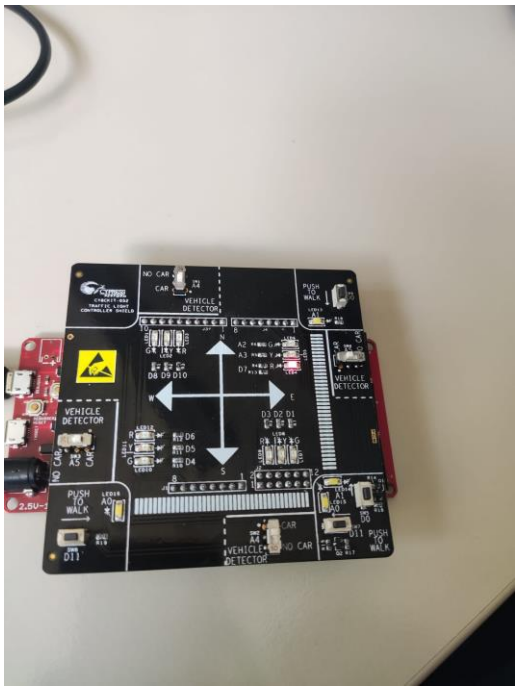
Mikroprozessorsysteme, Praktikum 1, SS22

Student 1: Damir Maksuti
Matrikelnr: 765984
Student 2: Jamil Boujada
Matrikelnr: 769479

1. Starten Sie PSoC-Creator und laden Sie das Projekt Termin 1: MPS22_Prakt_1.

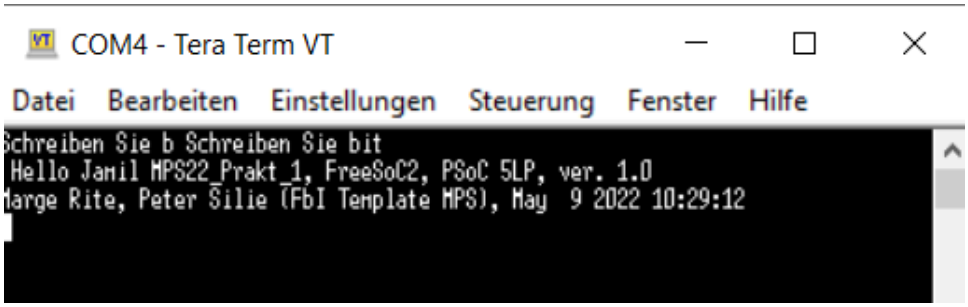


Handy Foto (Rote LED leuchtet)

2.Erzeugen Sie eine einfache Bildschirmausgabe: Formatierte Ausgabe (von Zahlen, char, strings, ...)

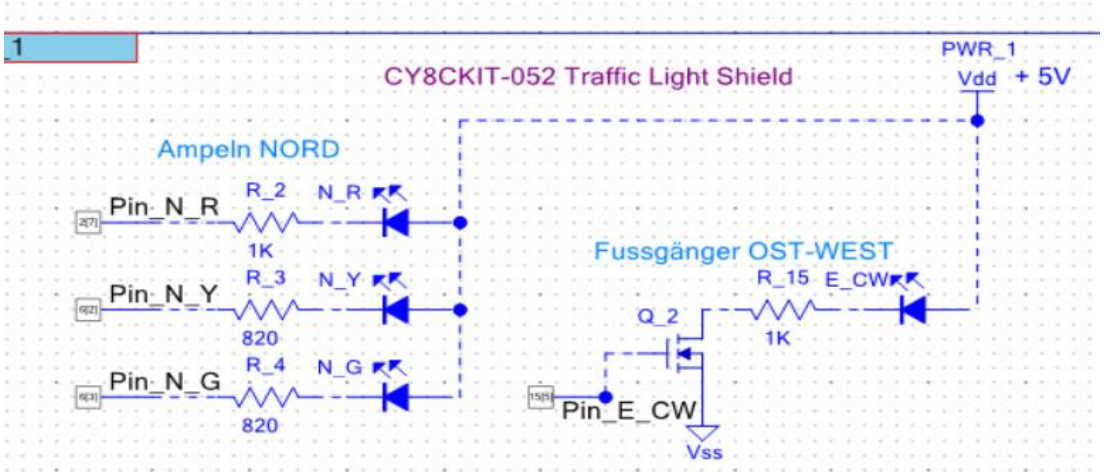
Hinweis: es hat sich als ressourcenschonender für den PSoC erwiesen, statt des bekannten printf(“<fmt>“, var, [...]);
eine formatierte Ausgabe in einen genügen großen string `buffer` zu machen.
`sprintf( buffer, ( "<fmt>", var, [...] );` und diesen dann mit
`UART_PutString( buffer );` auszugeben

```
/* Initialisierung */
UART_Start();                          // start UART

CyGlobalIntEnable; /* Enable global interrupts. */

/* Welcome and info text Ausgabe */
sprintf( buffer, "\n\r Je te souhaite une bonne journee Jamil %s\n\r", _VERSTR_);
UART_PutString( buffer );                              // Ausgabe auf UART
```



