

Mikroprozessorsysteme, Praktikum 2, SS22

Student 1: Damir Maksuti

Matrikelnr: 765984

Student 2: Jamil Boujada

Matrikelnr: 769479

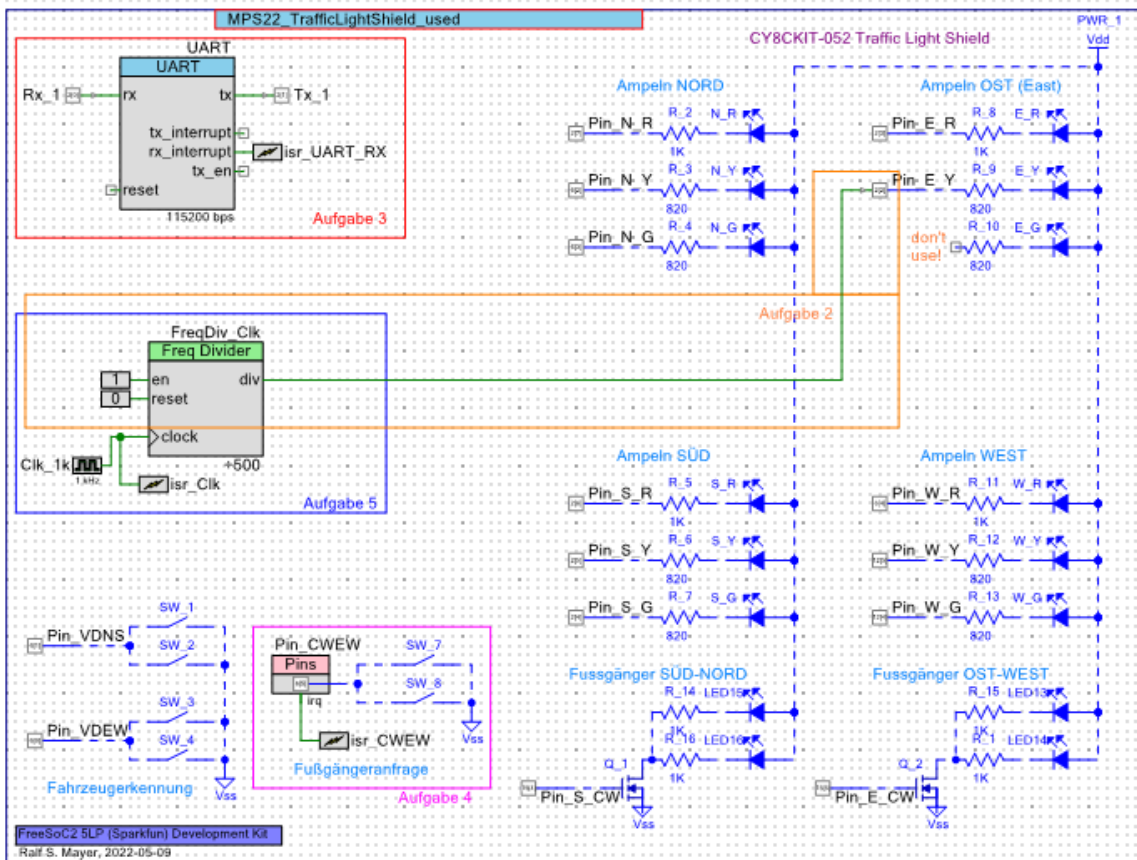


Bild 1 (Schaltplan)

Aufgabe 1

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?

