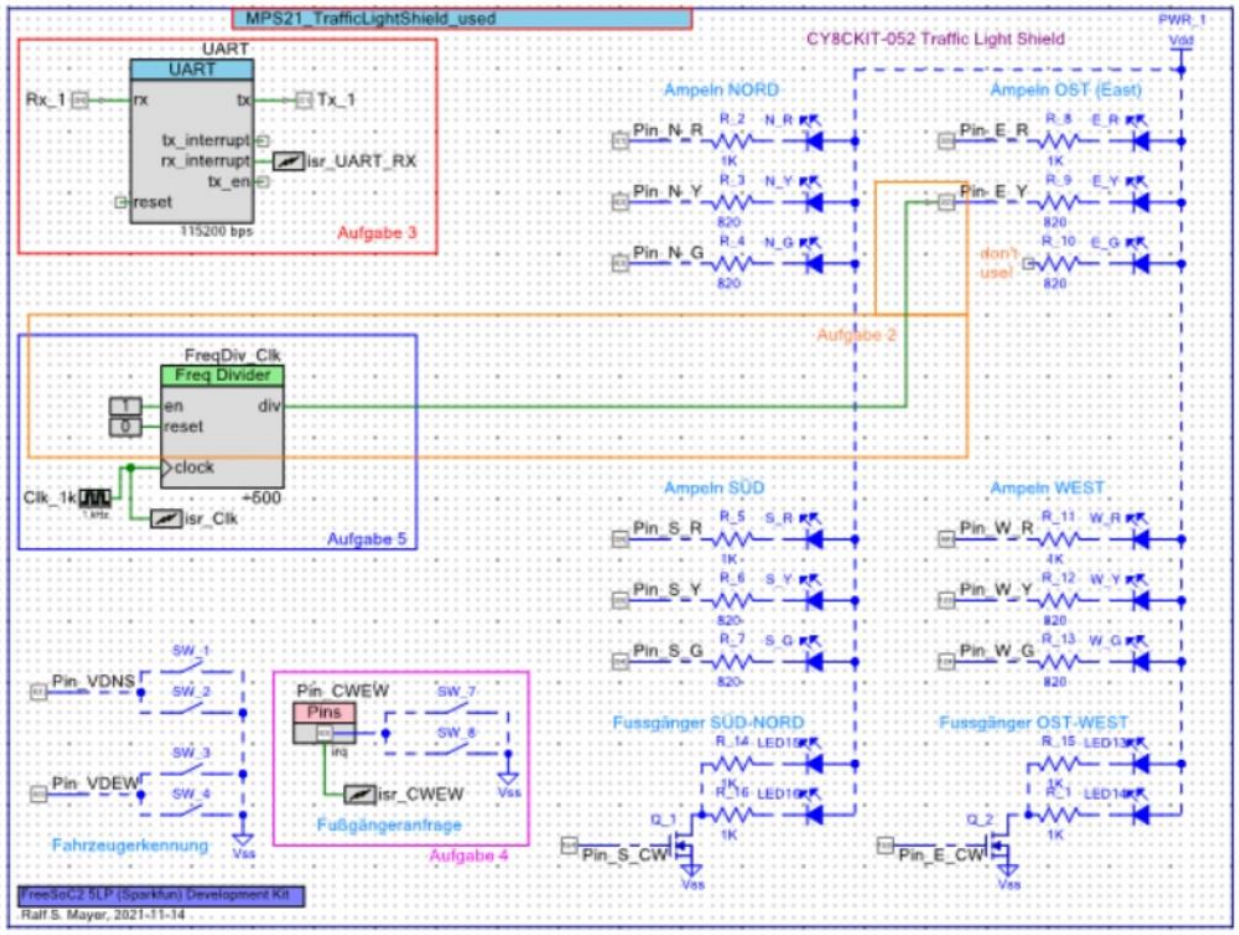
Praktikum 3

Lernziele: Verarbeitung von Ereignissen als Interrupts oder direkt in der Hardware (SoC)

1. Starten Sie PSoC-Creator und laden Sie das Projekt Termin 3: MPS21_Prakt_3.

a. Betrachten Sie das neue Hardware-Design in TopDesign.cysch.
(Nicht erschrecken: enthält jetzt alle Funktionen, die mit dem Traffic-Light-Shield und zusammen mit der UART möglich sind. Nur ein Teil wird Termin 3 benötigt.)



b. Starten Sie TeraTerm und verbinden Sie mit COM<xy> (Datei > Neue Verbindung >Seriell). Beachten Sie die UARTEinstellungen in TopDesign.cysch.

