Die Fragen beziehen sich konkret auf den Mikrocontroller AT91M63200, der zusammen mit diverser Peripherie und weiteren Komponenten auf dem Evaluierungsboard AT91EB63 verbaut ist und üblicherweise im Mikroprozessor-Labor eingesetzt wird.
Zur Lösung der Aufgaben ist es notwendig, die Handbücher der Hardware zur Hilfe zu nehmen!

1. Bit-Operationen

Die Peripheriekomponenten eines Mikrocontrollers werden durch Schreiben und Lesen ihrer Steuer- und Statusregister „bedient“.

Beim ersten Beschreiben eines Konfigurations- oder Datenregisters werden diese vollständig initialisiert. Dazu wird das komplette Register mittels des Zuweisungs-operators „=“ beschrieben.
Beim späteren Ändern einzelner Bits eines Registers werden dazu in der Regel Bit-Operationen benutzt.

Zur besseren Lesbarkeit werden die Bit-Konstanten definiert:

```
#define BIT0        (1<<0)
#define BIT1        (1<<1)
#define BIT2        (1<<2)
#define BIT3        (1<<3)
#define BIT4        (1<<4)
...
```

Definition der Register und deren Adresse:

```
typedef struct
{
   volatile unsigned int DataRegA;
   volatile unsigned int DataRegB;
} StructReg;                                          // Register der Hardware

#define REG_BASE      ((StructReg*) 0xFFFFF000)       // Basisadresse der Register
```

In main() wird das Register A wie folgt initialisiert:

```
REG_BASE->DataRegA = 0x01E4; // Bitmaske einschreiben
```

Der Programmverlauf enthält verschiedene weitere Anweisungen.

Wie lautet der Inhalt des Registers nach dem Ändern? Welche Funktionen haben die folgenden Bit-Operationen?

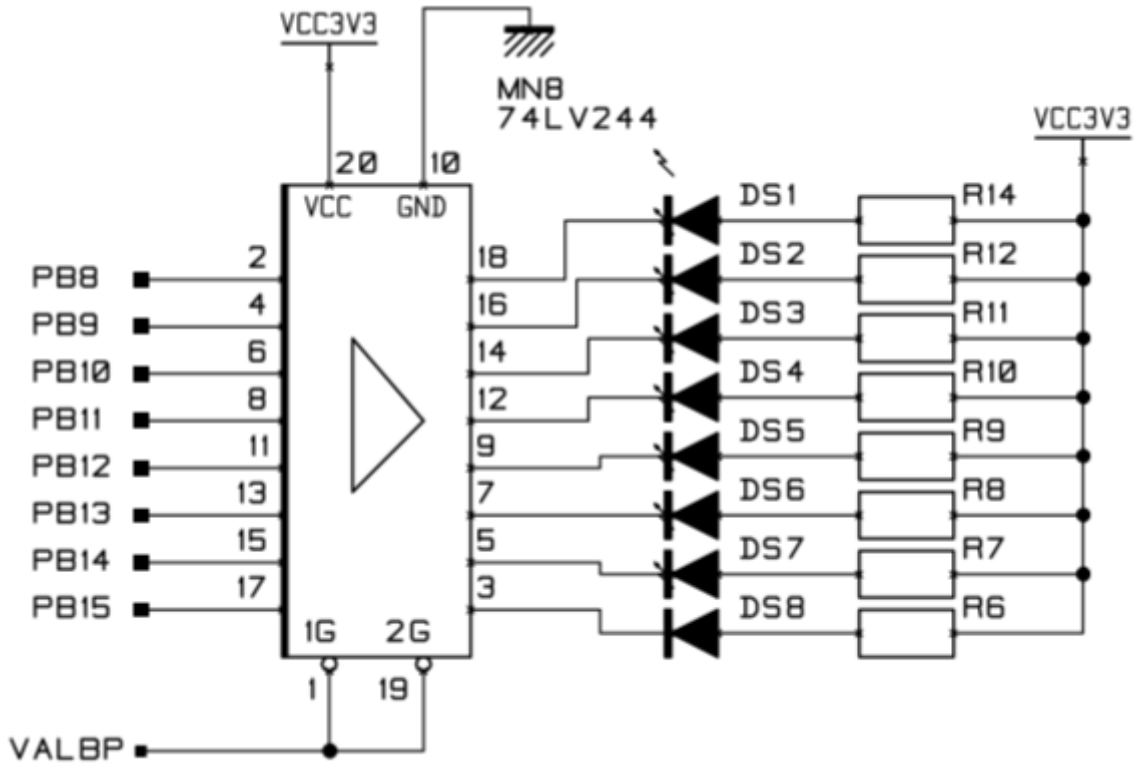
A) ***REG_BASE->DataRegA |= BIT0 | BIT1 ;***