(Die Namen haben wir in der main.c geändert)

3.Vorbereitung: Betrachten Sie das TopDesign. Fenster zeigt Schaltplan mit Komponenten:



UART (Pins 2[0] und 2[1] für RX und TX),

vier LEDs: (Pin_N_R 2[7], Pin_N_Y 6[2], Pin_N_G 6[3] und Pin_E_CW 15[5]) als Output,
einen Button (Pin_CWEW 6[5]) als Input.

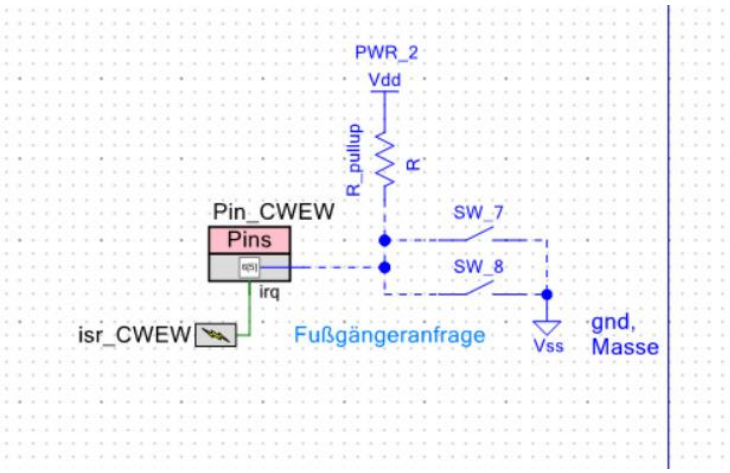
3a. Überlegen Sie, wann die LEDs R, Y, G leuchten? Active high oder low?

Je größer die Potenzialdifferenz zwischen zwei Punkten, desto größer der Strom, der durch die LEDs N_R /Y / Z durchfließt. Daher muss an den Pin_N_R / Y / Z ein Low anliegen,

3b. Überlegen Sie, wann LED_CW leuchtet? Transistor Q_2 wirkt als Inverter!

Der Transistor sperrt, wenn an PIN_E_CW kein high anliegt. Sobald ein High anliegt wirkt der Transistor leitend und verbindet damit VSS mit VDD. Die LED_CW leuchtet nun.

3c. Welchen Pegel nimmt Pin_CWEW an, wenn SW_7 bzw. SW_8 nicht gedrückt ist
und wenn gedrückt ist.



Sofern die Schalter nicht gedrückt sind, wird Pin_CWEW auf High gezogen. Sobald eines der schalter gedrückt ist wird Pin_CWEW auf Masse gezogen.

4. Lesen vom Zeichen vom Terminal, Menu, Schalten der LED(s) über Terminaleingaben

4a. Lesen Sie Zeichen von der UART: `UART_GetChar()`

4b. Schreiben Sie ein einfaches Menu, welches durch die Eingabezeichen gesteuert wird.

4c. Steuern Sie mit den Zeichen das Verhalten der LED(s) (s. 3, An- u. Ausschalten)

```
uint8_t chr = 0;                        // for char von der UART

/* +++ Applikationsschleife         +++
   +++ wird ständig durchlaufen     +++ */
for(;;)
{
    /* +++ Character aus Uart mit Polling abfragen +++ */
    chr = UART_GetChar();               // fragt UART nach empfangene char ab
                                        //  = 0: wenn nichts empfangen
                                        // != 0: empfangenes Zeichen
                                        // Polling, nicht wartend
    // chr auswerten falls != 0
    if ( chr != 0 ) {
        //  z.B. grüne LED ein- 'G' und 'g' ausschalten
        switch (chr) {
            case 'R':
                Pin_N_R_Write(LED_ON);
                break;
            case 'Y':
                Pin_N_Y_Write(LED_ON);
                break;
            case 'G':
                Pin_N_G_Write(LED_ON);
                break;
            case 'r':
                Pin_N_R_Write(LED_OFF);
                break;
            case 'y':
                Pin_N_Y_Write(LED_OFF);
                break;
            case 'g':
                Pin_N_G_Write(LED_OFF);
                break;
            default:
                sprintf( buffer, "valid characters are R-r,Y-y,G-g, but input was <%c> \n\r", chr );
                UART_PutString( buffer );
                break;
        } // end switch
        chr = 0;            // nicht vergessen (mit ISR)
```

```
71                                          // Polling, nicht wartend
72          // chr auswerten falls != 0
73          if ( chr != 0 ) {
74              //  z.B. grüne LED ein- 'G' und 'g' ausschalten
75              switch (chr) {
76                  case 'R':
77                      Pin_N_R_Write(LED_ON);
78                      break;
79                  case 'Y':
80                      Pin_N_Y_Write(LED_ON);
81                      break;
```