Einstellungen > Serieller Port:

- speed (baud): 115200
- data: 8
- parity: none
- stop: 1

Einstellungen > Terminal:

Übertragen: CR, Absenden: CR+LF

c. Betrachten Sie die HW-Verbindung von FreqDiv_Clk (div) mit Pin_E_Y. Welche Funktion hat FreqDiv_Clk (div) und wie ist Pin_E_Y konfiguriert?

Die FreqDiv_Clk teilt die Frequenz einer clock (Clk_1k) durch einen spezifizierten Wert (500) und gibt das Ergebnis als Takt an den Pin_E_Y weiter: 2Hz -> 2mal die Sekunde.

Der Pin wiederum verbindet über den Takt der FreqDiv einen Widerstand von 820 Ohm und eine gelbe Diode (E_Y) mit dem Stromfluss zu GND. Der Pin kann dann den Stromfluss zu der gelben Diode schließen und diese leuchtet.

d. Welche Funktion erfüllen die Komponenten isr_UART_RX, isr_Clk, und isr_CWEW?

isr_UART_RX, isr_Clk, isr_CWEW sind alle 3 Komponenten, die für definierte Interruptschnittstellen der Hardware zuständig sind.

Diese Schnittstellen unterbrechen, falls in der angeschlossenen Hardware eine Aktion / ein Event vorliegt, den normalen Programmabfluss um auf diese Events im Programm entsprechend eingehen zu können, und anschließend mit dem normalen Programm an entsprechender Stelle fortsetzen zu können.

isr_UART_RX: Interrupt Service bei Tastatureingaben
isr_CWEW: Interrupt nach Knopfdruck
isr_Clk: Interrupt Service nach Taktsignal

2. Betrachten Sie Pin_E_Y

a. Muss für die Funktion der LED E_Y etwas im Code konfiguriert und/oder initialisiert werden?

Ja. Es müssen 2 Makros für den Zustand an und aus (0u) und (1u) der LEDs definiert werden, um mit diesen dann entsprechend eine Stromzufuhr auf dem zugehörigen Pin des Stromkreises steuern zu können.

b. Mit welcher Frequenz blinkt die LED?

Clk_1k: 1kHz = 1000 Hz = 1000x pro Sekunde
1000x/sec * 60 = 60000x/min
60000x/min / 500 = 120x/min

Die LED blinkt 120-mal die Minute

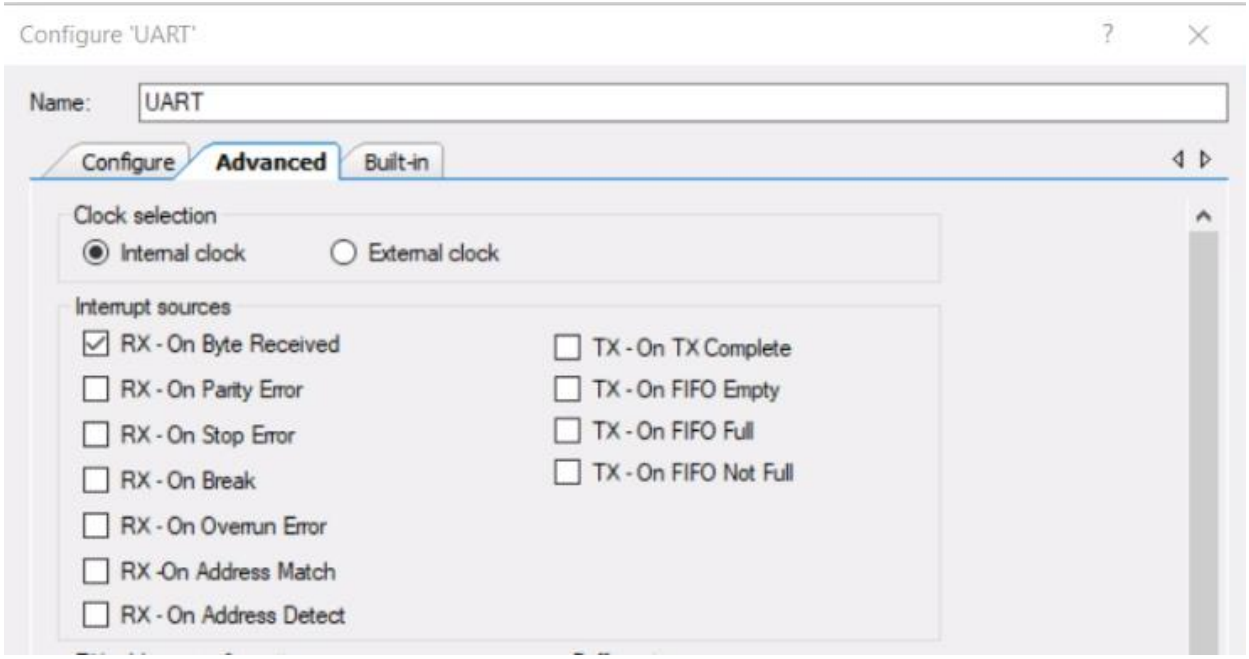
c. Wird das Blinken durch das Programm main in irgendeiner Weise beeinflusst? (Testen Sie dies, z.B. durch eine Delay im Programm)

