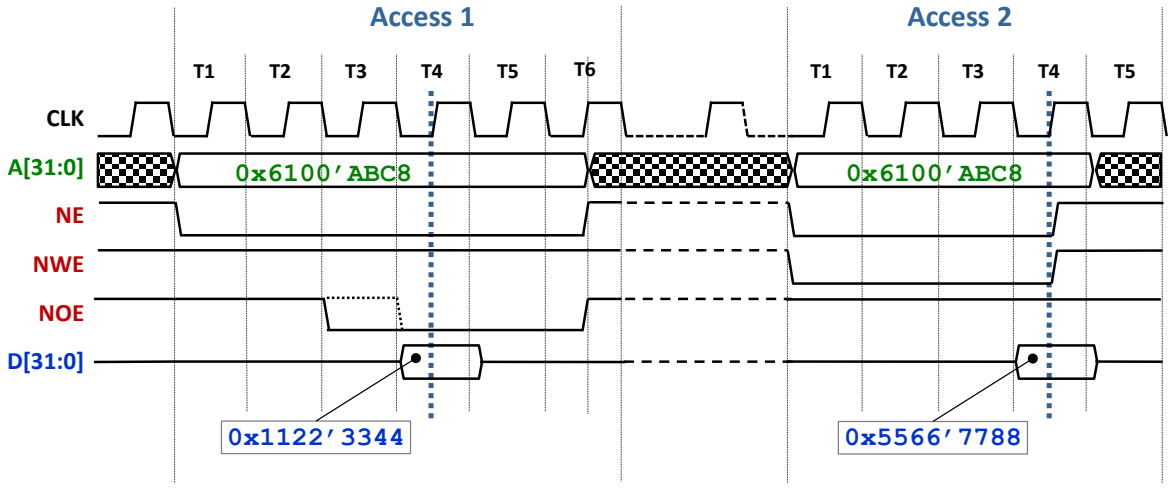
CT Übungsaufgaben
Microcontroller Basics

Aufgabe 1

Gegeben sind die folgenden Buszugriffe



a) Geben Sie für beide Zugriffe jeweils die Richtung an (write oder read) sowie die Adresse des Zugriffs und den geschriebenen bzw. gelesenen Wert.

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

b) Was enthält der Speicher **vor** dem Zugriff "Access 1"? Tragen Sie die Bytes, auf welche zugegriffen wird, mit ihren Adressen in der Memory Map ein. Der Prozessor ist little endian.

Adresse	Inhalt (Byte)
0x6100'ABC8	0x44
0x6100'ABC9	0x33
0x6100'ABCA	0x22
0x6100'ABCB	0x11

c) Was enthält der Speicher **nach** dem Zugriff "Access 2"? Tragen Sie die Bytes, auf welche zugegriffen wird, mit ihren Adressen in der Memory Map ein. Der Prozessor ist little endian.

Adresse	Inhalt (Byte)
0x6100'ABC8	0x88
0x6100'ABC9	0x77
0x6100'ABCA	0x66
0x6100'ABCB	0x55

Aufgabe 2

Gegeben ist ein System Bus mit den 6 Adresslinien **`A[5:0]`**.

a) Wie viele Bytes können damit adressiert werden?

2^6 = 64 → Es können 64 Bytes adressiert werden.

b) Unter wie vielen Adressen kann ein 8-bit Control Register angesprochen werden, wenn dafür 4 dieser Adressleitungen dekodiert werden?

$2^{(6-4)}$ = 4 → Das Byte kann unter 4 Adressen angesprochen werden.

c) 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

d) Unter welchen Adressen (in Hex) kann das Control Register angesprochen werden, wenn nur die unteren 4 Adressleitungen wie folgt dekodiert werden:

```
select = !A[3] & A[2] & A[1] & !A[0]
```

XX0110b → 0x06, 0x16, 0x26, 0x36

e) Unter welchen Adressen (in Hex) kann das Control Register angesprochen werden, wenn nur die mittleren 4 Adressleitungen wie folgt dekodiert werden:

```
select = !A[4] & !A[3] & A[2] & !A[1]
```

X0010Xb → 0x04, 0x05, 0x24, 0x25

f) Wie müssen die Adressen dekodiert werden, wenn das Control Register genau unter der Adresse **`0x28`** angesprochen werden soll ?

```
select = A[5] & !A[4] & A[3] & !A[2] & !A[1] & !A[0]
```

Aufgabe 3

Schreiben Sie Codesequenzen in C für die folgenden Fälle. Verwenden Sie unsigned integer Typen aus **`stdint.h`**

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

```
#define MY_BYTE_REG   (*((volatile uint8_t *)(0x61000007)))

uint8_t my_var;

my_var = MY_BYTE_REG;
```

b) Setzen Sie sämtliche Bits eines 16-Bit Control Registers an Adresse **`0x6100'0008`** auf ‚1'.

```
#define MY_HALFWORD_REG (*((volatile uint16_t *)(0x61000008)))

MY_HALFWORD_REG = 0xFFFF;
```

c) Warten Sie in einer Schleife, bis Bit 15 im 32-bit Control Register an der Adresse **`0x6100'000C`** auf ‚1' gesetzt ist.

```
#define MY_WORD_REG   (*((volatile uint32_t *)(0x6100000C)))

while (!(MY_WORD_REG & 0x00008000)){
}
```

d) Setzen Sie Bit 16 im Control Register an Adresse **`0x6100'0010`** auf ‚1' ohne die anderen Bits des Registers zu verändern.

```
#define MY_WORD_REG2   (*((volatile uint32_t *)(0x61000010)))

MY_WORD_REG2 |= 0x00010000;
```

CT Übungsaufgaben
GPIO Konfiguration und Programmierung

Auf dem STM32F429 Discovery Board soll der Wert des User Buttons (B1) eingelesen werden und der invertierte Wert an der grünen User Led (LD3) oder der roten User Led (LD4) oder in einem eigenen Muster ausgegeben werden. Die Dokumentation finden Sie unten (Seite 3) und verwenden Sie das in OLAT abgelegte Referenzmanual.

