# 1. Project Name:

**PO5\_LED STRING ANIMATION**

# 2. Table History:

|  |  |  |  |
| --- | --- | --- | --- |
| **Author** | **Version** | **Date** | **Change Description** |
| Walid Adel | 1.0 | 27/2/2020 | Initial Creation |
| Walid Adel | 1.1 | 28/2/2020 | Added Timer Driver Module API’s in MCAL |
| Walid Adel | 1.2 | 3/1/2020 | * Added 1- Input Output Signals   2-Software Features   * Modified Software Context Diagram * Added Req. ID and Req. Coverage to Each of API table |

# 3. Document Status:

|  |  |  |  |
| --- | --- | --- | --- |
| **Author** | **Version** | **Date** | **Status** |
| Walid Adel | 1.2 | 3/1/2020 | Draft |

# 4. Reference Documents:

|  |  |  |  |
| --- | --- | --- | --- |
| **Reference No.** | **Document Name** | **Version** | **Status** |
| 1 | Req\_ PO5\_LED STRING ANIMATION\_CYRS.doc | 2.2 | Released |
| 2 | Req\_ PO5\_LED STRING ANIMATION\_HSI.doc | 1.5 | Released |

# 4. Table of Content:

Contents

[1. Project Name: 1](#_Toc33985151)

[2. Table History: 1](#_Toc33985152)

[3. Document Status: 1](#_Toc33985153)

[4. Reference Documents: 1](#_Toc33985154)

[4. Table of Content: 2](#_Toc33985155)

[5. Software Context Diagram: 3](#_Toc33985156)

[6. Input Output Signals: 3](#_Toc33985157)

[6.2 Input Signals 3](#_Toc33985158)

[6.2 Output Signals 3](#_Toc33985159)

[7. Software Features: 4](#_Toc33985160)

[8. Static Architecture: 4](#_Toc33985161)

[8.1 Layered Architecture Diagram: 4](#_Toc33985162)

[8.2 Layers Description: 5](#_Toc33985163)

[9. Component API’s: 6](#_Toc33985164)

[9.1 APPLICATION Layer Components: 6](#_Toc33985165)

[9.1.1 MODE MODULE 6](#_Toc33985166)

[9.1.2 TI\_LEFT MODULE 7](#_Toc33985167)

[9.1.3 TI\_RIGHT MODULE 7](#_Toc33985168)

[9.1.4 TAIL MODULE 7](#_Toc33985169)

[9.1.5 BACK LIGHTING CONTROLLER MODULE 8](#_Toc33985170)

[9.2 ECUAL Components: 8](#_Toc33985171)

[9.2.1 LED HANDLER 8](#_Toc33985172)

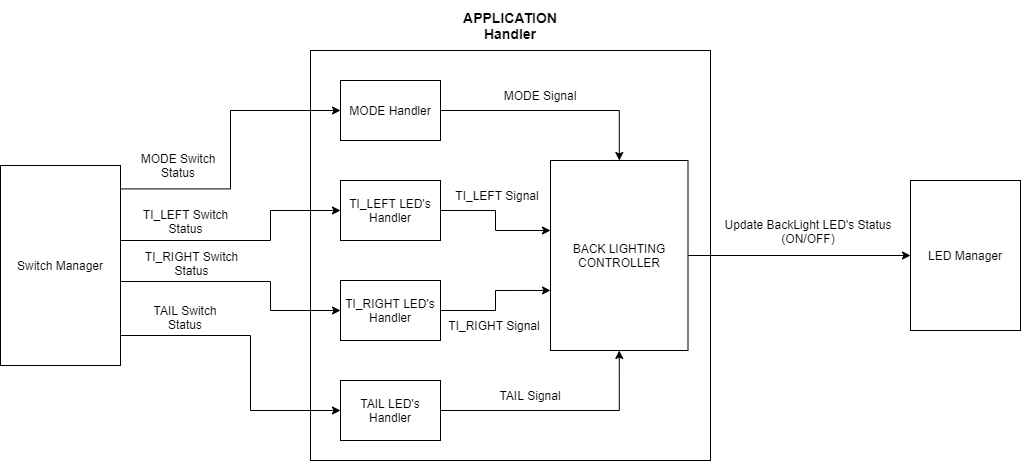
[9.2.2 SWITCH HANDLER 9](#_Toc33985173)

