: ***write nach Adresse 0x6100'ABC8 mit Wert 0x5566'7788***

Lesen Sie den Wert eines 8-bit Control Registers an der Adresse 0x6100 '0007 in eine von Ihnen zu definierende Variable ein.

```
#define M¥_BYTE_REG (*((volatile uint8_t *)(0x61000007)))
uint8_t my_var;
my_yar = M¥_BYTE_REG;
```

Bits eines 16-Bit Control Registers an Adresse 0x6100 '0008 auf ‚1'.

```
#define M¥_HALFWORD_REG (*((volatile uint16_t *)(0x61000008)))
MY_HALFWORD_REG = OXFFFF;
```

Warten Sie in einer Schleife, bis Bit 15 im 32-bit Control Register an der Adresse Ox6100 'OOOC auf ‚1' gesetzt ist.

```
#define M¥;WORD_REG (*((volatile uint32_t *)(0x6100000C)))
while(! (MY_WORD_REG & 0xOO008000)){}
```

Setzen Sie Bit 16 im Control Register an Adresse Ox61OO '0010 auf ‚1' ohne die anderen Bits des Registers zu verandern.

```
#define MY_WORD_REG2 (*((volatile uint32_t *> (Ox6l0O00l0)))
MY_WORD_REG2 |= 0x00010000;
```

	NOR Flash [1]		NAND Flash	
	trifft zu	trifft nicht zu	trifft zu	trifft nicht zu
Erlaubt das Ausführen von Programmcode direkt aus dem Speicher.	X			X
Der Speicher kann nur Sektor-weise gelöscht, d.h. auf '1' geschrieben werden.	X		X	
Eignet sich für wahlfreie Zugriffe auf einzelne Bytes.	X			X
Eignet sich für das effiziente Speichern von grossen Datenblöcken.		X	X	
Erfordert eine spezielle Schnittstelle, welche nicht mit asynchronous SRAM Lesezugriffen kompatibel ist.		X	X	
Bits in einem einzelnen Byte können immer auf '0' geschrieben werden.	X		X	
Bits in einem einzelnen Byte können immer auf '1' geschrieben werden.		X		X
Wenn Programmcode abgelegt wird, dann muss dieser in der Regel vor der Ausführung ins RAM geladen werden.		X	X	
Verwendet die Floating Gate Transistor Technologie	X		X	
Der erste Lesezugriff benötigt eine hohe Latenzzeit, die folgenden Lesezugriffe erfolgen aber sehr viel schneller.		X	X	
Grosse Datenblöcke können schnell abgespeichert werden.		X	X	
Wird vor allem für das Speichern von Programmcode und von persistenten Daten eingesetzt.	X			X
Solid-State-Disks (SSD) werden aus diesem Speicher gebaut.		X	X	

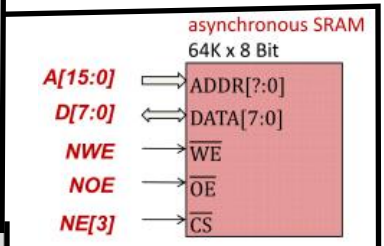
access 1:

0x6100'ABCB	0x11
0x6100'ABCA	0x22
0x6100'ABC9	0x33
0x6100'ABC8	0x44

access 2:

0x6100'ABCB	0x55
0x6100'ABCA	0x66
0x6100'ABC9	0x77
0x6100'ABC8	0x88

Bus Timing Options
Synchronous: Master and slaves use a common clock 1), Clock edges control bus transfer on both sides, Used by almost all on-chip busses, Off-chip: DDR and synchronous RAM
Asynchronous: Slaves have no access to the clock of the master Control signals carry timing information to allow synchronization, Widely used for low data-rate off-chip memories ->parallel flash memories and asynchronous RAM



Wie viele Adresspins benotigt der Baustein?
64K = 2''' a 16 Adresspins a ADDR[15:0]

Unter wie vielen 64KByte Adressblocken kann auf den Baustein zugegriffen werden?
A[25:16] -> 10 Adresslinien -> 2^10 = 1024 Adressblocke: 0X68XX'0000, 0X69XX'0000, 0x6AXX'0000, 0X6BXX'0000

System Bus mit den 6 Adresslinien `A[5:0]`.

