DESIGN DOCUMENT

SMALL OS DESIGN

(SOS)

Prepared by:

Amr Gamal El-Abd

Table of Contents: -

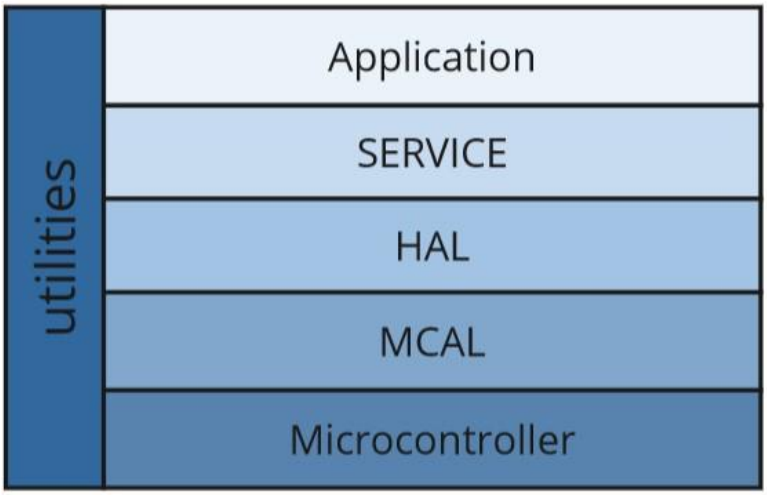
Project introduction:

The SOS (Simple Operating System) plays a vital role within the layered architecture of our embedded system project. Positioned between the hardware-specific layers and the application layer, the SOS abstraction layer provides a unified interface for interacting with the underlying operating system functionalities. It shields the upper layers from the complexities of hardware-dependent operations, enabling developers to write portable and platform-independent code. The SOS abstraction layer promotes modularity, reusability, and maintainability, facilitating the seamless integration of various software components within the system project.

Main Application Flow :

1. Implement an application that calls the SOS module and use 2 tasks
- Task 1: Toggle LED_0 (Every 3 Milli-Seconds)
- Task 2: Toggle LED_1 (Every 5 Milli-Seconds)
2. Make sure that these tasks occur periodically and forever
3. When pressing PBUTTON0, the SOS will stop
4. When Pressing PBUTTON1, the SOS will run.

Layered Architectures:-



Application Layer: This is the topmost layer of the software stack, which contains the actual application logic. It interacts with the lower layers to perform its tasks. It is responsible for implementing the desired functionality of the system.

SERVICE Layer: This layer plays a key role in abstracting and encapsulating the low-level hardware details provided by the MCAL. It provides a set of services, functions, and interfaces that enable the application layer to interact with the underlying hardware

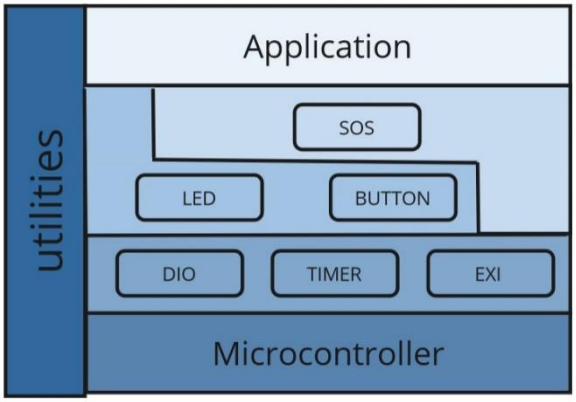
HAL Layer: This layer provides an abstraction for external devices connected to the microcontroller. The HAL layer provides interface to access external devices and hides the implementation details from the application layer.

MCAL Layer (Microcontroller Abstraction Layer): This layer provides an abstraction for the microcontroller hardware. It includes low-level drivers for peripherals. It hides the hardware details and provides a uniform interface to the upper layers.