Aufgaben:

a) Welche GPIO-Ports müssen für die Ein- bzw. Ausgabe angesprochen werden? Geben Sie den Namen und die Basisadressen an.

GPIO Port A → Eingabe 0x40020000
GPIO Port G → Ausgabe 0x40021800

b) Welche Register müssen für die Ausgabe an der LED konfiguriert werden? Geben Sie die Namen und die Adressen an.

MODER 0x40021800
TYPER 0x40021804
SPEEDR 0x40021808
PUPDR 0x4002180C

c) Konfiguration (Bit-Werte), welche Bits müssen vom Reset her verändert werden?

MODER 0x1 << (2*13) // Mode Out, Pin 13
TYPER 0x0 << 13 // Push Pull, Pin 13
SPEEDR 0x1 << (2*13) // Speed med., Pin 13
PUPDR 0x0 << (2*13) // No PullUp, Pin 13

d) Welche Register müssen für die Eingabe konfiguriert werden? Geben Sie die Namen und die Adressen an.

MODER 0x40020000
TYPER 0x40020004 // für Eingabe nicht relevant
SPEEDR 0x40020008 // für Eingabe nicht relevant
PUPDR 0x4002000C

e) Konfiguration (Bit-Werte), welche Bits müssen vom Reset her verändert werden?

MODER 0x0 << (2*0) // Mode In, Pin 0
TYPER // Wird nicht verwendet, ignoriert
SPEEDR // Wird nicht verwendet, ignoriert
PUPDR 0x0 << (2*0) // No PullUp, Pin 0

f) Wie werden die beiden LED gelöscht? Geben Sie Port, Registername und –Adresse sowie die dazugehörigen Bitwerte an

Port G + BSSR_offset 0x1 << (16+13) // Clr 0..15→ bit 16..31
0x40021800 + 0x18
Port G + BSSR_offset 0x1 << (16+14) // Clr 0..15→ bit 16..31
0x40021800 + 0x18

g) Schreiben Sie eine Eingabemethode, die den aktuellen Zustand des User Buttons B1 einliest und zurückgibt, damit wissen Sie, welche Werte B1 im Ruhezustand und im gedrückten Zustand liefert.

Port A + IDR_offset 0x40020000 + 0x10	Gelesenen Wert mit 0x1 maskieren, dann auf 1 (gedrückt) testen.

Hinweis:
Sie können die GPIO *mit dem gegebenen* Programmrahmen auch auf einem STM32F429 ohne CT-Board verwenden. Auf dem CT-Board muss dazu der Drehschalter P10 in Stellung 0 gesetzt werden. Zudem muss im Projekt-Teil „Options for target 1“ im Reiter C/C++ im Abschnitt „Preprocessor Symbols“ im Feld „Define:“ **NO_FMC** eingetragen werden.

Ideen für die Programmierung (freiwillig):

h) Anschliessend erweitern Sie die main-Schleife so, dass das Drücken des Buttons an der oder den Led angezeigt wird.

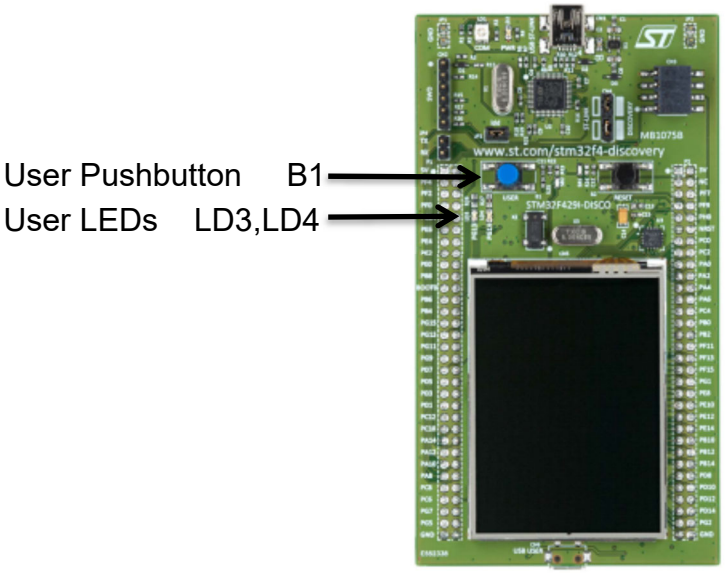
Button gedrückt ➔ grün leuchtet:

Port G + BSSR_offset 0x40021800 + 0x18	0x1 << (13)

Button gedrückt ➔ grün leuchtet, rot aus
Button nicht gedrückt ➔ grün aus, rot leuchtet:

Port G + BSSR_offset 0x40021800 + 0x18	0x1 << (13) und 0x1 << (16+14)
Port G + BSSR_offset 0x40021800 + 0x18	0x1 << (14) und 0x1 << (16+13)

Ausschnitt aus der Dokumentation des STM32F429 Boards



4.4 LEDs

- User LD3:
  Die grüne LED ist eine User LED, die über **I/O Port G Pin 13** des STM32F429ZIT6 Boards angeschlossen ist.
- User LD4:
  Die rote LED ist eine User LED, die über **I/O Port G Pin 14** des STM32F429ZIT6 Boards angeschlossen ist.

4.5 Taster

- B1 USER:
  Der blaue User-Button ist über **I/O Port A Pin 0** des STM32F429ZIT6 Boards angeschlossen.

Die komplette Dokumentation des STM32F429ZIT6 Boards ist auf OLAT zu finden.

CT Übungsaufgaben
Serielle Datenübertragung

Aufgabe 1

Auf einer seriellen asynchronen Übertragungsleitung (UART) mit 19‘200 Bit/s, 7 Daten-Bits, und einem Stop-Bit (ohne Parity Bit) soll die Zeichenfolge ”AC” übertragen werden.