Nein, die main feuert über eine Millionen Mal pro Sekunde – hier ist ein so geringer Einfluss vorhanden, dass dieser nicht spürbar ist.

3. UART-Interrupt

a. Welches Ereignis löst den UART-Interrupt aus? (Hinweis: UART-Komponente öffnen, Reiter advanced)

Der UART-ISR_RX Interrupt wird beim Ereignis OnByteReceived durch die ISR ausgelöst:



Hierfür wird eine Char Flag-Variable über MyIsrUartRX gesetzt und im loop behandelt.

b. Wo im Code ist die ISR (bereits vollständig) implementiert, und wo wird sie registriert?

Die ISR ist von ca. Zeile 64-67 bereits als Buttonpress auf MyIsrUartRX implementiert. Dabei wird eine Char Flag-Variable auf cRx gesetzt und im loop behandelt.

In Zeile ~57 wird über static char cRx dieser character registriert.

```
57  static char cRx = 0;                //!< character from UART, visible within main.c
58  /**
59   * UART RX interrupt service routine.
60   *
61   * Collect a characters from UART into a global input variable.
62   * @see cRx
63   */
64  CY_ISR( MyIsrUartRX )
65  {
66      cRx = UART_GetChar();       // read and copy char
67  }
```

c. Das Menu-Template reagiert jetzt nur auf den Interrupt, erklären Sie den Code

Die Eingabe eines chars wird immer durch den ISR_RX Interrupt aufgerufen und in cRx eingelesen. Nur diese chars werden in der Dauerschleife dann in der if Bedingung aufgegriffen und das Menü-Programm nur mit diesen betreten.

4. Button-Ereignis aus Interrupt MyIsrCWEW

a. Wo im Code ist die ISR (bereits vollständig) implementiert, und wo wird sie registriert?

Dieser Interrupt ist von ~Zeile 75 - 79 definiert:

```
69  static uint8_t fCWEW_Isr = 0;   //!< flag CW EW button isr, visible within main.c
70  /**
71   * Interrupt isr_CWSN for button Pin_CWSN interrupt service routine.
72   *
73   * @see fCWEW_Isr
74   */
75  CY_ISR( MyIsrCWEW ) // Wird bei Buttonpress ausgeführt
76  {
77      Pin_CWEW_ClearInterrupt();  // Clear Interrupt
78      fCWEW_Isr = 1;              // set flag
79  }
```

b. Toggeln Sie die LED Pin_E_CW wenn der Button gedrückt wird. Erweitern Sie dazu den vorgegebenen Code.

```
91  // Toggle LED
92  uint8_t toggleLED(uint8_t dword)
93  {
94      if(dword == LED_ON)
95          return LED_OFF;
96      else
97          return LED_ON;
98  }
```

```
168         // Behandlung Button-Ereignis aus ISR und LED toggle
169         if ( fCWEW_Isr ) {
170             Pin_E_CW_Write( toggleLED( Pin_E_CW_Read() ) );
171             fCWEW_Isr = 0;
172         }
```

c. Freiwillig: geben Sie ein Tonsignal auf Ihrem Terminal aus, wenn die Taste gedrückt wird. Hinweis: ASCII-Tabelle, Steuerzeichen!

