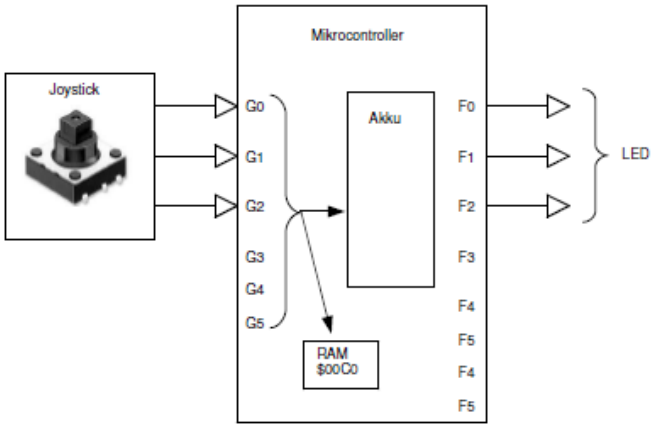
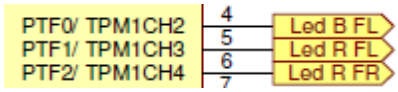
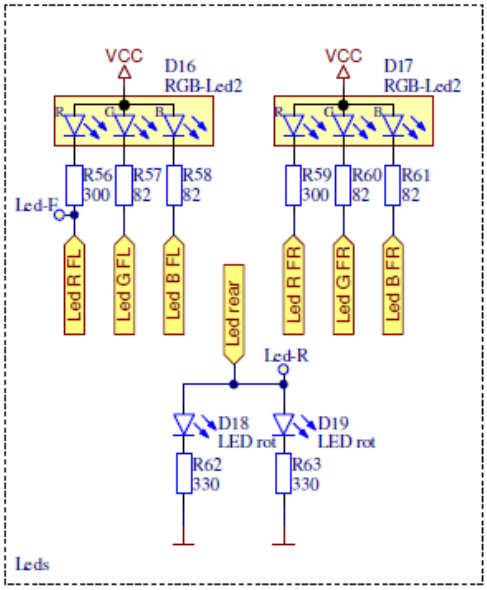
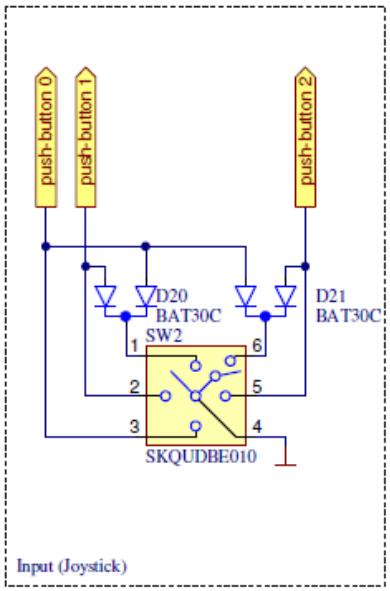
Beschreibung der Grundaufgabe – Übung 2

Am Port G liege durch den Joystick 3-bit Wert an. Dieser Wert ist direkt auf Port G und in das interne RAM auf Adresse $00C0 zu kopieren. Jede Änderung des 3-bit Wertes soll nachgeführt werden. Füllen Sie Anschliessend die Wahrheitstabelle aus.









MC9S08JM60CLH – Port / LED

PTF0 Led B FL

PTF1 Led R FL

PTF2 Led R FR

G2	G1	G0	G2-0 Dez	SW Position	LED
1	1	1	7	Neutral	---
1	1	0	6	Down	Blue FL
1	0	1	5	Left	Red FL
1	0	0	4	Up	Blue FL & Red FL
0	1	1	3	Right	Red FR
0	1	0	2	Press	Red FR & Blue FL
0	0	1	1	---	Red FR & Red FL
0	0	0	0	---	Red FR & Red FL & Blue FL

Assembler-Lösung

1. Ändern Sie ein auf dem Template „Ass_PortIO“ basierendes Assembler-Projekt so ab, dass es die gewünschte Funktion realisiert. Verwenden Sie dabei vorerst die symbolischen Namen der internen Steuerregister. Assemblieren Sie das File und kontrollieren Sie die Funktion auf dem MC CAR.

```
userMain: LDA   #$FF
          STA   PTGPE                   ; Port G Pull-Up Enable
          STA   PTFDD                   ; Port F DataDirection = Output

Loop:     LDA   PTGD                    ; Load Akku <-- Port G Data
          STA   PTFD                    ; Store Akku --> Port F Data
          BRA   Loop                    ; Branch --> Endlosschleife (Sprung zu Loop)
```

2. Ersetzen Sie im Programm jetzt die vordefinierten symbolischen Namen durch absolute Adressen (siehe „MC9S08JM60A.inc“ u. „HSLU_HCS08_Programing_Guide.pdf“) und prüfen Sie erneut die Funktion auf MC CAR.

```
userMain: LDA   #$FF
          STA   $00001858             ; PTGPE=$00001858 | Port G Pull-Up Enable
          STA   $0000000B             ; PTFDD=$0000000B | Port F DataDirection = Output

Loop:     LDA   $0000000C             ; PTGD=$0000000C | Load Akku <-- Port G Data
          STA   $0000000A             ; PTFD=$0000000A | Store Akku --> Port F Data
          BRA   Loop                  ; Branch --> Endlosschleife (Sprung zu Loop)
```