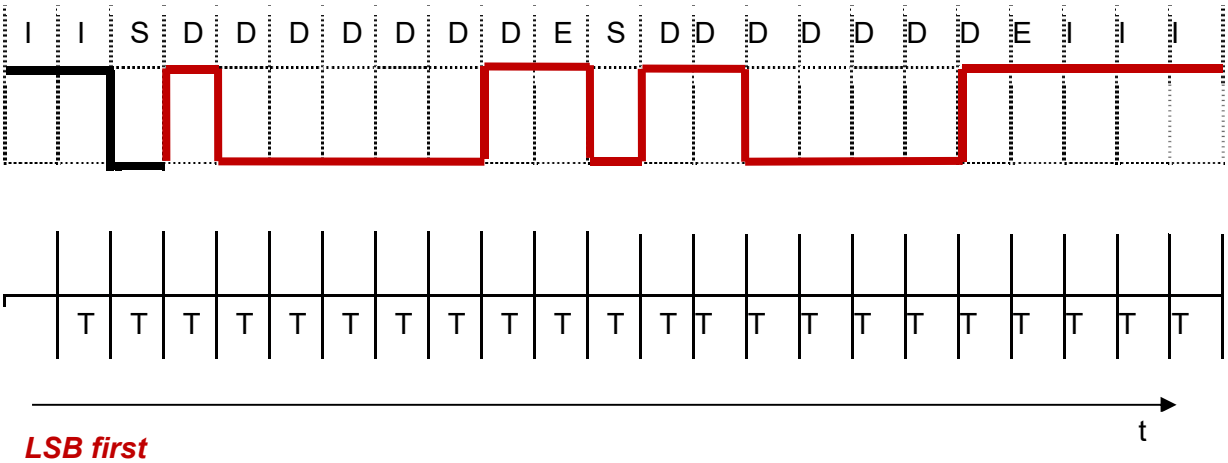
ASCII('A') = 0x41 = 100 0001b
ASCII('C') = 0x43 = 100 0011b

a) Wie lange dauert die Übertragung eines Bits (Periode T)?

1/19‘200 s = 52.1 us

b) Zeichnen Sie den zeitlichen Verlauf der Übertragung ein. Bezeichnen Sie die einzelnen Bits wie folgt:

S → Start-Bit
E → Stop-Bit (End)
D → Daten-Bit
I → Idle-Bit (keine Übertragung)



LSB first

c) 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 = ~ 6.67%

Aufgabe 2

a) Wie synchronisieren sich Sender und Empfänger bei einer UART?

Mit Hilfe der negativen Flanke des ersten übertragenen Bits
Letztes Bit (Stopp) oder Idle → '1'
Erstes Bit (Start) = '0'

b) 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

9600/12 bit/s = 800 Bytes/s

Aufgabe 3

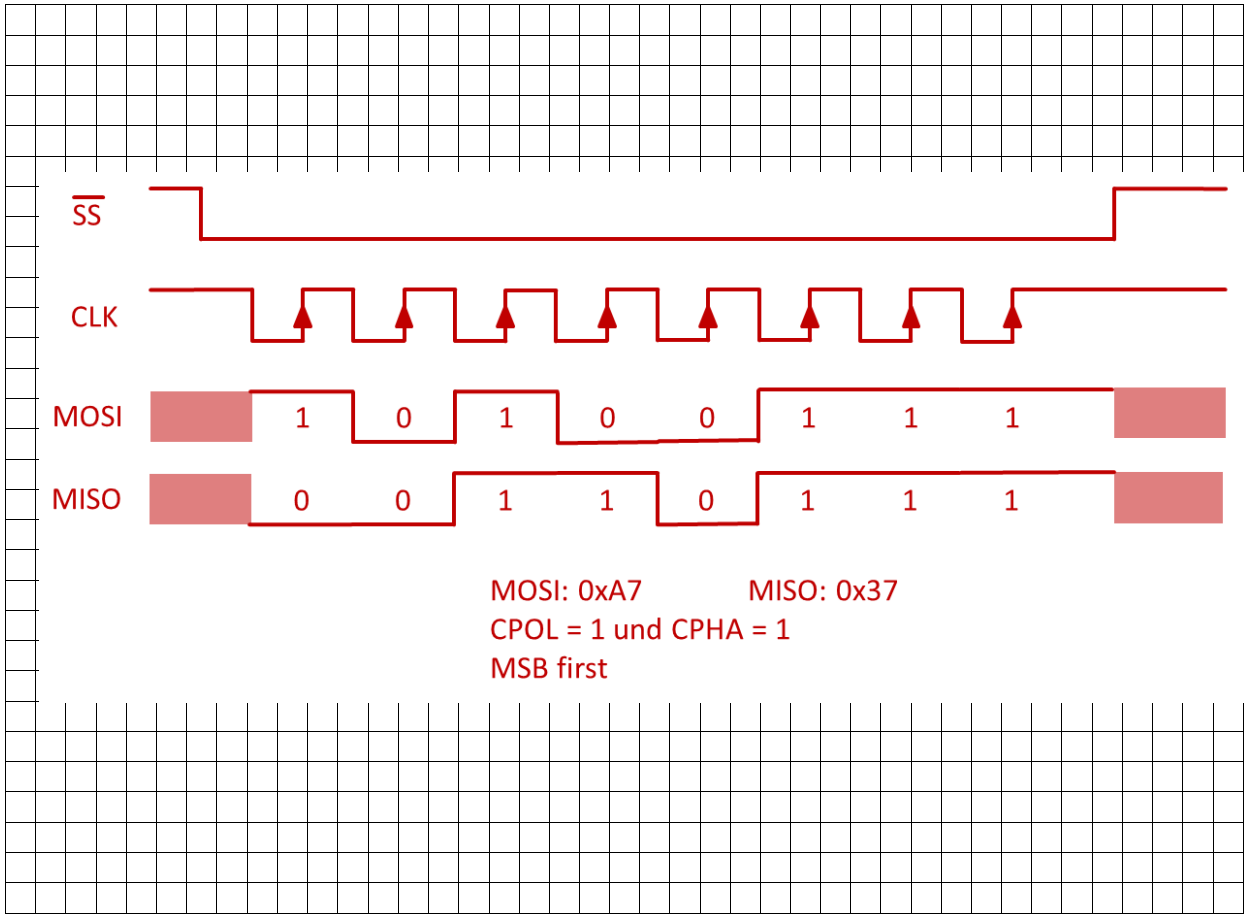
a) Zeichnen Sie das Timing Diagramm (Signale SCLK, SS#, MOSI, MISO) für ein SPI Interface