2^6 = 64 Byte Adressen
Unter welchen Adressen (in Hex) kann das Control Register angesprochen werden,wenn nur die oberen 4 Adressleitungen wie folgt dekodiert werden: *select = A[5] & A[4] & !A[3] & !A[2]*
1100XXb → 0x30, 0x31, 0x32, 0x33
genau unter der Adresse 0x28 angesprochen
select = A[5] & !A[4] & A[3] & !A[2] & !A[1] & !A[0]
Address lines Unidirectional: From master to slave, Number of lines -> yields size of address space, Cortex-M 32 parallel address lines, 2 32 = 4 Giga addresses 0x0000'0000 – 0xFFFF'FFFF
Data lines 8, 16, 32 or 64 parallel lines of data, bidirectional (read/write)
Control signals Control read/write direction, Provide timing information

Two Possibilities

- 1. Individual wait states can be programmed at a bus interface unit
  - Depending on the address of the bus cycle
  - gut für konstanten Zugriffszeiten

- 2. Slave tells bus interface unit when it is ready
  - Well suited for slaves with long or variable access times
  - gut für variable Zugriffszeiten

ready signal indicates when the slave is ready to provide the data

CPU writes `0x8899'AABB` to address `0x2100'0248`

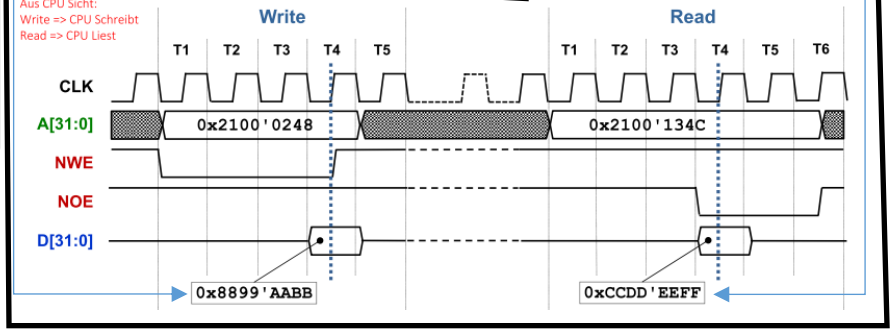
```
LDR   R0,=0x2100'0248
LDR   R1,=0x8899'AABB
STR   R1,[R0]
```

CPU reads `0xCCDD'EEFF` from address `0x2100'134C`

```
LDR   R0,=0x2100'134C
LDR   R1,[R0]
```

Assuming that the memory contains this value

Aus CPU Sicht:
Write => CPU Schreibt
Read => CPU Liest

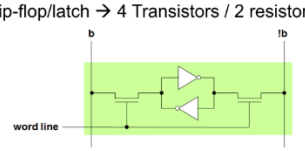
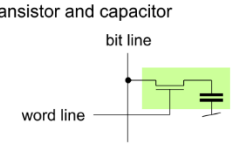


Asynchronous SRAM

$\overline{CS}$	$\overline{OE}$	$\overline{WE}$	I/O	Function
L	L	H	DATA OUT	Read Data
L	X	L	DATA IN	Write Data
L	H	H	HIGH-Z	Outputs Disabled
H	X	X	HIGH-Z	Deselected

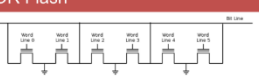
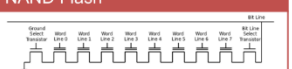
SDRAM – Synchronous Dynamic RAM

zh aw School of Engineering InES Institute of Embedded Systems

Static RAM (SRAM)	Synchronous Dynamic RAM (SDRAM)
Flip-flop/latch → 4 Transistors / 2 resistors 	Transistor and capacitor 
Large cell • Low density, high cost • Up to 64 Mb per device	Small cell • High density, low cost • Up to 4 Gb per device
Almost no static power consumption • Static i.e. no accesses taking place	Leakage currents • Requires periodic refresh
Asynchronous interface (no clock) • Simple connection to bus	Synchronous interface (clocked) • Requires dedicated SDRAM Controller
All accesses take roughly the same time • ~5ns per access → 200 MHz • Suitable for distributed accesses	Long latency for first access of a block • Fast access for blocks of data (bursts) • Large overhead for single byte

Non-volatile Memory – Flash

zh aw School of Engineering InES Institute of Embedded Systems

	NOR Flash	NAND Flash
Topology		
Applications	• Execute code directly from memory • Persistent device configurations (replacement of EEPROM)	• File-based IO, disks • Large amounts of sequential data (images, SD cards, SSD) • Load programs into RAM before executing
Density	• Medium Up to 2 GBit = 256 MByte	• High Up to 1 Tbit
Interface	• Read same as asynchronous SRAM • Types with serial interface available	• Special NAND flash interface • Error correction for defective blocks
Access	• Random access read ~0.12 µs • Writing individual bytes possible • Slow writes ~180 µs / 32 Byte	• Slow random access read: 1. Byte 25 µs, then 0.03 µs each • Writing individual bytes difficult • Fast block write ~300 µs / 2'112 Bytes