3. Überprüfen Sie jetzt die Funktion ihres Programmes im CW Debugger, in dem sie die Befehle **go**, **stepi**, **stop**, **mem**, **reg**, **bp** in der "Debugger Shell" verwenden.
   In the CodeWarrior Eclipse IDE, the Debugger Shell is available from the C/C++ Perspective by choosing Window > Other > Debug > Debugger Shell, or from Debugger Perspective by choosing Window > Show View > Debugger Shell

4. Im Memory Fenster des Debuggers kontrollieren Sie die transferierten Werte ("Refresh while Running" wählen). Ab welcher Adresse beginnt Ihr Anwenderprogramm?
   0x1969

C-Lösung

1. Führen Sie Aufgabe 1 in der Hochsprache C aus (Template „HCS08_C“). Überprüfen Sie im Debugger Disassembly-Fenster wie die C Anweisungen durch den Compiler übersetzt wurden. Was fällt Ihnen auf?
   … mein Versuch:

```
void main(void)
{
  EnableInterrupts;
  // user code
  PTGPE = 0xff; /* Port G Pull-Up Enable */
  PTFDD = 0xff; /* Port F DataDirection = Output */

  for(;;)
  {
    PTFD = PTGD; /* Port G (Taster) --> Port F (LED) */

    __RESET_WATCHDOG();     /* feeds the dog */
  } /* loop forever */

  /* please make sure that you never leave main */
}
```

?!?!?

Eigene Aufgabe (just-4-fun)

Taster nach vorne: Beide LED vorne leuchten grün
Taster nach hinten: Die LED hinten leuchten (rot)
Taster nach links: Die linke LED leuchtet rot
Taster nach rechts: Die rechte LED leuchtet rot
Taster drücken: Beide LED vorne leuchten blau.

```
/* just-4-fun */
#include "platform.h" /* include peripheral declarations */

#define LED_R_FL  PTFD_PTFD1 /* Red   Front Left */
#define LED_G_FL  PTCD_PTCD4 /* Green Front Left */
#define LED_B_FL  PTFD_PTFD0 /* Blue  Front Left */

#define LED_R_FR  PTFD_PTFD2 /* Red   Front Right */
#define LED_G_FR  PTCD_PTCD6 /* Green Front Right */
#define LED_B_FR  PTED_PTED7 /* Blue  Front Right */

#define LED_Rear  PTDD_PTDD2 /* Led Rear */

// Front-LED Pull-UP !!!
#define ON_Front 0;
#define OFF_Front 1;
// Rear-LED Pull-Down !!!
#define ON_Rear 1;
#define OFF_Rear 0;

void allLEDOff(void)
{
  /* rear */
  LED_Rear = OFF_Rear;
  /* front left */
  LED_R_FL = OFF_Front;
  LED_G_FL = OFF_Front;
  LED_B_FL = OFF_Front;
  /* front right */
  LED_R_FR = OFF_Front;
  LED_G_FR = OFF_Front;
  LED_B_FR = OFF_Front;
}
```

```
/**
 * Application entry point.
 */
void main(void)
{
  byte taster;

  EnableInterrupts;


  // init in & output
  PTGPE = 0x07; /* Port G Pull-Up Enable | Bit 0-3 Pull-Up (Taster) */
  PTCDD = 0x50; /* Port C Data Direction | Set Bit 4,6 as Output */
  PTDDD = 0x04; /* Port D Data Direction | Set Bit 2 as Output */
  PTEDD = 0x80; /* Port E Data Direction | Set Bit 7 as Output */
  PTFDD = 0x07; /* Port F Data Direction | Set Bit 0-3 as Output */

  allLEDOff();

  for(;;)
  {
    taster = PTGD; /* read Taster */

    switch (taster) {
    case 6: /* down */
      LED_Rear = ON_Rear;
      break;
    case 5: /* left */
      LED_R_FL = ON_Front;
      break;
    case 4: /* up */
      LED_G_FL = ON_Front;
      LED_G_FR = ON_Front;
      break;
    case 3: /* right */
      LED_R_FR = ON_Front;
      break;
     case 2: /* press */
       LED_B_FL = ON_Front;
       LED_B_FR = ON_Front;
       break;
     case 7: /* default */
     default:
       allLEDOff(); /* set all LED off */
     }

    __RESET_WATCHDOG();  /* feeds the dog */
  } /* loop forever */

  /* please make sure that you never leave main */
}
```

Assembler Übung aus Unterricht

Location	Obj. Code	Source			Desc.
	---	---	Label10: EQU	$13	Kein Obj. Code (nur Definition)
=0	---	---	Konstanten: SECTION		Definiert die Sektion
+1	0000	01	Label1: DC	$01	Hex 01 (Byte) in Flash speichern
+3	0001	02 03 04	Label2: DC.B	$02, $03, $04	Drei Konstante (Byte) speichern
+2	0004	12 34	Label3: DC.W	$1234	Ein Word (2 Byte) definieren
+4	0006	00 11 22 33	Label4: DC.L	$00112233	Ein Long (4 Byte) definieren
=0		---	Daten: SECTION		Neue Sektion definieren
+3	0000		Label5: DS	3	3 Byte definieren
+4	0003		Label6: DS.B	4	4 Byte definieren