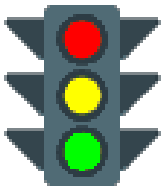
schule-on-line Mikrocontroller Projekt Ampel	**Projekt** **Ampel** **mit AT89C51CC03**	Name: Datum: Fach: Datei	 2021-03-15 ueb_ampel_89c51asm /hoe

from https://schule-on-line.de →Kurse →Mikrocontroller

Pflichtenheft:

- Es soll eine einfache Verkehrs-Ampel aufgebaut werden
- Die Zeiten für einzelnen Schaltphasen sollen einfach anpassbar sein
- Das Projekt soll mit einem AT89C51CC03 in Assembler realisiert werden



Step1: Ampel-LEDs

Die LEDAmpel soll an den Port **P1** angeschlossen werden:

Port1.2 ROT
Port1.1 GELB
Port1.0 GRUEN

Port 1	7	6	5	4	3	2	1	0

`#define AMPEL P1`
 	
`;NULL-aktiv // EINS aktiv`
`#define ROT 3 ; 4`
`#define ROTGELB 1 ; 6`
`#define GRUEN 6 ; 1`
`#define GELB 5 ; 2`								

Step2: Schaltzeiten

Die Schaltzeiten in Minuten bzw. Sekunden werden wie folgt festgelegt:

```
#define    WAIT_SEC      R5       ; verwende Register5 als Zeitwert

#define    ZEIT_ROT      6        ; sec zur Simulation „echt“ Minuten
#define    ZEIT_ROTGELB  1
#define    ZEIT_GRUEN    4
#define    ZEIT GELB     1
```

Step3: Hauptprogramm

```
Ampel:      MOV  AMPEL,#ROT
            MOV  WAIT_SEC,#ZEIT_ROT
            CALL UPG_WAIT_SEC
            MOV  AMPEL,#ROTGELB
            MOV  WAIT_SEC,#ZEIT_ROTGELB
            CALL UPG_WAIT_SEC
            MOV  AMPEL,#GRUEN
            MOV  WAIT_SEC,#ZEIT_GRUEN
            CALL UPG_WAIT_SEC
            MOV  AMPEL,#GELB
            MOV  WAIT_SEC,#ZEIT_GELB
            CALL UPG_WAIT_SEC
            LJMP Ampel
```

Step4: Unterprogramme

Weil Zeit-Unterprogramme immer wieder benötigt werden, ist es nützlich sie möglichst universell zu programmieren, so dass man Sie immer wieder verwenden kann. *Ideal ist, wenn ein Unterprogramm KEIN Register verändert.*

```
; dieses UPG macht einen Delay von genau 1.00 ms
delay_01ms:  PUSH 7          ; legt Register 7 auf den Stack
             MOV  R7,#255    ; 255 x 6 x „Nichtstun“ dauert genau 1 ms
delayloop0:  NOP
             NOP
             NOP
             NOP
             NOP
             NOP
             DJNZ R7,delayloop0
             POP  7         ; holt den ursprünglichen Wert von Register7 zurück
             RET

; dieses UPG macht einen Delay von genau 10.0 ms
delay_10ms:  PUSH 7          ; legt Register 7 auf den Stack
             MOV  R7,#10     ; ruft 10 mal das UPG für 1 Millisec auf
delayloop1:  CALL delay_01ms
             DJNZ R7,delayloop1
             POP  7         ; holt den ursprünglichen Wert von Register7 zurück
             RET

; dieses UPG macht einen Delay von genau 1.000 sec
delay_1sec:  PUSH 7          ; legt Register 7 auf den Stack
             MOV  R7,#100    ; ruft 10 mal das UPG für 10 Millisec auf
delayloop10: CALL delay_10ms
             DJNZ R7,delayloop10
             POP  7         ; holt den ursprünglichen Wert von Register7 zurück
             RET

; dieses UPG macht einen Delay von genau von 1 ms bis 65535 ms=16Bit
; der Verzögerungswert steht in R6 HIGH    R5 LOW
; Bsp:    MOV R6,#60000 SHR 8  ; MOV R5,#60000 MOD 256 ; „wartet“ 1 min
delay_R65ms: SJMP dmsloop1ms
dmsloopR5ms: MOV  R5,#255
dmsloop1ms:  CALL delay_01ms
             DJNZ R5,dmsloop1ms
             DJNZ R6,dmsloopR5
             RET

; dieses UPG macht einen Delay von genau  0.01 sec bis 255.99 sec
; der Verzögerungswert steht in R6 Sek 1..255   R5 zehntelsec 0..99
; Bsp:    MOV R6,#3  ; MOV R5,#50   ;  „wartet“ 3.5 sec
delay_R65sz: SJMP dmsloop1sz
dmsloopR5sz: MOV  R5,#99
dmsloop1sz:  CALL delay_01ms
             DJNZ R5,dmsloop1ms
             DJNZ R6,dmsloopR5
             RET

; dieses UPG macht einen Delay von genau  1 sec bis 255 min (> 4 Std!!)
; der Verzögerungswert steht in R6=Min=1..255   R5=Sekunde=0..59
; Bsp:    MOV R6,#45  ; MOV R5,#0    ; „wartet“ 45 min 0 sec
UPG_WAIT_SEC: MOV R6,#1
delay_R6m5s:  SJMP dmsloopsec
dmsloopmin:   MOV  R5,#59
dmsloopsec:   CALL delay_1sec
              DJNZ R5,dmsloopsec
              DJNZ R6,dmsloopmin
              RET
```

