Praktikum 2 MPS
Trung Thieu Quang – 771043
Viet Anh Müller - 1111510

1.

c. FreqDiv_Clk: dividiert Frequenz, um PIN_E_Y zu deaktivieren

d.

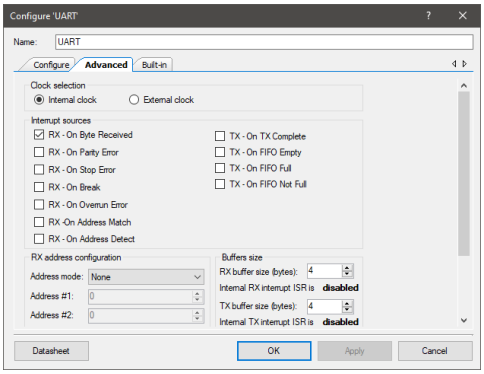
- isr_UART_RX: Receiver von interrupt signal
- isr_clk: Berechnen die clock für interrrupt signal (hier ist es 1 khz)
- isr_CWEW: jedes mal SW7 und SW8 gedrückt werden, wird PIN_CWEW zu high und isr_CWEW sendet ein interrupt signal

e. CySoftwareReset: einen Reset der Hardware erzwingen

2.

a. Die Funktion der LED E_Y muss nicht im Code konfiguriert werden, da alle schon im Hardware vorkommt
b. LED blinkt mit Frequency 1000/500 = 2HZ (2 mal pro Sekunde)
c. Nein, Frequency Divider hat das schon elerdigt, Clock vorgibt unabhängig von Programablauf den Takt

3.



a. Beim Eingabe von ein Char (1 Byte) wird Interrupt erlöst

b.

- implementierung in code

```
CY_ISR( MyIsrUartRX ) {
    cRx = UART_GetChar();       // read and copy char
}

static uint8_t fCWEW_Isr = 0;   //!< flag CW EW button isr, visible within main.c
```

- registierung

```
/* Initialize and start UART, register ISR */
UART_Start();                       // start UART
isr_UART_RX_StartEx( MyIsrUartRX ); // register ISR, enable itr uart
```

c. Der Variable cRx ist mit 0 initialisiert. Die ISR MyIsrUartRX wird registriert, damit Hauptprogramm nicht mehr für die Überwachung der UART zuständig wird. MyIsrUartRX wird im main aufgeruft und char cRx wird das Char von UART übernehmen. Falls cRx wahr ist (nicht 0), wird der Code in for-Schleife ausgeführt, in for-Schleife wird cRx wieder zu 0 zugewiesen, damit die if-Bedingung nicht ohne einen neuen Interrupt ausgeführt wird.

4.

a.

- Implementierung im Code

```
CY_ISR( MyIsrCWEW ) {
    /* Clear Interrupt first ! */
    Pin_CWEW_ClearInterrupt();
    fCWEW_Isr = 1;              // set flag
}
```

- Registierung im Code

```
    /* Initialize and start MyIsrCWEW, register ISR */
    isr_CWEW_StartEx( MyIsrCWEW );      // register CWEW itr
```

b. Toggeln die LED PIN_E_CW wenn der Button gedrückt wird

```
// Behandlung Button-Ereignis aus ISR
if ( fCWEW_Isr ) {
    // TODO: implementieren und LED Pin_E_CW toggeln!
    if(LED_CWEW) {
        Pin_E_CW_Write(1);      //weißE LEDs in OST_WEST ausschalten
        LED_CWEW = 0;
    }
    else {
        Pin_E_CW_Write(0);      ////weißE LEDs in OST_WEST anschalten
        LED_CWEW = 1;
    }
    fCWEW_Isr = 0;          //toggle end
}
```

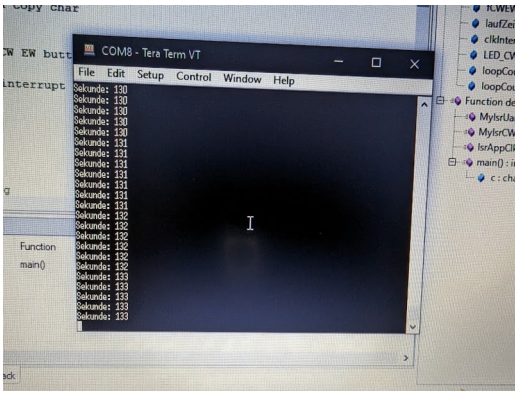
5. Timer-Ereignis isr_Clk

a. Dieser Interrupt wird 1000 mal pro Sekunde erlöst

b.

```
CY_ISR( IsrAppClk ) {
    // TODO: implementieren
    clkInterruptCounter++;
    if(clkInterruptCounter > 1000) {
        clkInterruptCounter = 0;
        laufZeit++;
        loopCountStart = 1;
    }
}
```

da der Interrupt 1000 mal pro Sekunde erlöst wird, wir können hier ein Counter (clkInterruptCounter) erstellen. Jedes mal das Counter 1000 reicht, das heißt, dass 1 Sekunde vorbei ist.



c. Durch den Menu können wir so die Information über die Laufzeit und Loop Counter ausdrücken

```
case 'm':   // so könnte man etwas steuern über die Konsole
    UART_PutString( "Menu-Beispiel: Taste 'm' gedrueckt\n\r" );
    break;
    case 'l':
    sprintf(buffer, "Sekunde: %d \n\r", laufZeit);
    UART_PutString(buffer);
    break;
    case 'c':
    sprintf(buffer, "Loop Counter: %d \n\r", loopCounter );
    UART_PutString(buffer);
    break;
```

e. Loop Counter: 412871
“● Programmable clocking
- 3- to 74-MHz internal oscillator, 1% accuracy at 3 MHz
- 4- to 25-MHz external crystal oscillator
- Internal PLL clock generation up to 80 MHz
- Low-power internal oscillator at 1, 33, and 100 kHz
- 32.768-kHz external watch crystal oscillator
- 12 clock dividers routable to any peripheral or I/O “ - datasheet