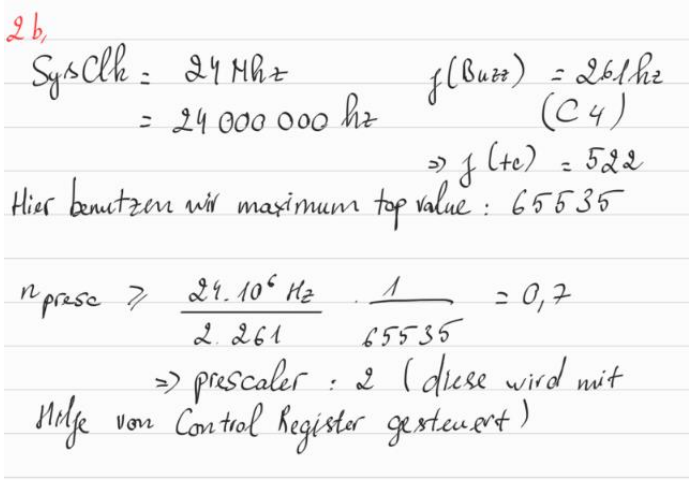
MPS Praktikum 4

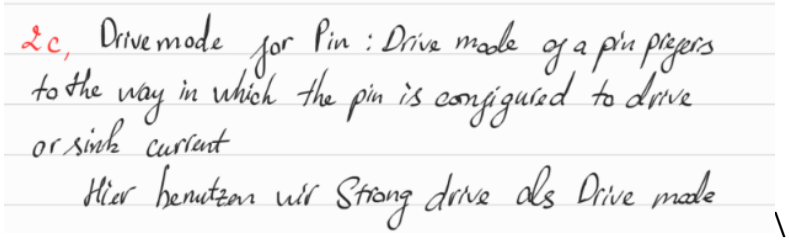
Trung Thieu Quang – 771043
Viet Anh Müller - 1111510

1. Bei üblichen Mikrocontrollern werden Timer und ähnliche SoC-Komponenten überschaltbare Prescaler angesteuert. Dies wird in der Schaltung im *TopDesign* simuliert
   a. Was bewirken die `FreqDiv_1` bis `FreqDiv_4`?
      Die Frequenz wird aufeinanderfolgenden dividiert, bei FreqDiv1 durch 2, FreqDiv_2, FreqDiv_3 und FreqDiv_4 durch 8

   b. Mit welchem Takt werden sie angesteuert?
      Bei FreqDiv_1 wird es mit 24Mhz angesteuert, 12Mhz bei 2, 1.5Mhz bei 3 und 0.1875Mhz bei 4
   c. Wie wird die `clock` des `Timer`s ausgewählt?
      Der Multiplexer leitet das Clocksignal an den Timer weiter. Dieser Multiplexer hat 4 Inputs, die Ausgänge von jedem Freqdiv sind. Das Auswählen von Input erfolgt durch den Control Register
      z.B: control[3:0] => SysClk/2 wird ausgewählt
   d. Welche Signalform erzeugt der Timer am Ausgang tc?
      Am Ausgang tc wird Signal in Form der Kippschwingung erzeugt
   e. Was für einen Signalform ist für eine einfache Tonerzeugung sinnvoll?
      Sinussignal
   f. Was bewirkt die FF-Schaltung? Welche Frequenz liegt am Ausgang q an?
      Wenn wir q und d eines D-flipflop durch ein NOT-gate verbinden, funktioniert dieser D-Flipflop als ein Toggle-Flipflop. Die Frequenz wird von diesem D-Flipflop durch 2 geteilt. Das heißt, f(q) = f(clk) / 2
   g. Wie wird da Signal am Pin_Buzz freigegeben?
      Das Signal wird von wird durch die Signale von q output von D-Flipflop UND Out_Ena Control Register gesteuert, wenn beide high sind, dann ist Pin_Buzz auch high
2. Planen Sie die effiziente Implementierung!
   a. Betrachten Sie die Tonfrequenzen in Header-Datei scale.h
   b. Berechnen Sie einen geeigneten Vorteiler für die gewünschten Frequenzen
      Hier haben wir versucht, aber irgendwie funktioniert diese Formel nicht im Programm, deshalb haben wir andere Werte in Codequelle probiert



c. Welche Drive Mode wählen Sie für Pin_Buzz, damit er den Buzzer gut treiben kann?



