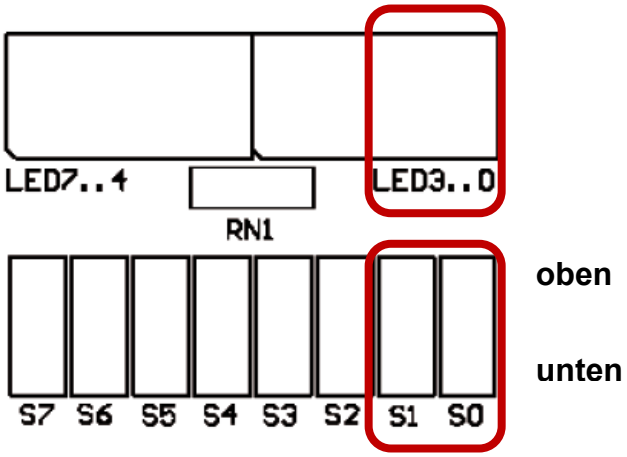
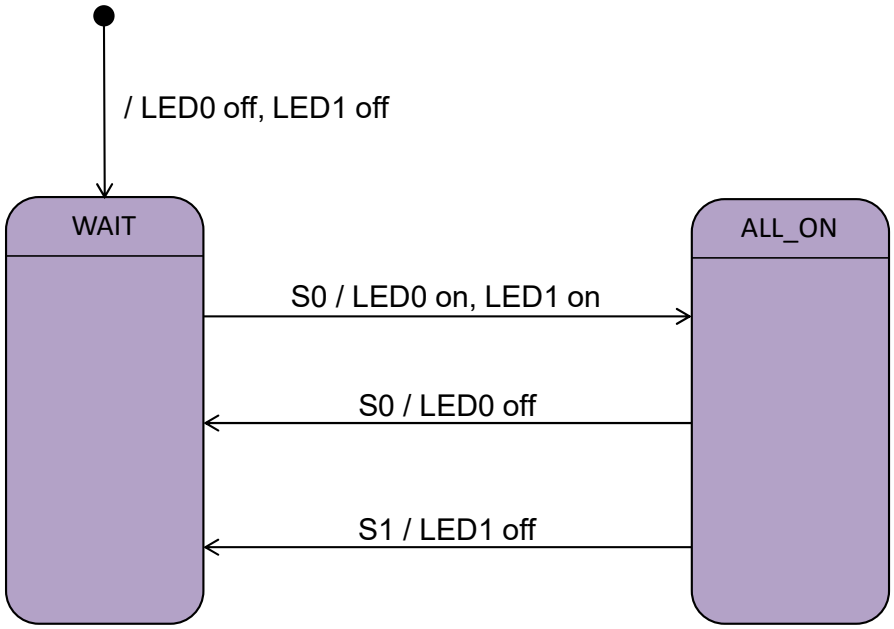
1. Spezifikation LED Steuerung mit CT Board

Beschreibung

Beim Starten des Systems werden beide LED's ausgeschaltet und das System geht in den Wartezustand *WAIT* über. Wird der Schiebeschalter S0 in die Position *oben* geschoben, werden beide LED's (LED0 und LED1) eingeschaltet und das System geht in den Zustand *ALL_ON* über. Wird nun erneut Schalter S0 oder Schalter S1 in die Position *oben* geschoben, wird die entsprechende LED ausgeschaltet und das System geht in den Wartezustand über.



UML-Diagramm



2. Software Struktur

Hauptprogramm

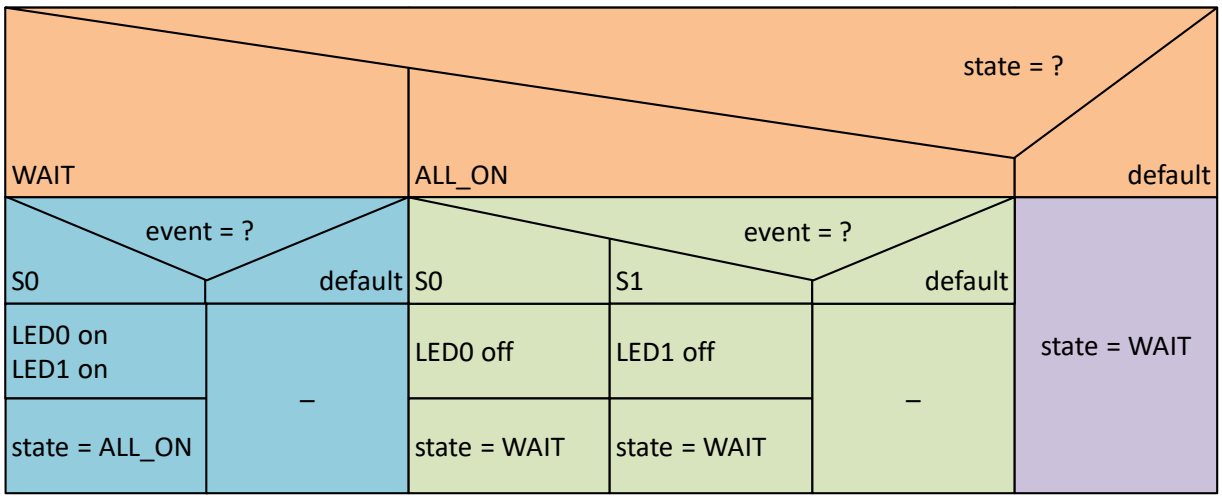
Das Hauptprogramm initialisiert das System und fragt in einer Schleife periodisch ab, ob ein Ereignis an den Schiebeschaltern aufgetreten ist. Falls ja wird das Ereignis zur Verarbeitung an die State Machine weitergegeben.

```
/*
 * Main program
 *
 * Endless loop to detect and process events
 */
int main(void)
{
    event_t event;

    fsm_init();
    timer_init();

    while (1) {
        event = get_event();
        if (event != NO_SWITCH){
            fsm_handle_event(event);
        }
    }
}
```