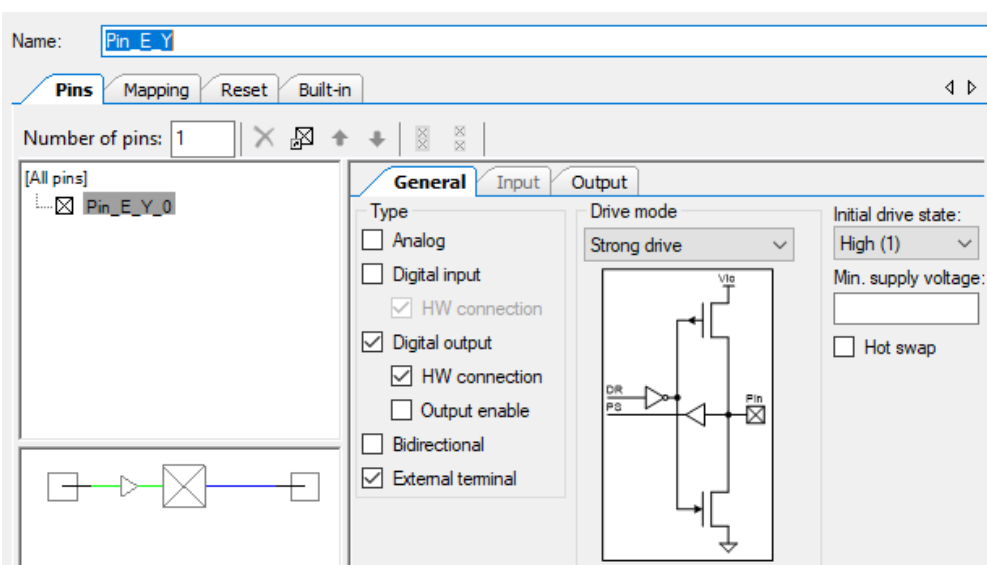


Bild 2 (Konfiguration Pins E_Y)

FreqDiv_Clk (div) ist Frequenzteiler (frequency divider). Seine Funktion besteht aus das Teilen des Eingangssignals und das Aussenden des geteilten Signals. Häufig werden hierzu T-Flipflops verwendet mit denen ein (Rückwärts-) Zähler realisiert werden kann. FreqDiv_Clk ist ein Output Gerät.

Input / Output Verbindungen Frequenzteilers sind:

en (Eingang/Input): Diese Verbindung aktiviert oder deaktiviert den internen Zähler des Frequenzteilers. Wenn auf diesem Eingang eine 0 („low“) liegt, werden bei der steigenden Flanke an Clock-Eingang, den internen Zähler nicht inkrementiert und die Ausgangsfrequenz nicht geändert.

reset (zurücksetzen): Diese Verbindung wird benutzt um den internen Zähler zurückzusetzen. Das passiert bei der nächsten steigenden Flanke an Clock-Eingang. Aufgrund dieses Prozesses, wird „div“-Ausgang automatisch auf „low“ gesetzt bis reset-Eingang wieder auf „low“ ist.

clock (Uhr): Der Clock-Eingang bestimmt das zu teilende Signal.

div (Teiler): An dieser Verbindung wird das geteilte Signal ausgesendet.

Die wichtigste Merkmale der Konfiguration des Pin_E_Y sind:

Initial drive state → high

HW connection → on

Output enable → off

(Siehe Bild 2)

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

Interrupt Service Routinen im Szenario:

isr_UART_RX führt die Benachrichtigung über den Erhalt eines Chars.

isr_Clk führt Interrupt-Signal, derer Frequenz mithilfe einer Uhr (Clock) und eines Frequenzteilers ausgelöst, kontrolliert und an Pin_E_Y und somit auch an die Kathode der Leuchtdiode (LED) E_Y gebracht wird.

isr_CWEW Interrupt wird ausgelöst, nachdem einer der Tasten (SW_7/SW_8) gedrückt wird und sich der Pegel auf „low“ ändert.

Aufgabe 2

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

Für die Funktion der LED_E_Y muss nichts im Code konfiguriert und/oder initialisiert werden, weil alle nötigen Konfigurationen an der Hardware vorgenommen werden können.

b) Mit welcher Frequenz blinkt die LED?

LED blinkt mit Frequenz von 2Hz. Das ergibt sich aus 1000Hz an Clock / 500 an Frequenzteiler.

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

Nein, da Clock unabhängig vom Programmablauf den Takt vorgibt.

Aufgabe 3

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

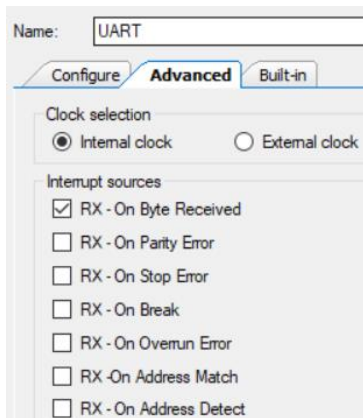


Bild 3 (Interrupt Sources der UART-Komponenten)

UART Interrupt reagiert auf Erhalt eines Chars (Siehe Bild 3).