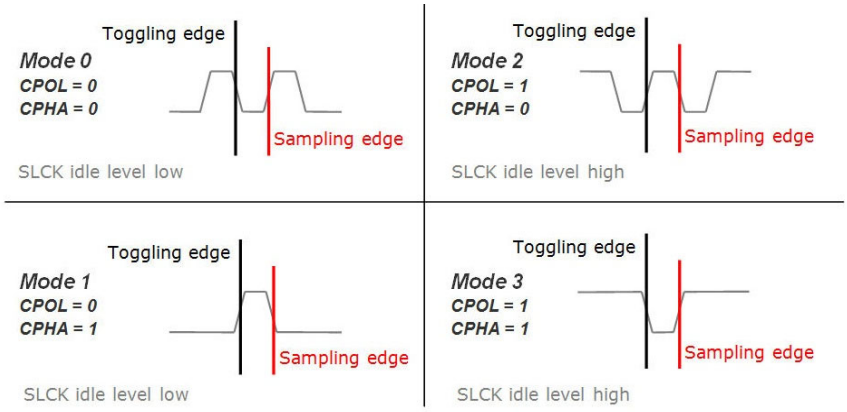
- 8 Bit Daten (MOSI: 0xA7, MISO: 0x37), MSB first
- Mode 3 d.h. CPOL = 1 und CPHA = 1

b) Zeichnen Sie die Sampling Edges ein.

c) Wie lange dauert eine Bit Cell d.h. eine Clockperiode wenn SCLK eine Frequenz von 100kHz hat?

0.01 ms





CT Übungsaufgaben
Timer/Counter, PWM

Aufgabe 1

a) Erklären Sie in Stichworten die einzelnen Funktionseinheiten eines Timers anhand der Tabelle.

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

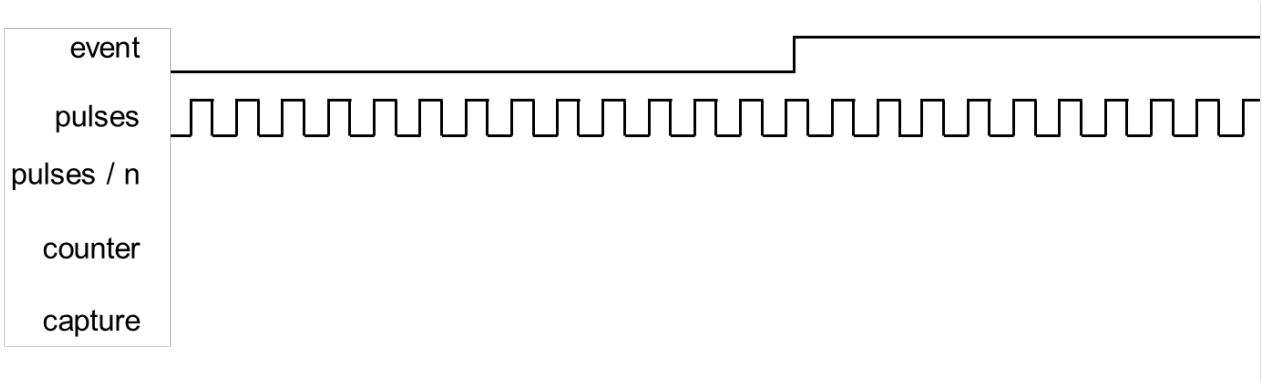
b) Erläutern Sie die Funktion Capture.

Bei einem Event wird der Inhalt des Counter Registers in das Capture / Compare Register kopiert. Der Counter läuft weiter.

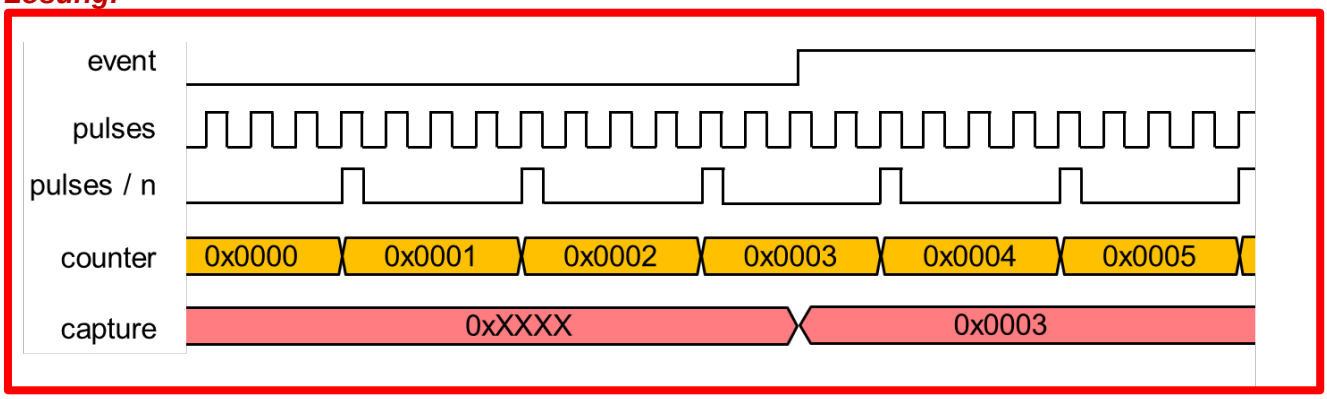
c) Erläutern Sie die Funktion Compare.