B) ***REG_BASE->DataRegA &= ~(BIT0 | BIT1) ;***

C) ***REG_BASE->DataRegA ^= BIT0 | BIT1 | BIT2;***

2. LED-Ansteuerung

Gegeben ist folgender schematischer Schaltplan des Boards AT91EB63, bei dem die LEDs DS1 bis DS8 an den Mikrocontroller AT91M632000 über dessen Parallelport durch einen Tristate-IC angeschlossen sind (vergleiche **AT91EB63_User_Guide.pdf**, „Figure 6-4. Push Buttons, LEDs, Reset and Serial Interface“, Seite 25)



A)
Müssen die Leitungen PB8 bis PB15 mit einem niedrigen Spannungspotential (Low-Pegel, log. 0) oder einem hohen Spannungspotential (High-Pegel, log. 1) beschaltet werden, damit die LEDs eingeschaltet werden können?

Mit welchem Spannungspotential müssen die Leitungen beschaltet werden, um die LEDs zu auszuschalten?

B)
Vergleichen Sie das Handbuch für den Mikrocontroller zum Thema Parallel I/O (**AT91M63200_Complete.pdf,** „PIO: Parallel I/O Controller“, Seite 55 ff.)!

Wie lautet der Name für das Port-Daten-Register, mit dem man entsprechend einzelne LEDs setzen/einschalten kann?

C)
Betrachten Sie folgenden Pseudo-Quellcode! Listen Sie alle LEDs auf, die durch das Beschreiben des Registers mit dem folgenden Datenwert beeinflusst werden können!

```
int main(void)
{
        StructPIO* piobaseB = PIOB_BASE;  // Basisadresse PIO B
        // …

        piobaseB->PIO_SODR = 0x00001500;
        // ...
 }
```

3. Tastendruck-Verarbeitung

Tastendrücke können durch einfaches Polling zyklisch abgefragt werden.
Die Abfragezeitpunkte sowie die Dauer zwischen den Abfragen sind abhängig von den weiteren Aufgaben innerhalb des Programmzyklus.

Betrachten Sie den folgenden C-Programmcode, bei dem ein Tastendruck zyklisch abgefragt wird und dann dadurch im Wechsel eine LED ein- bzw. wieder ausgeschaltet wird. Die eigentliche Hauptaufgabe des Controllers wird der Einfachheit halber durch eine Zählaufgabe simuliert. Abhängig vom Aufwand in der Hauptaufgabe kann die Definition von MAX_VALUE als Übergabeparameter an die Zählaufgabe angepasst werden.

```
#include "../h/pmc.h"
#include "../h/pio.h"
#include "../h/aic.h"

#define MAX_VALUE 1000000
unsigned int actCountValue = 0;

void count(unsigned value)
{
        unsigned int i;
        for (i =0; i< value; i++)
                actCountValue = i;

        return;
}


int main (void)
{

        int keystate=0;
```

```
        StructPMC* pmcbase =PMC_BASE;          // Basisadresse PMC
        StructPIO* piobaseB = PIOB_BASE;       // Basisadress PIOB

        pmcbase->PMC_PCER = 1 << PIOB_ID;      // Peripheral Clock Enable for PIOB

        piobaseB->PIO_PER = LED1 | KEY1;       // enable LED1 and KEY1
        piobaseB->PIO_OER = LED1;              // LED1 is output
        piobaseB->PIO_ODR = KEY1;              // KEY1 is input

        piobaseB->PIO_SODR = LED1;             // clear LED1


        // main loop
        while(1)
        {
                //do something
                count(MAX_VALUE);


                //switch LED when key pressed
                if(!keystate && piobaseB->PIO_PDSR & KEY1)      // if LED1 is not set
                {
                        piobaseB->PIO_CODR = LED1;              // set LED1
                        keystate = 1;

                }
                else
                {
                        piobaseB->PIO_SODR = LED1;              // clear LED1
                        keystate = 0;
                }
                        r
        }

        return (0);
}
```