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

```
45 | CY_ISR( MyIsrUartRX ) {  
46 |     cRx = UART_GetChar();  
47 | }
```

Bild 4 (Implementierung der UART-ISR)

```
93 |     isr_UART_RX_StartEx( MyIsrUartRX ); // register ISR, enable itr uart
```

Bild 5 (Registrierung der UART-ISR)

Die ISR der UART-Komponente ist im Code in Linien 45,46 und 47 implementiert und in Linie 93 registriert (Siehe Bilder 4 und 5).

c) Das Menu-Template reagiert jetzt nur auf den Interrupt, erklären Sie den Code. Die Variable „cRx“ ist mit 0 (Null) initialisiert. Damit für die Überwachung der UART nicht mehr das Hauptprogramm zuständig wird, wird die „ISR MyIsrUartRX“ registriert und implementiert. „MyIsrUartRX“ wird in main aufgerufen und static Char „cRx“ wird das Char von UART übernehmen. Falls „cRx“ wahr ist (nicht eine Null) wird innerhalb von for-Schleife if-Bedingung ausgeführt. Nachdem „cRx“ an char „c“ abgegeben wird, wird „cRx“ wieder eine Null zugewiesen damit if-Bedingung nicht ohne einen neuen Interrupt erfüllt wird.

d) Erweitern Sie später das Menu nach Bedarf für die folgenden Aufgaben oder für Ihre Tests.

```
126 // +++ Menuesteuerung +++
127 switch ( c ) {
128     case 'q':
129     case 'Q': // exit
130         CySoftwareReset();
131         break;
132     case 'z': // zeitabfrage
133         sprintf(buffer, "Seconds: <%d> \r", secondsSinceStart);
134         UART_PutString(buffer);
135         break;
136     case 'l':
137         sprintf(buffer, "Loop Count: <%d> \n\r", loopCount);
138         UART_PutString(buffer);
139         break;
140     default:
141         sprintf(buffer, "invalid Input: <%c>", c);
142         UART_PutString( buffer ); // Buchstabe auf Bildschirm ausgeben
143         break;
144 } // end switch
145 } // end if cRx
```

Bild 6 (Menüsteuerung)

Aufgabe 4

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

```
49 static uint8_t fCWEW_Isr = 1; //!< flag CW EW button isr, visible within main.c
50 /**
51  * Interrupt isr_CWSN for button Pin_CWSN interrupt service routine.
52  *
53  * @see fCWEW_Isr
54  */
55 static int CWEW_LED = 0;
56 // Fußgängeranfrage
57 CY_ISR( MyIsrCWEW ) {
58     /* Clear Interrupt first ! */
59     Pin_CWEW_ClearInterrupt();
60     fCWEW_Isr = 1; // set flag
61 }
```

Bild 7 (MyIsrCWEW-Implementierung)

```
95 /* Initialize and start MyIsrCWEW, register ISR */
96 isr_CWEW_StartEx( MyIsrCWEW ); // register CWEW itr
97 Pin_CWEW_ClearInterrupt(); // clear eventual interrupt
98 isr_CWEW_ClearPending(); // clear eventual pending interrupt
```

Bild 8 (MyIsrCWEW-Registrierung)

MyIsrCWEW ist im Code in Linien 49,55,57 bis 61 implementiert und in Linie 96 registriert (Siehe Bilder 7 und 8).

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

```
152 // Behandlung Button-Ereignis aus ISR
153 if ( fCWEW_Isr ) {
154     // TODO: implementieren und LED Pin_E_CW toggeln!
155     if(CWEW_LED){
156         Pin_E_CW_Write(1);
157         CWEW_LED = 0;
158     } else {
159         Pin_E_CW_Write(0);
160         CWEW_LED = 1;
161     }
162     fCWEW_Isr=0;
163 }
```

Bild 9 (LED Pin_E_CW Toggeln - Code)

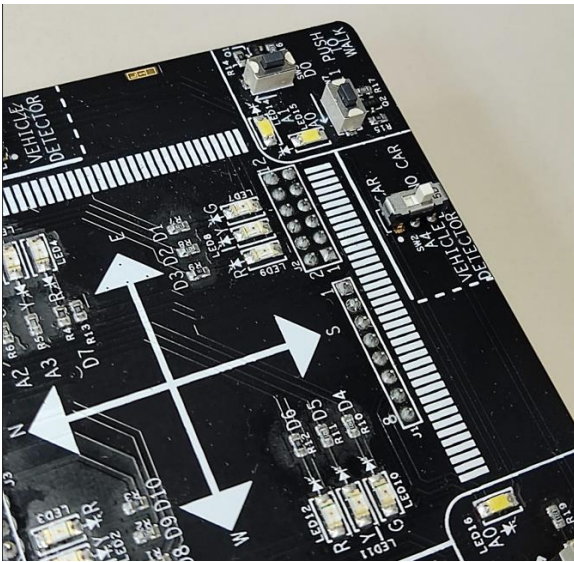


Bild 10 (LED Pin_E_CW aus)