Sobald der Counter den Wert des Capture / Compare Register erreicht hat, wird ein Event oder ein Interrupt ausgelöst. Der Counter läuft weiter.

d) Ergänzen Sie das gegebene Timing Diagram. Die Funktion Capture wird bei steigender Flanke des Signals „event “ ausgelöst. Der Timer ist als Upcounter konfiguriert, und der Prescaler ist auf 4 (im Register steht 0x03) eingestellt. Die Startwerte entsprechen sonst dem Zustand nach einem Reset.



Lösung:



Aufgabe 2

Es soll Timer 3 des STM32F429 konfiguriert werden. Verwenden Sie das Reference Manual (zu finden in OLAT) zur Lösung der Aufgabe. Geben Sie die entsprechende Codezeilen in C an.

a) Als Source soll die interne Clock CK_INT mit 84 MHz verwendet werden. Setzen Sie die Bits im entsprechenden Register auf die notwendigen Werte.
Hinweis: andere Bits des Registers sollen nicht verändert werden.

```
TIM3_SMCR &= 0xFFF8;   // TIM3_SMCR [2:0]   0b000
```

b) Der Timer soll als Upcounter konfiguriert werden. Hinweis: andere Bits des Registers sollen nicht verändert werden.

```
TIM3_CR1 &= 0xFF8F
```

c) Die Zeit für 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 = 0x4E1F   // (20000   – 1) entspricht 200 ms

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

Aufgabe 3

Timer 4 des STM32F429 ist bereits als Upcounter konfiguriert und läuft. Das Reload Register enthält folgenden Wert:

```
TIM4_ARR = 0x9C3F
```

a) Geben Sie den Wert für das CCR-Register an, damit ein Duty Cycle von 25% mittels PWM Mode 1 erzeugt wird.

```
TIM4_CCR1 = 0x2710 // 0x9C3F=(40000-1) -> 0x2710=(10000)
```

TIMx_CNT zählt von 0 …39999 = 40000 Ticks
-> Duty Cycle 25% entspricht 10000 Ticks

PWM Mode 1 (Upcounting): OC_REFx='1' as long as TIMx_CNT<TIMx_CCR otherwise OC_REFx= '0'
-> 0…9999 = 10000 Ticks -> TIMx_CNT<TIMx_CCR -> OC_REFx= '1'
-> 10000…39999 = 30000 Ticks -> TIMx_CNT>=TIMx_CCR -> OC_REFx= '0'

b) Nun ist Timer 4 als Downcounter im PWM Mode 2 statt als Upcounter im PWM Mode 1 konfiguriert. Was müssen Sie ändern, um ein identisches elektrisches Signal zu erhalten (Werte)?

TIM4_CCR1 anpassen

```
TIM4_CCR1 = 0x752F  // entspricht (30000-1)
```

PWM Mode 2 (Downcounting): OC_REFx='1' as long as TIMx_CNT>TIMx_CCR otherwise OC_REFx= '0'
-> 39999…30000 = 10000 Ticks -> TIMx_CNT>TIMx_CCR -> OC_REFx= '1'
-> 29999…0 = 30000 Ticks -> TIMx_CNT<=TIMx_CCR -> OC_REFx= '0'

Aufgabe 4

Der Timer ist als Upcounter konfiguriert. Bestimmen Sie das generierte PWM-Signal am Ausgang (Zahlen + Skizze). Die Source liefert ein Signal der Frequenz 0,5 MHz.

Die relevanten Konfigurationsregister sind wie folgt initialisiert:

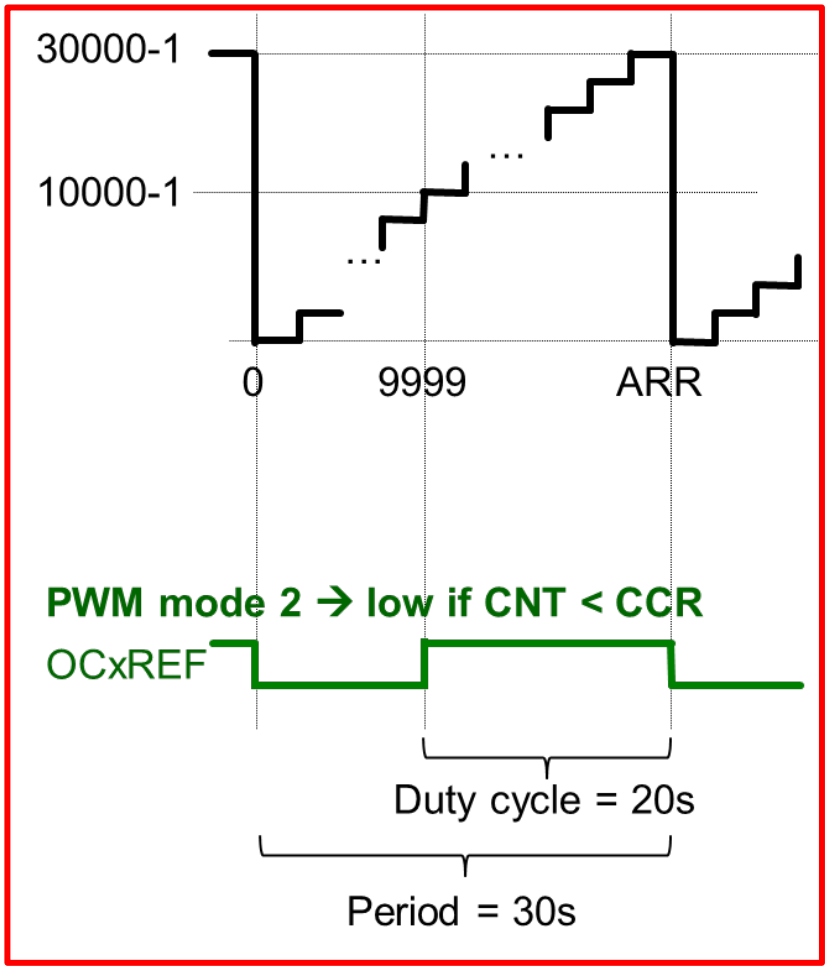
```
TIM3_PSC      = 0x01F3
TIM3_ARR      = 0x752F
TIM3_CCR1     = 0x2710
TIM3_CCMR1    = 0x0070
```

Welches Signal wird erzeugt? Zeichnen Sie das Signal und geben Sie die Werte für Periode und Duty Cycle an (Zeitangaben).