5.Taste CWEW abfragen und Aktion, toggeln der weißen LEDs überprüfen

a. Wird nur unzuverlässig funktionieren, weil die Taste prellt!
(Prellen bedeutet, dass der mechanische Kontakt nicht sofort zuverlässig den neuen Zustand einnimmt, und anfangs mehrmals zurückspringt)

Wir könnten erkennen, dass der Tastenklick nur teilweise funktioniert hatte.

b. Wie könnte man per Software entprellen?

Indem man den Programmablauf künstlich verzögert. Wenn Komponenten der Hardware (oder externe Systeme) mehr Zeit brauchen bis sie aktiv sind, dann muss das Programm dies berücksichtigen. Möglich wäre ein Delay oder eine eingebaute Schleife.

```
/* +++ Taste CWEW abfragen +++ */
if ( Pin_CWEW_Read() == 0 ) {    // warum auf 0 abfragen?
    while( !Pin_CWEW_Read() );
    Pin_E_CW_Write( !Pin_E_CW_Read() ); // toggle
}
```

Die 0 ist in der ASCII Tabelle ein Zeichen dafür, dass nichts empfangen wurde.

6.Aktivieren Sie das Blinken der roten LED durch Entfernen der Kommentarzeichen ‚//'.
Funktion: an ... warten ... aus ... warten.
Das Warten wird mit Schleifen realisiert (nicht ändern!).

a. Kompilieren Sie mit Debug und testen Sie Funktion.

Im Debug mode konnten wir uns durch den Code debuggen (mit breakpoints).Der Code hatte funktioniert.

b. Kompilieren Sie jetzt mit Release und testen Sie Funktion. Warum geht es jetzt nicht mehr? (es scheint, als ob die Schleifen nicht ausgeführt werden)

Die While loop wurde durch den compiler wegoptimiert.

c. Erinnern Sie sich an Rechnerarchitektur und Optimierung des Codes durch den Compiler. (wird die Variabel ‚loop' denn gebraucht?)

Im Release Mode wird der Code optmiert. Hierbei verändert der Compiler den Code.

Beispiele hierfür aus Rechnerarchitektur: die Addition 4+4+4+4 wird zu 2^4 (LSL) oder das Laden einer Variablen aus dem RAM muss für den Compiler nicht zwingend in einer Loop geschehen (wenn das Register nicht verändert wird). Er wird folglich nur 1mal lesend auf den Speicher zugegriffen. Das externen Faktoren z.B. ein Buttonklick, Tastaturinput Werte im Speicher ändern können, weiß der Compiler nicht.

Recherchieren Sie in C-Referenzen nach ‚volatile'. Verwenden und testen Sie Volatile

Volatile sorgt dafür Werte immer aus dem Speicher neugeladen werden

```
    /* +++ LED blinken lassen, warten mit Schleife +++ */
    // für Teilaufgabe 6 Kommentare entfernen
    // Testen mit DEBUG und RELEASE
   volatile uint32_t loop;                  // geht mit DEBUG, aber nicht mit RELE
                                          // warum???
                                            // Frage, was macht 'volatile'?
    Pin_N_R_Write( LED_ON );          // anschalten
    for ( loop = 0; loop < 1000000; loop++ )     // warten mit Schleife
        ;
    Pin_N_R_Write( LED_OFF );         // ausschalten
    for ( loop = 0; loop < 1000000; loop++ )     // warten mit Schleife
        ;

} // end for

' end main
```

7. Testen Sie jetzt mit 6. nochmal die Funktion der Keyboard-Eingabe und des Buttons a. Sind Sie mit der Schnelligkeit der Reaktion auf Terminal-Eingabe zufrieden? b. Funktioniert der Button überhaupt noch zuverlässig? c. Warum ist das so? Könnte man so eine Lösung erfolgreich verkaufen?

Das Ampelsystem funktionierte bei uns problemlos. Eingaben führten schnell zum gewünschten Ergebnis. Lediglich die Buttons für die Fußgängeranfrage mussten 1-2 Sekunden lang gedrückt werden, damit die weißen LEDs an oder aus gingen.

Im Alltag muss ich allerdings auch ,,bewusst" (also 1-2 Sekunden) den Pushbutton drücken, um die Fußgängeranfrage auslösen zu können. Daher denke ich, dass es nicht wirklich schlimm ist. Wir vermuten, dass die Schleifen den Vorgang ein bisschen abbremsen.