Utilities Layer: the utilities layer includes memory mapping, standard types, and utils.h. Memory mapping involves defining the memory layout and addresses for different components. Standard types provide a set of predefined data types that ensure consistency and portability across different platforms. The utils.h header file contains utility functions and macros that offer commonly used functionalities, such as bit manipulation.

Microcontroller: This layer represents the physical hardware layer consisting of the microcontroller chip. The microcontroller is responsible for executing the code stored in its memory and controlling the behavior of the system.

Modules description:



SOS Module:

Description: The Simple Operating System (SOS) module is a critical component within the framework of our embedded system project. Designed to provide a streamlined and efficient operating environment, SOS serves as the core software layer responsible for managing system resources, facilitating task scheduling, and enabling seamless communication between various hardware and software components.

LED Module:

Description: The LED driver is responsible for setting up and controlling the LEDs of the microcontroller. It contains different LED states like ON, OFF and toggle.

Button Driver:

Description: The Button driver is responsible for setting up and controlling the buttons of the microcontroller. This driver will be used to detect button presses.

DIO Module:

Description: The DIO (Digital Input Output) driver is responsible for setting up the digital pins of the microcontroller to either input or output mode.

Timer Module:

Description: The Timer module is responsible for setting up and controlling the timers of the microcontroller. This driver will be used to create the timing functionalities required in the project.

Interrupt Module:

Description: The Interrupt module is responsible for setting up and controlling the interrupts of the microcontroller. This driver will be used to set the interrupt functionalities.

Drivers' documentation:-

Service layer:

SOS Module:

Description: The Simple Operating System (SOS) module is a critical component within the framework of our embedded system project. Designed to provide a streamlined and efficient operating environment, SOS serves as the core software layer responsible for managing system resources, facilitating task scheduling, and enabling seamless communication between various hardware and software components.

Functions:

```
enu_sos_Status_t sos_Init() : enu_sos_Status_t;
enu_sos_Status_t sos_CreateTask(u8 u8_Priority, str_sos_configTask_t* str_sos_configTask);
enu_sos_Status_t sos_DeleteTask(u8 u8_Priority) ;
enu_sos_Status_t sos_Run(void) ;
enu_sos_Status_t sos_Deinit(void) ;
enu_sos_Status_t sos_modify_task(u8 u8_Priority, str_sos_configTask_t* str_sos_configTask) ;
enu_sos_Status_t sos_disable(void) ;
```

HAL drivers:

1. LED Driver:

Description: The LED driver is responsible for setting up and controlling the LEDs of the microcontroller. It contains different LED states like ON, OFF and toggle.

Functions:

```
LED_ERROR_TYPE LED_INIT(DIO_PIN_TYPE PIN);
LED_ERROR_TYPE LED_ON(DIO_PIN_TYPE PIN);
LED_ERROR_TYPE LED_OFF(DIO_PIN_TYPE PIN);
```

2. Button Driver:

Description: The Button driver is responsible for setting up and controlling the buttons of the microcontroller. This driver will be used to detect button presses.

Functions:

```
BUTTON_ERROR_TYPE Button_INIT(DIO_PIN_TYPE PIN);
BUTTON_ERROR_TYPE Button_read(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE*VOLT);
```

MCAL drivers:

1. DIO Driver:

Description: The DIO (Digital Input Output) driver is responsible for setting up the digital pins of the microcontroller to either input or output mode. This driver will be used to control the buttons and LEDs.

Functions:

```
DIO_ERROR_TYPE DIO_INITPIN(DIO_PIN_TYPE PIN,DIO_PINSTATUS_TYPE STATUS);
DIO_ERROR_TYPE DIO_WRITEPIN(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE VOLTAGE);
DIO_ERROR_TYPE DIO_READPIN(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE* VOLT);
void DIO_TogglePin(DIO_PIN_TYPE pin);
```

2. Interrupt Driver:

Description: The Interrupt driver is responsible for setting up and controlling the interrupts of the microcontroller. This driver will be used to detect button presses.