Lösung:

Gegebene Register entsprechen:

```
TIM3_PSC      = 0x01F3  // (500-1) -> 1 kHz
TIM3_ARR      = 0x752F  // (30000-1) -> Periode 30 Sekunden
TIM3_CCR1     = 0x2710  // (10000) -> 10 Sekunden
TIM3_CCMR1    = 0x0070  // PWM Mode 2
```



Periode: 30 Sekunden, Duty Cycle: (30 Sekunden-10Sekunden) = 20 Sekunden

CT Übungsaufgaben
ADC

1. Offset errors

a. Draw the transfer function of an ideal 3-Bit ADC. V_{REF} is set to 2V.
b. That ADC has an offset error of -1.5 LSB.
    I. Draw the new transfer function
    II. Convert the LSB in volts
    III. Convert the offset error in Volts
    IV. Calculate the offset error in % of FSR

2. Programming the ADC

Consider the ADC registers and addresses given below. Assume that the ADCs are properly initialized and started for regular channel conversion.

Write C code (including correct register addresses) to do the following:

a. Wait until ADC1 conversion has completed.
b. Set an 8-bit variable (var2) to 0xFF if there was a loss of data on ADC2, otherwise reset that variable to 0.
c. Set ADC3 resolution to 10-bit.

13.13 ADC registers

Refer to *Section 1.1 on page 57* for a list of abbreviations used in register descriptions.

The peripheral registers must be written at word level (32 bits). Read accesses can be done by bytes (8 bits), half-words (16 bits) or words (32 bits).

13.13.1 ADC status register (ADC_SR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										OVR	STRT	JSTRT	JEOC	EOC	AWD
										rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Bits 31:6 Reserved, must be kept at reset value.

Bit 5 **OVR:** Overrun

This bit is set by hardware when data are lost (either in single mode or in dual/triple mode). It is cleared by software. Overrun detection is enabled only when DMA = 1 or EOCS = 1.

0: No overrun occurred

1: Overrun has occurred

Bit 4 **STRT:** Regular channel start flag

This bit is set by hardware when regular channel conversion starts. It is cleared by software.

0: No regular channel conversion started

1: Regular channel conversion has started

Bit 3 **JSTRT:** Injected channel start flag

This bit is set by hardware when injected group conversion starts. It is cleared by software.

0: No injected group conversion started

1: Injected group conversion has started

Bit 2 **JEOC:** Injected channel end of conversion

This bit is set by hardware at the end of the conversion of all injected channels in the group. It is cleared by software.

0: Conversion is not complete

1: Conversion complete

Bit 1 **EOC:** Regular channel end of conversion

This bit is set by hardware at the end of the conversion of a regular group of channels. It is cleared by software or by reading the ADC_DR register.

0: Conversion not complete (EOCS=0), or sequence of conversions not complete (EOCS=1)

1: Conversion complete (EOCS=0), or sequence of conversions complete (EOCS=1)

Bit 0 **AWD:** Analog watchdog flag

This bit is set by hardware when the converted voltage crosses the values programmed in the ADC_LTR and ADC_HTR registers. It is cleared by software.

0: No analog watchdog event occurred

1: Analog watchdog event occurred

13.13.2 ADC control register 1 (ADC_CR1)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					OVRIE	RES		AWDEN	JAWDEN	Reserved					
					rw	rw	rw	rw	rw						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:27 Reserved, must be kept at reset value.

Bit 26 **OVRIE:** Overrun interrupt enable

This bit is set and cleared by software to enable/disable the Overrun interrupt.
0: Overrun interrupt disabled
1: Overrun interrupt enabled. An interrupt is generated when the OVR bit is set.

Bits 25:24 **RES[1:0]:** Resolution

These bits are written by software to select the resolution of the conversion.
00: 12-bit (15 ADCCLK cycles)
01: 10-bit (13 ADCCLK cycles)
10: 8-bit (11 ADCCLK cycles)
11: 6-bit (9 ADCCLK cycles)

Bit 23 **AWDEN:** Analog watchdog enable on regular channels

This bit is set and cleared by software.
0: Analog watchdog disabled on regular channels
1: Analog watchdog enabled on regular channels

Bit 22 **JAWDEN:** Analog watchdog enable on injected channels

This bit is set and cleared by software.
0: Analog watchdog disabled on injected channels
1: Analog watchdog enabled on injected channels

Bits 21:16 Reserved, must be kept at reset value.