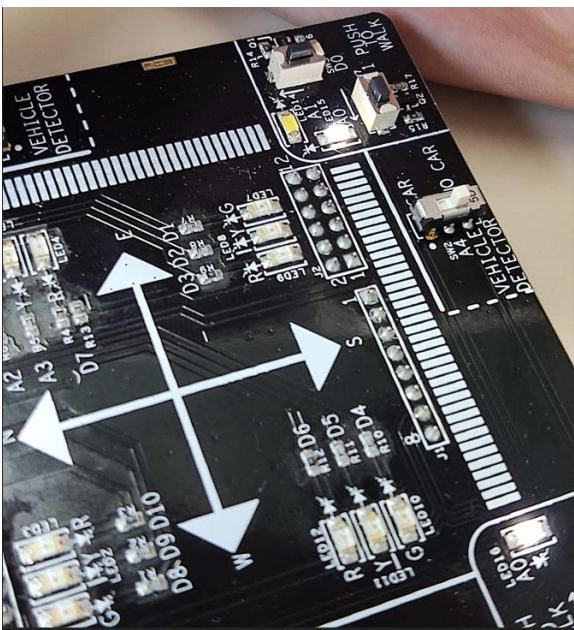


Bild 11 (LED Pin_E_CW an)

Aufgabe 5

a) Wie häufig wird dieser Interrupt ausgelöst?
Interrupt wird mit Frequenz von 1000Hz ausgelöst.

b) Wie könnte man die vergangene Zeit seit dem Start des Programms messen?
Zeigen Sie diese über das Menu an.

```

72 // Sekundentakt
73 CY_ISR( IsrAppClk ) {
74     // TODO: implementieren
75     clkInterruptCounter++;
76     if(clkInterruptCounter == 1000){
77         clkInterruptCounter = 0;
78         secondsSinceStart++; // setzen des Flags
79         newSecondHasStarted = 1; // start loop iteration
80     }
81 }

```

Bild 12 (Zeitmessung - Code)

Frequenz zeigt wie schnell bei einem periodischen Vorgang die Wiederholungen aufeinander folgen. Die Frequenz eines sich regelmäßig wiederholenden Vorgangs ist definiert als der Kehrwert der Periodendauer.

$1\text{Hz} = 1\text{s}^{-1}$ --> eins pro Sekunde <--> $1000\text{Hz} = 1000\text{s}^{-1}$ --> tausend pro Sekunde

Weil die „isr_Clk“ mit Frequenz von 1000Hz ausgelöst wird, bedeutet das, dass wenn „clkInterruptCounter“ Wert von 1000 hat, eine Sekunde vorbei ist. Darauf basiert, ist es möglich die Zeit seit Programmstart zu messen durch das Inkrementieren der Variable „secondsSinceStart“ immer wenn „clkInterruptCounter“ den Wert von 1000 hat (Siehe Bild 12, Zeile 78). Nachdem die Variable „secondsSinceStart“ inkrementiert wird, wird die Variable „clkInterruptCounter“ wieder auf 0 (Null) gesetzt. Die Ausgabe auf Terminal, wird mithilfe von Switch-Case (Menü) durchgeführt. Um die vergangene Zeit auf Terminal auszugeben, muss die Taste „z“ gedrückt werden (Siehe Bild 6, Zeile 132).

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!*

```
112 |         for(;;)
113 |         {
114 |             /*
115 |              * Character aus Uart ISR abfragen
116 |              * Beispiel für einfache Menuesteuerung
117 |              * Achtung: Für UART in diesem Programm ist ein zusätzlicher FTDI USB-TTL-Adapter notwendig
118 |              */
119 |             sprintf(buffer, "Seconds: <%d> \r", secondsSinceStart);
120 |             UART_PutString(buffer);
121 |             if ( cRx ) {
122 |                 char c = cRx; // copy char
123 |                 cRx = 0;      // nicht vergessen!!! Warum?
124 |                 UART_PutString("\n");
125 |             }
126 |         }
```

Bild 13 (Kontinuierliche Zeitausgabe – Code, Zeilen 119 und 120)

„\r“ --> wird benutzt um Cursor am Anfang der aktuellen Zeile zu bewegen.