Step5: **Assembler-Programm fuer 89C51CC03**

```
01:   $NOMOD51                      ; vergesse alle Standard 8051 Definitionen
02:   $include(at89c51cc03.inc)     ; binde die Definitionen fuer MEINE CPU ein
03:
04:                         LJMP Start     ; Springe zum Start
05:
06:                         ORG 0100h      ; Hauptprogramm soll ab 0100h beginnen, kein Konflikt mit Interrupts
07:   Start:
08:   WAIT_SEC              EQU   R5       ; verwende Register5 als Zeitwert
09:
10:   #define               ZEIT_ROT     6 ; sec zfuer Ampel-Phasen hier SEC -> "echt" Minuten
11:   #define               ZEIT_ROTGELB 1
12:   #define               ZEIT_GRUEN   4
13:   #define               ZEIT_GELB    1
14:
15:   #define               PortAmpel    P1
16:
17:                                      ;NULL-aktiv // EINS aktiv
18:   #define               ROT      3     ; 4
19:   #define               ROTGELB  1     ; 6
20:   #define               GRUEN    6     ; 1
21:   #define               GELB     5     ; 2
22:
23:   Ampel:                MOV  PortAmpel,#ROT
24:                         MOV  WAIT_SEC,#ZEIT_ROT
25:                         CALL UPG_WAIT_SEC
26:
27:                         MOV  PortAmpel,#ROTGELB
28:                         MOV  WAIT_SEC,#ZEIT_ROTGELB
29:                         CALL UPG_WAIT_SEC
30:
31:                         MOV  PortAmpel,#GRUEN
32:                         MOV  WAIT_SEC,#ZEIT_GRUEN
33:                         CALL UPG_WAIT_SEC
34:
35:                         MOV  PortAmpel ,#GELB
36:                         MOV  WAIT_SEC,#ZEIT_GELB
37:                         CALL UPG_WAIT_SEC
38:
39:                         LJMP Ampel
40:
41:
42:   ; dieses UPG macht einen Delay von genau 1.00 ms
43:   delay_01ms:           PUSH 7         ; legt Register 7 auf den Stack
44:                         MOV  R7,#255   ; 255 x 6 x „Nichtstun“ dauert genau 1 ms
45:   delayloop0:           NOP
46:                         NOP
47:                         NOP
48:                         NOP
49:                         NOP
50:                         NOP
51:                         DJNZ R7,delayloop0
52:                         POP  7         ; holt den ursprünglichen Wert von Register7 zurück
53:                         RET
54:
55:   ; dieses UPG macht einen Delay von genau 10.0 ms
56:   delay_10ms:           PUSH 7         ; legt Register 7 auf den Stack
57:                         MOV  R7,#10    ; ruft 10 mal das UPG für 1 Millisec auf
58:   delayloop1:           CALL delay_01ms
59:                         DJNZ R7,delayloop1
60:                         POP  7         ; holt den ursprünglichen Wert von Register7 zurück
61:                         RET
62:
63:   ; dieses UPG macht einen Delay von genau 1.000 sec
64:   delay_1sec:           PUSH 7         ; legt Register 7 auf den Stack
65:                         MOV  R7,#100   ; ruft 10 mal das UPG für 10 Millisec auf
66:   delayloop10:          CALL delay_10ms
67:                         DJNZ R7,delayloop10
68:                         POP  7         ; holt den ursprünglichen Wert von Register7 zurück
69:                         RET
70:
71:   ; dieses UPG macht einen Delay von genau von 1 ms bis 65535 ms=16Bit
72:   ; der Verzögerungswert steht in R6 HIGH    R5 LOW
73:   ; Bsp:    MOV R6,#60000 SHR 8  ; MOV R5,#60000 MOD 256 ; „wartet“ 1 min
74:   delay_R65ms:          SJMP dmsloop1ms
75:   dmsloopR5ms:          MOV  R5,#255
76:   dmsloop1ms:           CALL delay_01ms
77:                         DJNZ R5,dmsloop1ms
78:                         DJNZ R6,dmsloopR5ms
79:                         RET
80:
81:   ; dieses UPG macht einen Delay von genau  0.01 sec bis 255.99 sec
82:   ; der Verzögerungswert steht in R6 Sek 1..255   R5 zehntelsec 0..99
83:   ; Bsp:    MOV R6,#3  ; MOV R5,#50 ;         „wartet“ 3.5 sec
84:   delay_R65sz:          SJMP dmsloop1sz
85:   dmsloopR5sz:          MOV  R5,#99
86:   dmsloop1sz:           CALL delay_01ms
87:                         DJNZ R5,dmsloop1ms
88:                         DJNZ R6,dmsloopR5sz
89:                         RET
90:
91:   ; dieses UPG macht einen Delay von genau  1 sec bis 255 min (> 4 Std!!)
92:   ; der Verzögerungswert steht in R6=Min=1..255   R5=Sekunde=0..59
93:   ; Bsp:    MOV R6,#45  ; MOV R5,#0 ;         „wartet“ 45 min 0 sec
94:   UPG_WAIT_SEC:         MOV  R6,#1
95:   delay_R6m5s:          SJMP dmsloopsec
96:   dmsloopmin:           MOV  R5,#59
97:   dmsloopsec:           CALL delay_1sec
98:                         DJNZ R5,dmsloopsec
99:                         DJNZ R6,dmsloopmin
100:                        RET
101:
102:                        END
```