Zustandsautomat



3. Mögliche Implementierung

Wichtiger Hinweis: Unten stehender Programm-Code ist nur ein Musterbeispiel für eine mögliche Implementierung der einzelnen Funktionen ohne Aufteilung in einzelne Programmmodule, etc. Auch die Handhabung von Ereignissen etc. ist vollständig auf das einfache Beispiel ausgerichtet.

Includes und Macros

```
/* standard includes */
#include <stdint.h>

/* macros for accessing CT Board */
#define LED_07_00   (*((volatile uint8_t *)(0x60000100)))
#define DIPSW_07_00 (*((volatile uint8_t *)(0x60000200)))
```

Definition von Enumerationstypen

```
/* type definitions for states, events and leds */
typedef enum {
    WAIT,
    ALL_ON
} fsm_state_t;

typedef enum {
    NO_SWITCH = 0x00,
    S0        = 0x01,    // bit position of switch S0
    S1        = 0x02     // bit position of switch S1
} event_t;

typedef enum {
    NO_LED    = 0x00,
    LED0      = 0x01,    // bit position of LED0
    LED1      = 0x02,    // bit position of LED1
    LED0_LED1 = 0x03     // bit positions LED0 & LED1
} led_t;
```

Definition einer modullokalen Variable für die Zustände

```
/* use static state variable for module internal usage */
static fsm_state_t state = WAIT;
```

Einlesen eines Ereignisses

```
/*
 * Reads the dip switches and detects events, i.e. changes from
 * 0 to 1 compared to the last reading of the switches. The function
 * includes a priorisation of events for simultaneous events.
 *
 * The variable last_switch_value is defined as static to
 * retain the value between function calls.
 *
 * Returns a single event
 */
static event_t get_event(void)
{
    static uint8_t last_switch_value = 0x3;    // avoids event
                                               // @first call
    event_t retval = NO_SWITCH;
    uint8_t all_events;
    uint8_t current_switch_value;

    current_switch_value = DIPSW_07_00;

    // rising edge detection
    // handle only one event in case of simultaneous events
    all_events = ~last_switch_value & current_switch_value;
    last_switch_value = current_switch_value;
    if (all_events & (uint8_t)S0) {
        retval = S0;
    } else if (all_events & (uint8_t)S1) {
        retval = S1;
    }
    timer_wait_for_tick(); // delay for debouncing
    return retval;
}
```

Ausgeben von Aktionen

```
/*
 * Turns LEDs off according to parameters
 * param[in]: leds  bitmask indicating LEDS to be turned off
 */
static void led_turn_off(led_t leds)
{
    LED_07_00 &= ~leds;
}

/*
 * Turns LEDs on according to parameters
 * param[in]: leds  bitmask indicating LEDS to be turned on
 */
static void led_turn_on(led_t leds)
{
    LED_07_00 |= leds;
}
```

Initialisierung der FSM

```
/*
 * Initializes the FSM and sets the initial state and actions
 */
static void fsm_init(void)
{
    LED_07_00 = 0x0;
    state = WAIT;
}
```

Ereignisse mit Zustandsautomat (FSM) verarbeiten

```
/*
 * Finite State Machine implementation
 *
 * The function processes the given event based on the current
 * state. It sets the new state and triggers the required actions
 *
 * param[in]: event   the event to be processed
 */
static void fsm_handle_event(event_t event)
{
    /* Implementation FSM */
    switch (state) {
        case WAIT:
            switch (event) {
                // an if statement could be used alternatively
                case S0:
                    led_turn_on(LED0_LED1);
                    state = ALL_ON;
                    break;
                default:
                    state = WAIT;
            }
            break;

        case ALL_ON:
            switch (event) {
                case S0:
                    led_turn_off(LED0);
                    state = WAIT;
                    break;

                case S1:
                    led_turn_off(LED1);
                    state = WAIT;
                    break;

                default:
                    state = ALL_ON;
            }
            break;

        default:
            state = WAIT;
    }
}
```