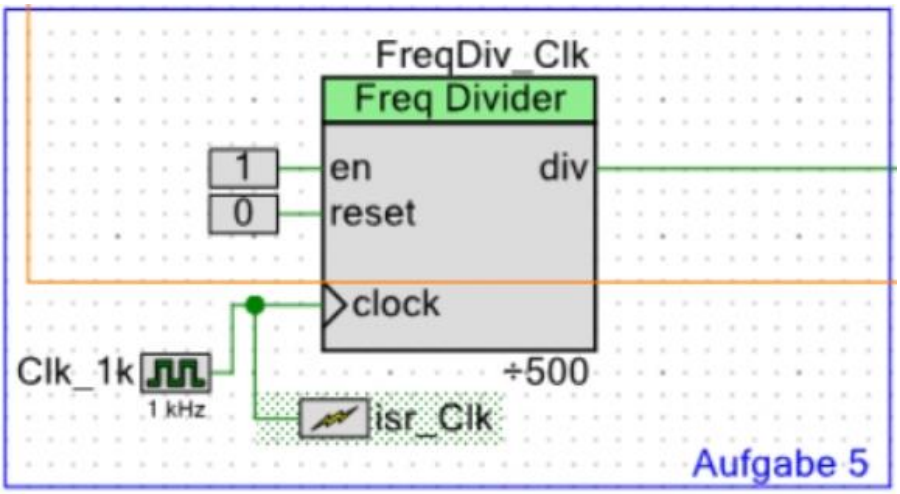
```
168         // Behandlung Button-Ereignis aus ISR und LED toggle
169         if ( fCWEW_Isr ) {
170             Pin_E_CW_Write( toggleLED( Pin_E_CW_Read() ) );
171             UART_PutChar(7);// Spiele Ton ab mit Steuerzeichen BEL (7)
172             fCWEW_Isr = 0;
173         }
```

Steuerzeichen BEL („BELL“) **wird zum** Ton ausgeben verwendet.

5. Timer-Ereignis isr_Clk

```
/**
 * Application clock interrupt service routine for isr_Clk
 *
 * @see fClock
 */
CY_ISR( IsrAppClk ) {
    // TODO: implementieren
    UART_PutString("App Clock");
}
```

a. Wie häufig wird dieser Interrupt ausgelöst?



Da die isr_Clock noch direkt an der Clk_1k Leitung **vor** dem FrqDiv_Clk liegt, fängt der Interrupt noch die 1kHz ab.
Hz zu Zeit: (1 / Hz)s => Ausgelöst jede 1/1000 Sekunde.

b. Wie könnte man die vergangene Zeit seit dem Start des Programms messen?

Man könnte eine Zählervariable hochzählen, wenn das Ereignis feuert und diese nach einer vergangenen Unit - Timestamp - Minute ausgeben.

Zeigen Sie diese über das Menu an.

```
static int clk_count = 1;
static uint8_t clk_flag = 0;
/**
 * Application clock interrupt service routine for isr_Clk
 *
 * @see fClock
 */
CY_ISR( IsrAppClk )
{
    clk_count++;
    if (clk_count % 1000 == 0)  // Reached 1000th hz call for 1/1000sec => 1 sec passed
    {
        clk_flag = 1;
        return;
    }
    clk_flag = 0;
}
```

c. Setzen Sie in der ISR alle Sekunden ein Flag, welches Sie in Ihrem main-Programm auswerten. Zeigen Sie die seit Programmstart vergangene Zeit kontinuierlich in ein und derselben Terminalzeile an. Hinweis: ASCII-Steuerzeichen!

```
164         // Behandlung CLK-Ereignis jede Sekunde
165         if ( clk_flag ) {
166             sprintf( buffer, "Time passed: %d sec.\r", clk_count / 1000);
167             UART_PutString( buffer );
168         }
```

Zweiter Aufgabenteil siehe bereits Aufgabe 5b.

e. Wie oft - schätzen Sie – wird die Endlosschleife in main pro Sekunde durchlaufen? Messen Sie! (Dazu ggf. c. und d. auskommentieren)

Schätzung: Ohne Interrupts wird die Endlosschleife jedes Mal in der Baudrate auf einen empfangenen char überprüfen, also 115200bps = 115200Hz = 115,0 kHz

Messung: ~800 kHz = ~800000Hz