Step6: Modulares Assembler-Programm fuer 89C51CC03

Auch bei einem ASM Programm muss nicht der vollständige Quellcode in nur einer einzigen Datei stehen, sondern kann aufteilt werden:

StartUP.A51

```
01: /*
02: ==========================================
03: Projekt:     Ampel_ASM_V02
04: Datei:       StartUP.A51
05: Aufgabe:     Ampel-Ansteuerung
06: Sprache:     Keil C
07: Autor:       R. Hoermann (schule-on-line.de)
08: Version:     2021-03-15
09: */
10:
11: $NOMOD51                        ; vergesse alle Standard 8051 Definitionen
12: $include(at89c51cc03.inc)       ; binde die Definitionen fuer MEINE CPU ein
13:
14:
15:                                 LJMP Start     ; Springe zum Start
16:
17:                                 ORG 0100h      ; Hauptprogramm soll ab 0100h beginnen, kein Konflikt mit Interrupts
18: $include(my_config.inc)         ; Configurations_datei für das Projekt
19: $include(my_upgs.inc)           ; Sammlung von nuetztlichen bzw. haeufig gebrauchten Unterprogrammen
20:
22: Start:
23: Ampel:                          MOV  PortAmpel,#ROT
24:                                 MOV  WAIT_SEC,#ZEIT_ROT
25:                                 CALL UPG_WAIT_SEC
26:
27:                                 MOV  PortAmpel,#ROTGELB
28:                                 MOV  WAIT_SEC,#ZEIT_ROTGELB
29:                                 CALL UPG_WAIT_SEC
30:
31:                                 MOV  PortAmpel,#GRUEN
32:                                 MOV  WAIT_SEC,#ZEIT_GRUEN
33:                                 CALL UPG_WAIT_SEC
34:
35:                                 MOV  PortAmpel ,#GELB
36:                                 MOV  WAIT_SEC,#ZEIT_GELB
37:                                 CALL UPG_WAIT_SEC
38:
39:                                 LJMP Ampel
40:
41:                                 END
```