Der entsprechende Arm-Assembler-Code, den der Compiler ohne Optimierung daraus generiert hat, sieht folgendermaßen aus:

```
        .file   "ParallelIO_Aufgabe2.c"
        .global actCountValue
        .bss
        .align  2
        .type   actCountValue, %object
        .size   actCountValue, 4
actCountValue:
        .space  4
        .text
        .align  2
        .global count
        .type   count, %function
count:
        @ args = 0, pretend = 0, frame = 8
        @ frame_needed = 1, uses_anonymous_args = 0
```

```
        @ link register save eliminated.
        str     fp, [sp, #-4]!
        add     fp, sp, #0
        sub     sp, sp, #8
        str     r0, [fp, #-8]
        mov     r3, #0
        str     r3, [fp, #-4]
        b       .L2
.L3:    @ for-loop
        ldr     r3, .L5
        ldr     r2, [fp, #-4]
        str     r2, [r3, #0]
        ldr     r3, [fp, #-4]
        add     r3, r3, #1
        str     r3, [fp, #-4]
.L2:    @ check for-loop conditions
        ldr     r2, [fp, #-4]
        ldr     r3, [fp, #-8]
        cmp     r2, r3
        bcc     .L3
        add     sp, fp, #0
        ldmfd   sp!, {fp}
        bx      lr
.L6:
        .align  2
.L5:
        .word   actCountValue
        .size   count, .-count
        .align  2
        .global main
        .type   main, %function
main:
        @ args = 0, pretend = 0, frame = 12
        @ frame_needed = 1, uses_anonymous_args = 0
        stmfd   sp!, {fp, lr}
        add     fp, sp, #4
        sub     sp, sp, #12
        mov     r3, #0
        str     r3, [fp, #-16]
        mov     r3, #-1610612736
        mov     r3, r3, asr #15
        str     r3, [fp, #-12]
        mov     r3, #-2147483648
        mov     r3, r3, asr #15
        str     r3, [fp, #-8]
        ldr     r3, [fp, #-12]
        mov     r2, #16384
        str     r2, [r3, #16]
        ldr     r3, [fp, #-8]
        mov     r2, #264
        str     r2, [r3, #0]
        ldr     r3, [fp, #-8]
        mov     r2, #256
        str     r2, [r3, #16]
        ldr     r3, [fp, #-8]
        mov     r2, #8
        str     r2, [r3, #20]
        ldr     r3, [fp, #-8]
```

```
        mov     r2, #256
        str     r2, [r3, #48]
.L10:   @ main loop
        mov     r0, #999424
        add     r0, r0, #576
        bl      count
        @ check if-conditions
        ldr     r3, [fp, #-16]
        cmp     r3, #0
        bne     .L8
        ldr     r3, [fp, #-8]
        ldr     r3, [r3, #60]
        and     r3, r3, #8
        cmp     r3, #0
        beq     .L8
        @ if case
        ldr     r3, [fp, #-8]
        mov     r2, #256
        str     r2, [r3, #52]
        mov     r3, #1
        str     r3, [fp, #-16]
        mov     r0, r0    @ nop
        b       .L10
.L8:    @ else case
        ldr     r3, [fp, #-8]
        mov     r2, #256
        str     r2, [r3, #48]
        mov     r3, #0
        str     r3, [fp, #-16]
        b       .L10
        .size   main, .-main
        .ident  "GCC: (crosstool-NG 1.13.3 - arm-elf) 4.4.6"
```

A)

Wie lange dauert ein Zyklus der CPU, wenn Sie davon ausgehen, dass der Mikrocontroller mit einer Taktfrequenz von 20 MHz arbeitet?