Functions:

```
EN_int__error_t EXI_Enable (EN_int_t Interrupt);
EN_int__error_t EXI_Disable (EN_int_t Interrupt);
EN_int__error_t EXI_Trigger(EN_int_t Interrupt,EN_trig trigger);
void EXI_SetCallBack(EN_int_t Interrupt,void(*ptrf)(void));
```

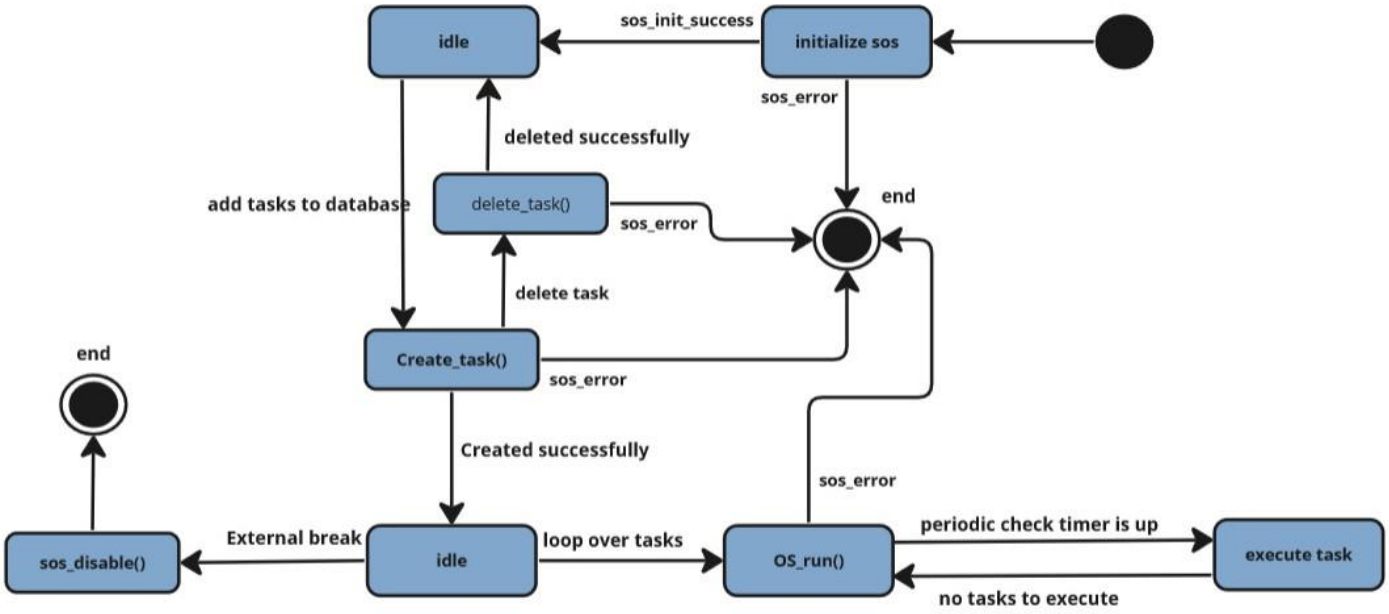
3. Timer Driver:

Description: The Timer driver is responsible for setting up and controlling the timers of the microcontroller. This driver will be used to create the timing delays required in the project.