my_config.inc

```
01: WAIT_SEC             EQU   R5       ; verwende Register5 als Zeitwert
02:
03: #define              ZEIT_ROT     6 ; sec zfuer Ampel-Phasen hier SEC -> "echt" Minuten
04: #define              ZEIT_ROTGELB 1
05: #define              ZEIT_GRUEN   4
06: #define              ZEIT_GELB    1
07:
08: #define              PortAmpel    P1
09:
10:                                    ;NULL-aktiv // EINS aktiv
11: #define              ROT      3     ; 4
12: #define              ROTGELB  1     ; 6
13: #define              GRUEN    6     ; 1
14: #define              GELB     5     ; 2
```

my_upgs.inc (siehe Zeile 42..101)

Step7: Erweiterung um Serielle Ausgabe

Das Programm soll so erweitert werden, dass es zusätzlich die folgenden Ausgaben über seine Serielle Schnittstelle mit **9600 Bit/sec** macht:

Zustand:	Ausgabe via Serial-Interface:	
Ampel Rot	**R**$_{CRLF}$	
Ampel RotGelb	**RY**$_{CRLF}$	**Y="YELLOW"**
Ampel Grün	**G**$_{CRLF}$	
Ampel Gelb	**Y**$_{LCRLF}$	

Ergänzen Sie dazu die Datei **my_upgs.inc** um folgende Zeilen:

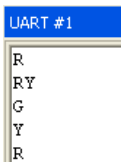
```
..
Serial_Begin_9600:
                    MOV       CKCON,#1
                    MOV       SCON,#50h
                    MOV       RCAP2H,#0FFh
                    MOV       RCAP2L,#0B2h          ;x0FFB2=>9600 x0FFF3=>57600
                    SETB      RCLK
                    SETB      TCLK
                    SETB      TR2                   ; Timer 2 Aktivieren
                    SETB      REN
                    CLR       RI                    ; Recieve Interrupt Flag loeschen
                    SETB      EA
                    CLR       ES                    ; NO Interrupts Enable Polling
                    RET

Serial_WriteAkku:
                    MOV       SBUF,A
                    JNB       TI,$
                    CLR       TI
                    RET

Serial_Writeln:
                    MOV       SBUF,#0x0Dh      ; 0Dh=12d=Wagenruecklauf = \r = CR
                    JNB       TI,$
                    CLR       TI
                    MOV       SBUF,#0x0Ah       ; 0Ah=12d=Wagenruecklauf = \n = LF
                    JNB       TI,$
                    CLR       TI
                    RET
```

und ändern Sie das Hauptprogramm **StartUP.A51** wie folgt ab:

```
Start:              CALL Serial_Begin_9600

Ampel:              MOV  PortAmpel,#ROT
                    MOV  A,#'R'
                    CALL Serial_WriteAkku
                    CALL Serial_Writeln
                    MOV  WAIT_SEC,#ZEIT_ROT
                    CALL UPG_WAIT_SEC

                    MOV  PortAmpel,#ROTGELB
                    MOV  A,#'R'
                    CALL Serial_WriteAkku
                    MOV  A,#'Y'
                    CALL Serial_WriteAkku
                    CALL Serial_Writeln
                    MOV  WAIT_SEC,#ZEIT_ROTGELB
                    CALL UPG_WAIT_SEC

                    MOV  PortAmpel,#GRUEN
                    MOV  A,#'G'
                    CALL Serial_WriteAkku
                    CALL Serial_Writeln
                    MOV  WAIT_SEC,#ZEIT_GRUEN
                    CALL UPG_WAIT_SEC

                    MOV  PortAmpel ,#GELB
                    MOV  A,#'Y'
                    CALL Serial_WriteAkku
                    CALL Serial_Writeln
                    MOV  WAIT_SEC,#ZEIT_GELB
                    CALL UPG_WAIT_SEC

                    LJMP Ampel
```