B)

Schätzen Sie nun die maximale Latenz in Abhängigkeit vom Aufwand in der Hauptaufgabe ab, bis ein Tastendruck verarbeitet werden kann. Gehen Sie dabei zur Vereinfachung von folgenden Annahmen aus:

- Der Taster ist prell frei.
- Pipelining findet nicht statt.
- Jede Arm-Assembler-Instruktion benötigt 2 Taktzyklen.
- Der aktuelle Aufwand der Zählaufgabe count() wird durch die (Re-)Definition von MAX_VALUE als Übergabeparameter bestimmt.

4. Interrupt-Verarbeitung

Gegeben sind folgende Informationen über den Interrupt-Controller (Details siehe **AT91M63200_Complete.pdf,** „AIC: Advanced Interrupt Controller“, Seite 39 ff.).

Register des AIC

AIC User Interface

Base Address: 0xFFFFF000

Table 8. AIC Memory Map

Offset	Register	Name	Access	Reset State
0x000	Source Mode Register 0	AIC_SMR0	Read/Write	0
0x004	Source Mode Register 1	AIC_SMR1	Read/Write	0
–	–	–	Read/Write	0
0x07C	Source Mode Register 31	AIC_SMR31	Read/Write	0
0x080	Source Vector Register 0	AIC_SVR0	Read/Write	0
0x084	Source Vector Register 1	AIC_SVR1	Read/Write	0
–	–	–	Read/Write	0
0x0FC	Source Vector Register 31	AIC_SVR31	Read/Write	0
0x100	IRQ Vector Register	AIC_IVR	Read only	0
0x104	FIQ Vector Register	AIC_FVR	Read only	0
0x108	Interrupt Status Register	AIC_ISR	Read only	0
0x10C	Interrupt Pending Register	AIC_IPR	Read only	(see Note 1)
0x110	Interrupt Mask Register	AIC_IMR	Read only	0
0x114	Core Interrupt Status Register	AIC_CISR	Read only	0
0x118	Reserved	–	–	–
0x11C	Reserved	–	–	–
0x120	Interrupt Enable Command Register	AIC_IECR	Write only	–
0x124	Interrupt Disable Command Register	AIC_IDCR	Write only	–
0x128	Interrupt Clear Command Register	AIC_ICCR	Write only	–
0x12C	Interrupt Set Command Register	AIC_ISCR	Write only	–
0x130	End of Interrupt Command Register	AIC_EOICR	Write only	–
0x134	Spurious Vector Register	AIC_SPU	Read/Write	0

Note: 1. The reset value of this register depends on the level of the external IRQ lines. All other sources are cleared at reset.

Interrupt-Quellen

Table 7. AIC Interrupt Sources

Interrupt Source	Interrupt Name	Interrupt Description
0	FIQ	Fast interrupt
1	SWIRQ	Soft interrupt (generated by the AIC)
2	US0IRQ	USART Channel 0 interrupt
3	US1IRQ	USART Channel 1 interrupt
4	US2IRQ	USART Channel 2 interrupt
5	SPIRQ	SPI interrupt
6	TC0IRQ	Timer Channel 0 interrupt
7	TC1IRQ	Timer Channel 1 interrupt
8	TC2IRQ	Timer Channel 2 interrupt
9	TC3IRQ	Timer Channel 3 interrupt
10	TC4IRQ	Timer Channel 4 interrupt
11	TC5IRQ	Timer Channel 5 interrupt
12	WDIRQ	Watchdog interrupt
13	PIOAIRQ	Parallel I/O Controller A interrupt
14	PIOBIRQ	Parallel I/O Controller B interrupt
15	---	Reserved
16	---	Reserved
17	---	Reserved
18	---	Reserved
19	---	Reserved
20	---	Reserved
21	---	Reserved
22	---	Reserved
23	---	Reserved
24	---	Reserved
25	---	Reserved
26	---	Reserved
27	---	Reserved
28	IRQ3	External interrupt 3
29	IRQ2	External interrupt 2
30	IRQ1	External interrupt 1
31	IRQ0	External interrupt 0

Bitmaske Enable/Disable/Status Register

31	30	29	28	27	26	25	24
IRQ0	IRQ1	IRQ2	IRQ3	---	---	---	---
23	22	21	20	19	18	17	16
---	---	---	---	---	---	---	---
15	14	13	12	11	10	9	8
---	PIOBIRQ	PIOAIRQ	WDIRQ	TC5IRQ	TC4IRQ	TC3IRQ	TC2IRQ
7	6	5	4	3	2	1	0
TC1IRQ	TC0IRQ	SPIRQ	US2IRQ	US1IRQ	US0IRQ	SWIRQ	FIQ

Source Mode Register

AIC Source Mode Register

Register Name: AIC_SMR0...AIC_SMR31
Access Type: Read/Write
Reset Value: 0

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	SRCTYPE		–	–	PRIOR		

- **PRIOR: Priority Level**
  Program the priority level for all sources except source 0 (FIQ).
  The priority level can be between 0 (lowest) and 7 (highest).
  The priority level is not used for the FIQ in the SMR0.

- **SRCTYPE: Interrupt Source Type**
  Program the input to be positive- or negative-edge triggered or positive- or negative-level sensitive.
  The active level or edge is not programmable for the internal sources.

SRCTYPE		Internal Sources	External Sources
0	0	Level Sensitive	Low-Level Sensitive
0	1	Edge Triggered	Negative-Edge Triggered
1	0	Level Sensitive	High-Level Sensitive
1	1	Edge Triggered	Positive-Edge Triggered

A)
Außerdem liegt Ihnen folgender Quellcode vor. Bestimmen Sie anhand dessen die verwendete Interrupt-Quelle.

```
typedef struct
{
   at91_reg      AIC_SMR[32] ;     /* Source Mode Register */
   at91_reg      AIC_SVR[32] ;     /* Source Vector Register */
   at91_reg      AIC_IVR ;         /* IRQ Vector Register */
   at91_reg      AIC_FVR ;         /* FIQ Vector Register */
   at91_reg      AIC_ISR ;         /* Interrupt Status Register */
   at91_reg      AIC_IPR ;         /* Interrupt Pending Register */
   at91_reg      AIC_IMR ;         /* Interrupt Mask Register */
   at91_reg      AIC_CISR ;        /* Core Interrupt Status Register */
   at91_reg      reserved0 ;
   at91_reg      reserved1 ;
   at91_reg      AIC_IECR ;        /* Interrupt Enable Command Register */
   at91_reg      AIC_IDCR ;        /* Interrupt Disable Command Register */
   at91_reg      AIC_ICCR ;        /* Interrupt Clear Command Register */
   at91_reg      AIC_ISCR ;        /* Interrupt Set Command Register */
   at91_reg      AIC_EOICR ;       /* End of Interrupt Command Register */
   at91_reg      AIC_SPU ;         /* Spurious Vector Register */
} StructAIC ;

#define AIC_BASE                 ((StructAIC *)0xFFFFF000)

...

int main(void) {
        StructAIC* aicbase = AIC_BASE; // Basisadresse AIC
        ...

        ...
        aicbase->AIC_SVR[ 0x800] = (unsigned int)irq_handler;
        aicbase->AIC_SMR[0x800] = 0x07;
        aicbase->AIC_IECR = 0x800;

        ...

        while( 1 ); // Endlosschleife
        …
}
```

B)
Bestimmen Sie die Eigenschaften (source mode) des Interrupts!

5. Timer/Counter-Funktionalität

Der Mikrocontroller AT91M63200 verfügt über mehrere Timer/Counter-Einheiten („Timer/Counter Channel“), die in den beiden Modi „Waveform“ (Zeitgeberbetrieb) oder „Capture“ (Zählerbetrieb) eingestellt werden können.

A)
Ordnen Sie die beiden Beschreibungen den Timer/Counter-Betriebsmodi zu!

1) „Impulse kommen über einen Anschlussstift von außen in den Mikrocontroller und werden einfach gezählt. Das Einstellen eines Vorteilers ist möglich.“

2) „Impulse werden durch Herunterteilen des internen Oszillatortaktes gewonnen. Da der Oszillatortakt bekannt ist, korrespondiert derZählwert mit der vergangenen Zeit. Exakte Zeitmessungen sind möglich!“

B)
Welchem TC-Betriebsmodi entspräche die Verwendung eines TC-Channels als periodische Interrupt-Quelle, bei dem die aktuell verstrichene Zeit mit einem bestimmten Wert verglichen wird und bei Erreichen des Wertes einen Interrupt auslöst?

C)
Jede Timer/Counter-Einheit verfügt über ein 16bit großes Zählregister, das kontinuierlich inkrementiert wird.

Welchen Wertebereich kann das Zählregister annehmen?

D)
Folgende Annahmen werden getroffen:
- 16-Bit-Zählregister
- Systemtakt = 25 MHz
- Vorteiler = MCKL/8
- Zähler läuft aufwärts

Berechnen Sie, nach welcher Zeit der beschriebene Zähler einen Überlauf hat!

Ist es mit diesen Annahmen möglich, einen periodischen Interrupt nach jeweils 50 ms auszulösen?

E)
Wie muss der Timer/Counter Channel 5 initialisiert werden, um einen periodischen Interrupt alle 50 ms auslösen zu können? Nehmen Sie dazu wieder an, dass der Systemtakt 25MHz beträgt
Vervollständigen Sie zur Lösung der Aufgabe den folgenden Pseudo-Quellcode für die Initialisierung des TC5!

```
typedef struct
{
  at91_reg      TC_CCR ;      /* Control Register */
  at91_reg      TC_CMR ;      /* Mode Register */
  at91_reg      Reserved0 ;
  at91_reg      Reserved1 ;
  at91_reg      TC_CV ;       /* Counter value */
  at91_reg      TC_RA ;       /* Register A */
  at91_reg      TC_RB ;       /* Register B */
  at91_reg      TC_RC ;       /* Register C */
  at91_reg      TC_SR ;       /* Status Register */
  at91_reg      TC_IER ;      /* Interrupt Enable Register */
  at91_reg      TC_IDR ;      /* Interrupt Disable Register */
  at91_reg      TC_IMR ;      /* Interrupt Mask Register */
  at91_reg      Reserved2 ;
```

```
   at91_reg       Reserved3 ;
   at91_reg       Reserved4 ;
   at91_reg       Reserved5 ;
} StructTC ;

#define TCB5_BASE      ((StructTC *)0xFFFD4080)

void TC5_init()
{
        StructTC* timerbase5  = TCB5_BASE;
        …

        … // TC5 clock disable

        // TC5 mode setup
        timerbase5->TC_____ = _____________;

        // Register value
        timerbase5->TC_____ = _____________;

        // Enable interrupt on register compare
        timerbase5->TC_____ = _____________;

        … // TC5 clock enable and performe a software trigger
}
```

Zur Hilfe betrachten Sie bitte die Informationen über die Timer/Counter des Mikrocontrollers im Referenzhandbuch **AT91M63200_Complete.pdf,** „TC: Timer/Counter“, Seite 111 ff.