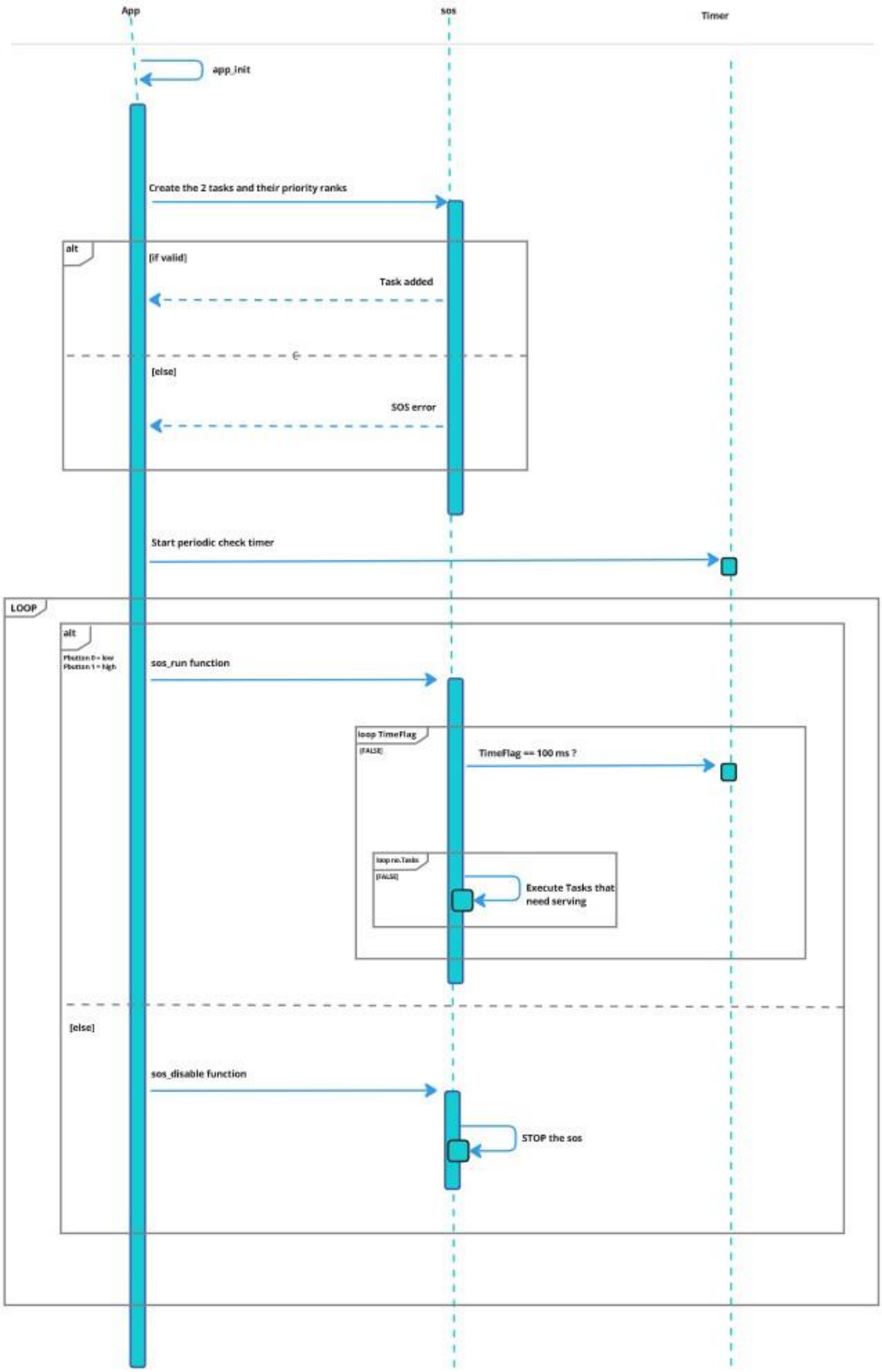
Functions:

```
Timer_ErrorStatus TIMER_2_init(Timer_Mode mode);
Timer_ErrorStatus TIMER_2_start(Timer_Prescaler prescaler);
void TIMER_2_stop(void);
Timer_ErrorStatus TIMER_2_setIntialValue(uint8_t value);
Timer_ErrorStatus TIMER_2_OvfNum(double overflow);
void TIMER_2_DELAY_MS(double _delay);
void TIMER_2_INT();
```

State machine for SOS module:



Sequence Diagram for APP:



Class diagram for SOS module: