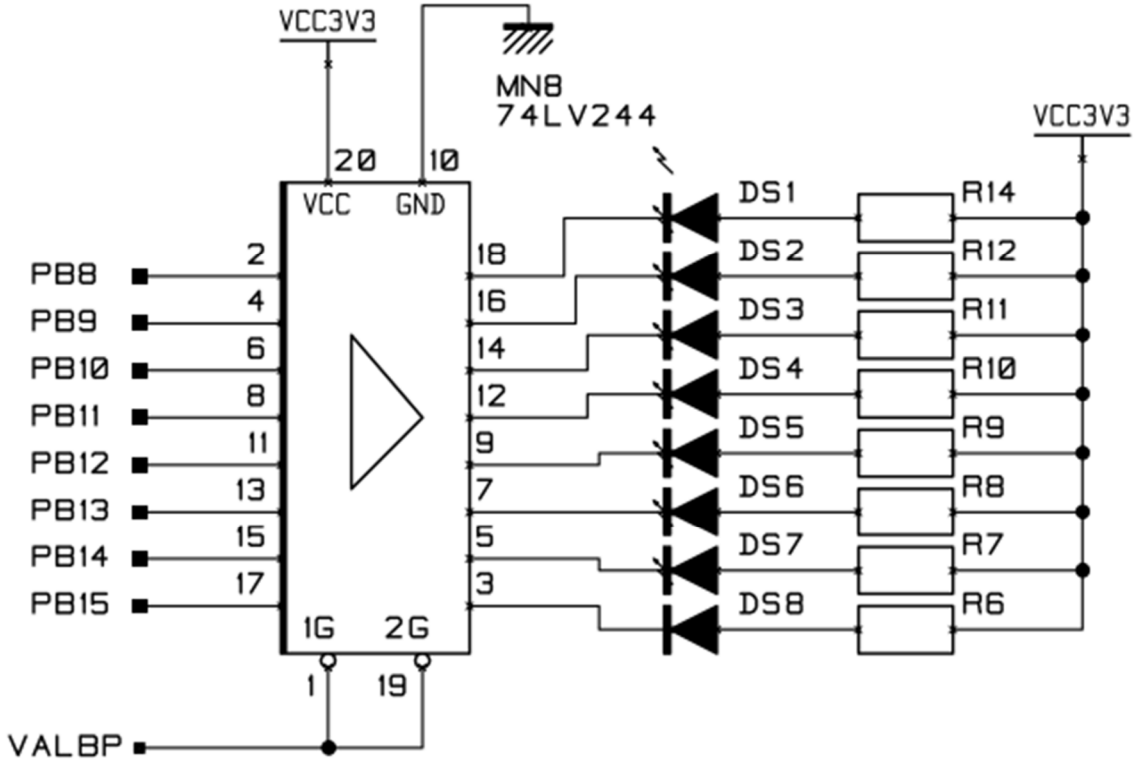
Die Fragen beziehen sich konkret auf den Mikrocontroller AT91M63200, der zusammen mit diverser Peripherie und weiteren Komponenten auf dem Evaluierungsboard AT91EB63 verbaut ist und üblicherweise im Mikroprozessor-Labor eingesetzt wird.
Zur Lösung der Aufgaben ist es notwendig, die Handbücher der Hardware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

1. 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 PB17 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 = 0x00001050;
        // ...
 }
```

D)

Ermitteln theoretisch Sie die exakten Speicheradressen der Register SODR, CODR, ODSR mit Hilfe des Handbuchs für den Mikrocontroller **AT91M63200_Complete.pdf**. Vergleichen Sie dazu die Kapitel „Memory Map“ und „PIO: Parallel I/O Controller“.

E)

Die Datei Termin2Aufgabe1.c enthält Quellcode zum Ansteuern von LED DS1, die an den AT91-Controller links angeschlossen ist. Analysieren und kompilieren Sie den Programmcode mit der SourceNavigator-IDE. Debuggen Sie anschließend das Programm im Simulator.

Ermitteln Sie nun praktisch die exakten Speicheradressen der Register SODR, CODR, ODSR mit Hilfe des Insight-Debuggers und vergleichen Sie diese mit den Ergebnissen der vorherigen Aufgabe!

F)

Öffnen im Unterordner „Termin2Aufgabe1-f“ das vorgegebene Codeblocks-Projekt.

Kompilieren Sie das Projekt und starten Sie das erstellte Programm mit dem „Run“-Button der IDE. Was beobachten Sie?

Analysieren Sie anschließend das erstellte Programm detaillierter. Stellen Sie sicher, dass Sie für dieses Projekt den default-Debugger der Codeblocks-IDE verwenden, indem Sie in der IDE im Menü *Debug → Active debuggers* den default-Debugger einstellen.

Verwenden Sie den Default-Debugger. Können Sie hiermit die exakten Speicheradressen der Register SODR, CODR, ODSR ermitteln? Welche Ursache kann ein entstehender Fehler haben?

Beantworten Sie die Fragen zu den jeweiligen Aufgaben im Fragenkatalog!

2. Zyklisches Verhalten

Die Datei Termin2Aufgabe2.c enthält Quellcode zum zyklischen An- und Ausschalten von LED DS1, die an den AT91-Controller links angeschlossen ist.

A)
Analysieren und kompilieren Sie den Programmcode mit der SourceNavigator-IDE. Debuggen Sie anschließend das Programm im Simulator. Versuchen Sie dabei die Konstante für *CYCLE_COUNT_MAX* so einzustellen, dass Sie einen Zyklus von ca. 1 Sekunde erreichen können.
Beachten Sie, dass die Zykluszeit individuell von Ihrer Hardware abhängig ist: In der Virtualisierungssoftware ist die CPU-Begrenzung für dieses Image per default ausgeschaltet. Die Leistung der VM kann also maximal 100% der Leistung Ihres Arbeits-PCs betragen.

B)
Kompilieren Sie nun Ihr Quellcode nochmals im Terminal ohne die HW-Quelldateien mit dem Befehl *arm-elf-gcc Termin2Aufgabe2.c -O0*.
Führen Sie das so erstellte Programm anschließend im Terminal mit dem Befehl *time arm-elf-run a.out* aus.

Beachten Sie dabei folgende Aspekte:
- Variieren Sie die Anzahl der Zyklen.
- Stimmt die Zykluszeit mit der aus Aufgabe a) in etwa überein? Korrigieren Sie ggf. die Konstante *CYCLE_COUNT_MAX* für 1 Sekunde nochmals.

C)
Lagern Sie den Code aus der vorherigen Aufgabe in eine *wait(unsigned int countValue)-Funktion* aus!

Kontrollieren Sie, ob die zuvor gewählte Konstante für die Zykluszeit von 1 Sekunde noch korrekt ist und passen sie ggf. wieder an.

Begründen Sie eventuelle Abweichungen!

D)
Definieren Sie weitere Konstanten als Zählwerte für verschiedene Zykluszeiten von 100ms, 10ms, 1ms, 100us, 10us, 1us!
Überprüfen Sie Ihre Annahmen für diese Werte praktisch durch die Auswertung der verschiedenen Ausführungszeiten Ihres Programms.

Beantworten Sie die Fragen zu den jeweiligen Aufgaben im Fragenkatalog!

3. Tastendruckverarbeitung – Polling

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 ausschaltet 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   "ParallelO_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. Meilenstein
Sie konnten die Fragen aus dem Fragenkatalog beantworten und konnten eventuelle Unklarheiten klären.