Bits 15:13 **DISCNUM[2:0]:** Discontinuous mode channel count

These bits are written by software to define the number of regular channels to be converted in discontinuous mode, after receiving an external trigger.
000: 1 channel
001: 2 channels
...
111: 8 channels

Bit 12 **JDISCEN:** Discontinuous mode on injected channels

This bit is set and cleared by software to enable/disable discontinuous mode on the injected channels of a group.
0: Discontinuous mode on injected channels disabled
1: Discontinuous mode on injected channels enabled

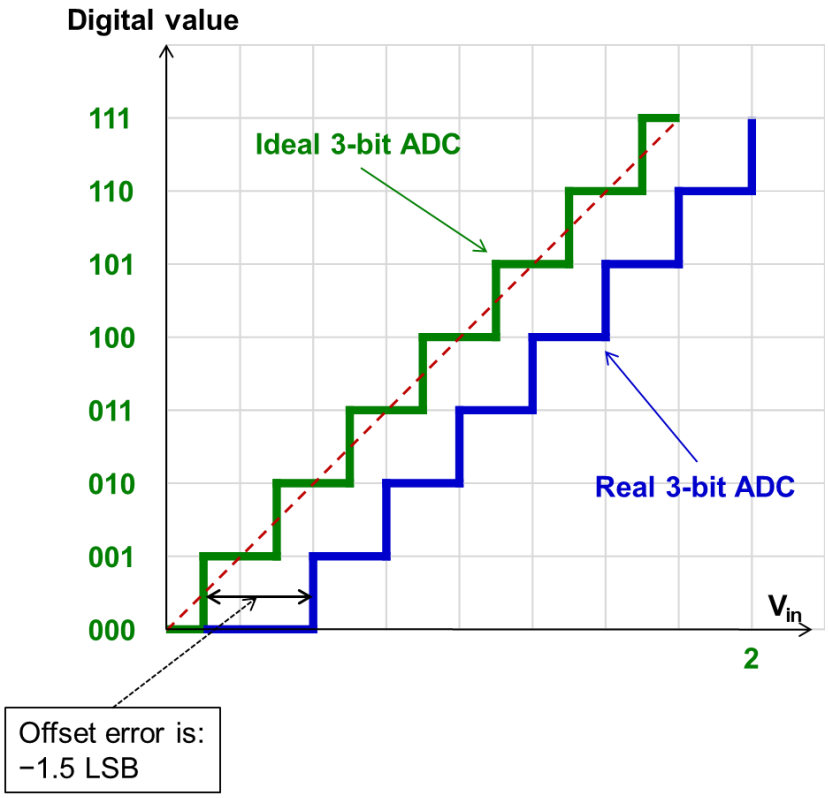
ADC region		offset		Address of register
0x4001 2000 - 0x4001 23FF	ADC1	0x000 - 0x04C	Specific registers	0x40012000 + 0x000 + register offset
			Reserved	
	ADC2	0x100 - 0x14C	Specific registers	0x40012000 + 0x100 + register offset
			Reserved	
	ADC3	0x200 - 0x24C	Specific registers	0x40012000 + 0x200 + register offset
			Reserved	
	Common	0x300 - 0x308	Common registers	0x40012000 + 0x300 + register offset

Solution

1. Offset errors

Solution 1.a (as below, but only the green curve)

Solution for 1.b (only the blue curve)



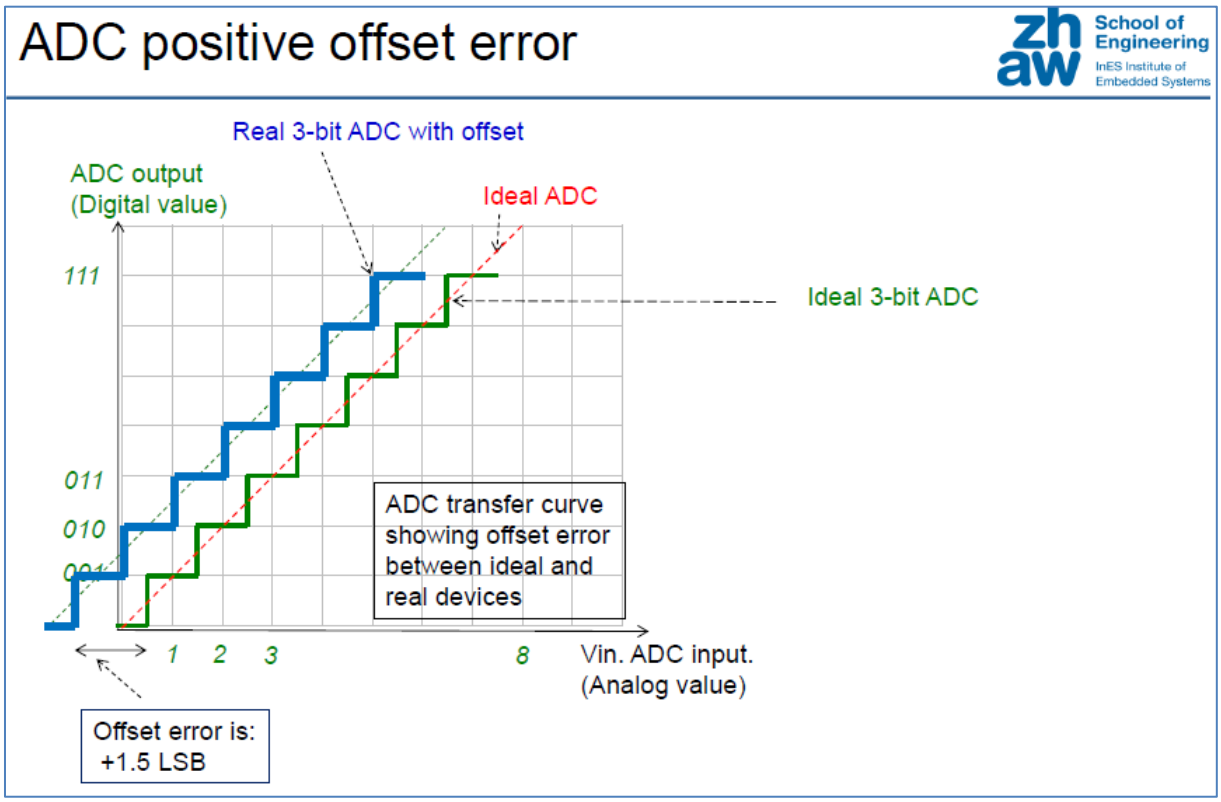
V_{REF} = 2 V → 1LSB = $2/2^3$ = 2/8 = 0.25V

Offset error of -1.5 LSB corresponds to -1.5 * 0.25V = 0.375V

FSR = V_{REF} – 1LSB = 2 - 0.25V = 1.75V

Offset Error (%FSR) = Offset Error (V) * 100 / FSR = 0.375V * 100 / 1.75V = 21.43%

Additional information:
In case of a positive offset error of +1.5 LSB, the graph will look like below.



2. Programming the ADC

```
a.
// Check Bit1 (EOC) of ADC1 status register.
// Address of status register of ADC1 register is:
//0x4001'2000 + base offset of ADC1 + status reg offset = 0x40012000

#define MY_STATUS_REG_ADC1 (*((volatile uint8_t *)(0x40012000)))
while (!(MY_STATUS_REG_ADC1 & 0x02)){
}

b.
// If bit5 of status register is 1, then an overrun has occurred.
// Address of status register of ADC1 register is:
//0x4001'2000 + base offset of ADC2 + status reg offset = 0x40012100

#define MY_STATUS_REG_ADC2 (*((volatile uint8_t *)(0x40012100)))

if (MY_STATUS_REG_ADC2 & 0x20){
    var2 = 0xFF;
} else {
    var2 = 0x00;
}

c.
// Write [bit25 bit24] of ADC3 control register1 to [01]
// Address of that register is:
//0x4001'2000 + base offset ADC3 + Control reg1 offset = 0x40012204
```

```
#define MY_control_REG1_ADC3 (*((volatile uint32_t *)(0x40012204)))

MY_control_REG1_ADC3 |= 0x01000000; // force bit24 to 1
MY_control_REG1_ADC3 &= 0xFDFFFFFF; // force bit25 to 0
```

Could also be written as:

```
MY_control_REG1_ADC3 &= 0xFCFFFFFF; // force [bit25 bit24] to [00]
MY_control_REG1_ADC3 |= 0x01000000; // force [bit25 bit24] to [01]
```

CT Übungsaufgaben
Memory

Aufgabe 1

a) Nennen Sie drei Unterschiede zwischen einem 'asynchronous SRAM' (statisches RAM) und einem SDRAM (Synchronous Dynamic RAM).

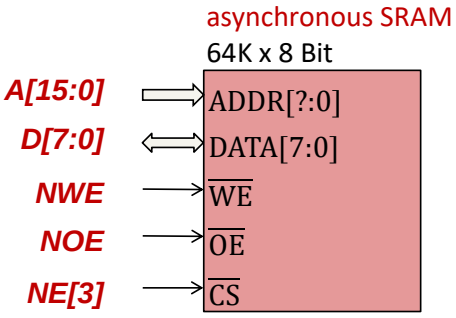
1. ***statisch (Inhalt bleibt bestehen solange Speisung anliegt) vs. dynamisch (permanenter refresh notwendig)***
2. ***Speicherelement: Flip-flop vs. Kondensator***
3. ***Schnittstelle: asynchron (NWE,NOE) vs. synchron (RAS,CAS)***
4. ***Zugriffszeit:***
   SRAM alle Zugriffzeiten gleich lang
   SDRAM Hohe Latenz für ersten Zugriff, kurze Zugriffszeit für Folgeadressen
5. ***SDRAM: kleinere Speicherzellen→ höhere Anzahl von Speicherzellen pro Fläche***

b) Was ist die typische Funktion des Pins $\overline{\mathrm{OE}}$ bei einem asynchronen SRAM?

Der Pin kontrolliert, ob das Memory Daten auf den Bus schreibt (treibt) oder ob sich die SRAM Ausgangstreiber der Datenleitungen im Floating Zustand befinden. Der Pin wird beim Auslesen des RAMs durch den Prozessor aktiviert (low gesetzt).

Aufgabe 2

Gegeben ist der folgende 'asynchronous SRAM' Baustein.



a) Wie viele Adresspins benötigt der Baustein?

64K = 2^{16} → 16 Adresspins → ADDR[15:0]

b) Der Baustein wird an den 'Flexible Memory Controller' (FMC) des STM32F4xxx angeschlossen. Adresse **`0x6800'0000`** soll die tiefste Adresse sein, unter welcher der Baustein angesprochen werden kann. Tragen Sie die anzuschliessenden FMC–Signale direkt links neben den Pfeilen ein.

c) Unter welcher Adresse greifen Sie aus der Software heraus auf das Byte an der höchsten Adresse des Bausteines zu?

0x6800'FFFF

d) Beim Entwickeln der Software stellen Sie fest, dass Sie unter der Adresse **`0x6BFF'0000`** auf das identische Byte des Bausteins wie unter **`0x6800'0000`** zugreifen. Was ist die Erklärung?

Partial Address Decoding: Die Bits A[25:16] sind nicht angeschlossen und werden deshalb nicht decodiert.

e) Unter wie vielen 64KByte Adressblöcken kann auf den Baustein zugegriffen werden?

A[25:16] → 10 Adresslinien → 2^{10} = 1024 Adressblöcke

0x68XX'0000
0x69XX'0000
0x6AXX'0000
0x6BXX'0000

Aufgabe 3

Welche der folgenden Aussagen treffen für ein DRAM (Dynamisches RAM) zu?
Bezeichnen Sie das entsprechende Feld mit einem ‚X'.

	trifft zu	***trifft nicht zu***
Die Daten werden in einer RS Flip-flop artigen Zelle gespeichert.		X
Zugriffe auf einzelne Speicherstellen haben eine hohe Latenz.	X	
Auf Grund der Leckströme ist ein periodischer Refresh notwendig.	X	
Der Preis pro Speicherzelle ist hoch.		X
Ist ein flüchtiger (volatiler) Speicher.	X	
Eignet sich gut für Blockzugriffe.	X	
Falls keine Zugriffe erfolgen, ist der Leistungsverbrauch sehr klein.		X

Aufgabe 4

Kreuzen Sie jeweils für NOR und NAND Flash an, ob die Aussagen zutreffen.

	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	

1) NOR mit Parallel Interface