d. Nutzliche Makros

```
#include "scale.h"
#define SysClk 24000000
#define pre2 2      //f(tc) multiplier
#define topValue 1
#define prescaler 16

    /*Define your macro callbacks here */
    /*For more information, refer to the Writing Code topic in the PSoC Creator Help.*/


/* +++ TODO                +++
 *     nützliche Macros ...
 */

    void setTone(eTone p_tone) {
        Timer_WritePeriod(SysClk / (pre2 * prescaler * topValue) / p_tone);
        Out_Ena_Write(1);
    }
```

3. Starten Sie Psoc-Creator und laden Sie das Projekt Termin 4
   a. Suchen Sie die Pins für den Anschluss des Piezzo-Buzzer und lassen Sie sich die Schaltung abnehmen
   Die Pins können wir in Pin File finden

Name	Port	Pin	Lock
Pin_Buzz	P0[6]	78	☑
Rx_1	P2[0]	95	☑
Tx_1	P2[1]	96	☑

   b. Testen Sie zuerst den Prescaler über die Tastatureingaben ‘0‘ bis ‘3‘.

```
switch ( cRx ) {        // write Prescale Register value

    case '0':
        RegPrescaler_Write(0);
    break;
    case '1':
        RegPrescaler_Write(1);
    break;
    case '2':
        RegPrescaler_Write(2);
    break;
    case '3':
        //regVal = cRx - '0';     // erklären, wie das funktioniert!
    //basically, cRx is a character, in this case when we input 3, we can turn it to integer by substract with character '0'
        //RegPrescaler_Write( regVal );
        RegPrescaler_Write(3);
        break;
```

   Wir haben alle getestet, und die Tonfrequenz funktioniert nur mit Reg Prescaler 2 und 3, beim 3 ist der Ton sehr hoch, beim 0 und 1 ist die Frequenz sehr niedrig, deshalb können wir sogar jede herzt der Note
   c. Tastatur ein eines einfachen Klaviers:

```
            case 'z':
            {
                toneSettings(C5, 1./4);
                uint16 period = Timer_ReadPeriod();
                sprintf(buffer, "\n\r Periode: %d", period);
                UART_PutString(buffer);

                break;
            }
            case 'c':
                toneSettings(C4, 1./4);

                break;
            case 'd':
                toneSettings(D4, 1./4);
                uint16 period = RegPrescaler_Read();
                sprintf(buffer, "\n\r RegPrescaler: %d", period);
                UART_PutString(buffer);
                break;
            case 'e':
                toneSettings(E4, 1./4);
                break;
            case 'f':
                toneSettings(F4, 1./4);
                break;
            case 'g':
                toneSettings(G4, 1./4);
                break;
            case 'h':
                toneSettings(H4, 1./4);
                break;
            case 'a':
                toneSettings(A4, 1./4);
                break;
            case 'p':
                playSong();
                break;
            default:
                UART_PutChar( cRx );    // (unbenutzten) Buchstaben auf Bildschirm ausgeben
                break;
        } // end switch
        cRx = 0;                    // nicht vergessen. Warum?
```

```
void playTone(const sNoteSimple_t* psNote) {
    setTone(psNote->tone);
    SOUND_ON;
    CyDelay(psNote->duration*1000);
    SOUND_OFF;
}
void toneSettings(eTone tone, float dauer) {
    sNoteSimple_t tempNote;
    tempNote.duration = dauer;
    tempNote.tone = tone;
    playTone(&tempNote);
}
```

4. Spielen Sie mit der Firmware ein einfaches Lied ab. *Delay* darf für die Tondauer benutzt werden
    a. Ich habe eine Alternative für den Method play(), da ich wenige Erfahrung mit C habe , dennoch funktioniert es ohne Problem

```
void playSong(){
    int length = sizeof(harm)/sizeof(harm[0]);
    for(int i= 0; i < length; i++) {
        playTone(&harm[i]);
    }
}
```