State Machine

Zephyr has a State Machine Framework SMF which implements states as defined in a *state table struct **smf_state***

These states can be populated with functions that are respectively called on entering the state, as the main function of the state and on leaving the state.

A simple representation of how the State Machine in Zephyr can be set up:

```
/* Forward declaration of state table */
static const struct smf_state mvpi_states[];

/* List of MVPI states */
enum mvpi_state { Standby, Operational, Reset_menu, Standby_state, Factory_reset };

/* User defined object */
struct s_object {
        /* This must be first */
        struct smf_ctx ctx;

        /* Other state specific data add here */
} s_obj;

/*
    FUNCTIONS ARE DEFINED HERE
    example: Standby_entry
*/

void Standby_entry(void *o)
{
    LOG_INF("Standby\n");
    state_index = Standby;
    show_state(false);
}

/* Populate state table */
struct smf_state mvpi_states[] = {
    [Standby] = SMF_CREATE_STATE(Standby_entry, Standby_run, Standby_exit, NULL),
    [Operational] = SMF_CREATE_STATE(Operational_entry, Operational_run, Operational_exit, NULL),
    [Configuration] = SMF_CREATE_STATE(Configuration_entry, NULL, Configuration_exit, NULL),
    [Configuration_idle] = SMF_CREATE_STATE(Configuration_idle_entry, Configuration_idle_run, NULL, &mvpi_states
    [Reset_menu] = SMF_CREATE_STATE(Reset_menu_entry, Reset_menu_run, Reset_menu_exit, &mvpi_states[Configuratio
    [Standby_state] = SMF_CREATE_STATE(Standby_state_entry, Standby_state_run, Standby_state_exit, &mvpi_states[
    [Factory_reset] = SMF_CREATE_STATE(Factory_reset_entry, Factory_reset_run, Factory_reset_exit, &mvpi_states[
};


int main(void)
{
    /* Set initial state */
    smf_set_initial(SMF_CTX(&s_obj), &mvpi_states[Standby]);

    /* Run the state machine */
    ret = smf_run_state(SMF_CTX(&s_obj));
}
```

There are two buttons present to change between states as described in the Aloxy Pulse manual.

On pressing either one of the buttons, an interrupt is triggered with a callback to check the input and post an event event corresponding to the input that has triggered.

The different types of input that are checked for, are:

- L_singlepress
- R_singlepress
- R_hold
- R_hold_10s
- Doublepress
- Doublehold

```
#ifndef ZEPHYR_INCLUDE_SM_EVENT_HANDLER_H
#define ZEPHYR_INCLUDE_SM_EVENT_HANDLER_H

#include <zephyr/kernel.h>
#include <zephyr/drivers/gpio.h>
#include <zephyr/smf.h>

#define EVENT_PRESS_LEFT BIT(0)
#define EVENT_PRESS_RIGHT BIT(1)
#define EVENT_PRESS_BOTH BIT(2)
#define EVENT_HOLD_BOTH_3S BIT(3)
#define EVENT_HOLD_RIGHT BIT(4)
#define EVENT_HOLD_RIGHT_10S BIT(5)

/**
 * @def input_event_handler_init()
 * @brief Function to initialise s_obj with events
 */
int sm_event_handler_init(struct k_event *smf_event);

#endif // ZEPHYR_INCLUDE_SM_EVENT_HANDLER_H
```

The full state machine application is split up as follows:

Main

→ init state machine

→ run state machine

State Machine

→ init LED controller

→ define input events

→ define state_object: s_obj

→ define state machine functions

→ set up state machine

→ init input event handler

Event handler

→ init buttons

→ define state machine to handle inputs

→ define intermediate input events:

```c
/* events for internal use */
#define EVENT_BTN_LEFT_DOWN BIT(0)
#define EVENT_BTN_LEFT_UP BIT(1)
#define EVENT_BTN_RIGHT_DOWN BIT(2)
#define EVENT_BTN_RIGHT_UP BIT(3)
#define EVENT_TIMER_3S BIT(4)
#define EVENT_TIMER_10S BIT(5)
```

→ setup interrupts and callback functions on buttons