(Siehe Bild 12, Zeile 78 und Bild 13).

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

Die ISR am Timer unterbricht den Programmfluss 1000mal pro Sekunde und führt ihn temporär an einer anderen Stelle fort. Wir vermuten, dass dadurch wenige Schleifendurchgänge möglich sind als ohne Messung. Wir schätzen 600 Iterationen.

```
147 |         // Count loops in 1 sec
148 |         if( newSecondHasStarted && secondsSinceStart < 2 ){
149 |             loopCount++;
150 |         }
```

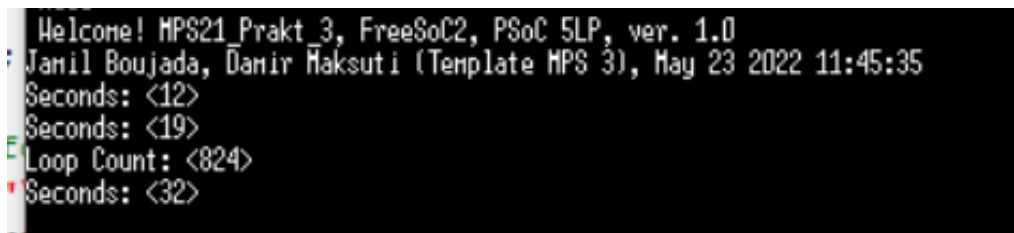
Bild 14 (Das Messen der Anzahl der Iterationen innerhalb einer Sekunde - Code)

Damit die Anzahl der „for-Loop“ Iterationen innerhalb einer Sekunde berechnet wird, muss der Anfang und das Ende der Sekunde bekannt sein.

Wenn die Variable „clkInterruptCounter“ das erste Mal den Wert 1000 hat, wird die Variable „newSecondHasStarted“ auf 1 (eins) gesetzt. Somit ist der Anfang der neuen Sekunde bekannt (Siehe Bild 12, Zeile 79).

Innerhalb von „for-Loop“ befindet sich eine „if-Anweisung“ (Siehe Bild 14) die wahr wird, nur wenn zwei Bedingungen wahr sind, und zwar wenn, die Variable „newSecondHasStarted“ Wert 1 (eins) hat und die Variable „secondsSinceStart“ < 2 ist. Immer wenn diese zwei Bedingungen wahr sind (die zweite Sekunde nachdem Programmstart ist angefangen und ist noch nicht beendet), wird die Variable „loopCount“ inkrementiert und somit die Anzahl der Iterationen berechnet. Wenn die Variable „secondsSinceStart“ ≥ 2 ist, bedeutet das, dass die Sekunde innerhalb der, die Anzahl der Iterationen berechnet wird abgelaufen ist, und somit wird die Berechnung nicht mehr ausgeführt.

Um Anzahl der Iterationen innerhalb einer Sekunde auszugeben, muss die Taste „l“ gedrückt werden (Siehe Bild 6, Zeile 136).

A screenshot of a terminal window with a black background and white text. The text shows the program's output: a welcome message, the names of the authors (Jamil Boujada and Damir Maksuti), the date and time (May 23 2022 11:45:35), and then three lines of data: 'Seconds: <12>', 'Seconds: <19>', and 'Loop Count: <824>'. The last line is 'Seconds: <32>'.

```
Welcome! MPS21 Prakt 3, FreeSoC2, PSoC 5LP, ver. 1.0
Jamil Boujada, Damir Maksuti (Template MPS 3), May 23 2022 11:45:35
Seconds: <12>
Seconds: <19>
Loop Count: <824>
Seconds: <32>
```

Bild 15 (Die Anzahl der Iterationen innerhalb einer Sekunde mit Ausgabe auf Terminal)

Ohne Ausgabe auf Terminal ist der Wert der Iterationen pro Sekunde ungefähr 775000.

Hinweis:

Leider haben wir den Screenshot der Anzahl der Iterationen „for-Loops“ ohne Ausgabe auf Terminal verloren.