Wenn Sie nun

- das Programm im Keil-Debugger laufen lassen
- und das Serial-Fenster einblenden
- sehen sie darin folgende Ausgaben:



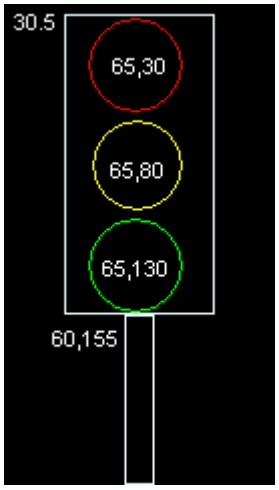
Step8: **Anschluss eines HMI-Farbdisplays**

Als **H**uman-**M**achine-**I**nterface bezeichnet man mehr oder weniger intelligente Displays, die die Kommunikation zwischen Mensch und Maschine erleichtern. Dazu wird der Anlagen-Status auf dem Displays angezeigt und Eingaben an die Maschine weitergeleitet.

Ein einfaches, aber schon eindrucksvolles HMI-Interface
lässt sich mit einem M5-StickCPlus für € 15.00 aufbauen!

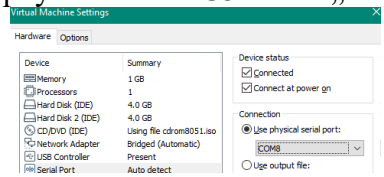
```
#include <M5StickCPlus.h>

#define r 23 // Radius

void do_HMI()
{
          String s=Serial.readStringUntil('\n');
          if(s.indexOf('R')>=0)
                {M5.Lcd.fillCircle(65, 30,r,RED);   }
            else {M5.Lcd.fillCircle(65, 30,r,BLACK);}
          if(s.indexOf('Y')>=0)
                {M5.Lcd.fillCircle(65, 80,r,YELLOW);}
            else {M5.Lcd.fillCircle(65, 80,r,BLACK);}
          if(s.indexOf('G')>=0)
                {M5.Lcd.fillCircle(65,130,r,GREEN); }
            else {M5.Lcd.fillCircle(65,130,r,BLACK);}
}

void setup()
{
          M5.begin();
          M5.Lcd.setRotation(0); M5.Lcd.fillScreen(BLACK);
          M5.Lcd.drawRect(30,  5,75,150,WHITE);
          M5.Lcd.drawRect(60,155,15, 85,WHITE);
          Serial.begin(9600);
}

void loop()
{
          while(Serial.available())
          {
            do_HMI();
          }
}
```

M5.Lcd = 135 → x 240↓



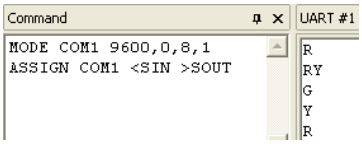
Damit Sie Ausgaben des 89C51CC03 beim Display ankommen müssen Sie

- den M5Stick mit dem obigen Programm bespielen
- die virtuelle Serielle Schnittstelle(**COM1**) der VM mit uVision, mit der physikalischen **COMx** des „echten“ M5Sticks verbinden



- den Debugger in µVision starten
- im Command-Fenster die folgenden beiden Zeilen eingeben:

```
MODE   COM1  9600, 0,8,1
ASSIGN COM1 <SIN >SOUT
```



Challenges:

CH1: Fügen dem Programm einen „Lampentest“ hinzu, der für kurze Zeit alle 3 Lampen der Ampel anmacht.

CH2: Fügen Sie einen Eingangs-Pin (=“Fussgängeranforderung“), der die ROT Zeit der Ampel einmalig verdoppelt, auch wenn der Port nur kurz betätigt wurde.

CH3: Fügen Sie eine zweite Ampel hinzu, um den Verkehr an einer Strassenkreuzung zu steuern!