[9.3 MCAL Components: 9](#_Toc33985174)

[9.3.1 GPIO DRIVER 9](#_Toc33985175)

[9.3.2 TIMER DRIVER 10](#_Toc33985176)

# 5. Software Context Diagram:



# 6. Input Output Signals:

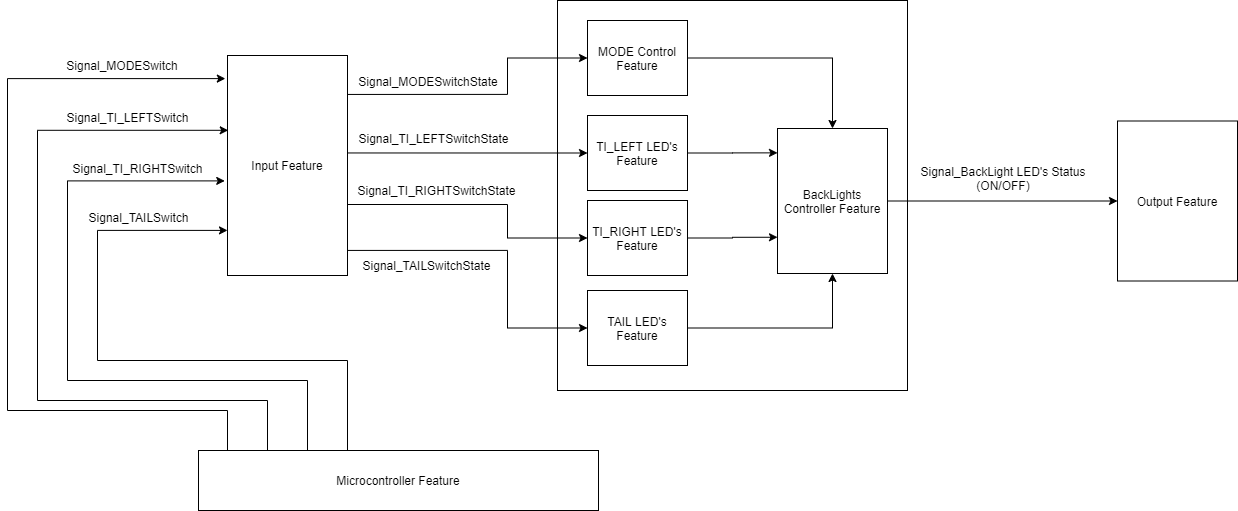
## 6.2 Input Signals

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal Name** | **Signal Type** | **Unit** | **Range** |
| Signal\_MODESwitch | Input | N/A | MODE\_1, MODE\_2 |
| Signal\_TI\_LEFTSwitch | Input | N/A | ON, OFF |
| Signal\_TI\_RIGHTSwitch | Input | N/A | ON, OFF |
| Signal\_TAILSwitch | Input | N/A | ON, OFF |

## 6.2 Output Signals

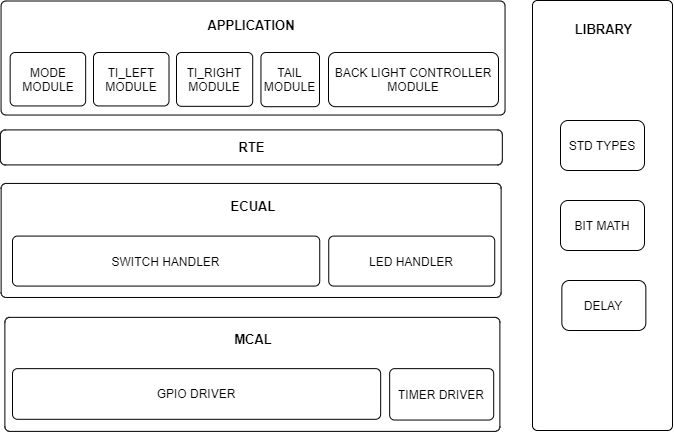
|  |  |  |  |
| --- | --- | --- | --- |
| **Signal Name** | **Signal Type** | **Unit** | **Range** |
| Signal\_UpdateLEDStatus | Output | N/A | ON, OFF |

# 7. Software Features:



# 8. Static Architecture:

## 8.1 Layered Architecture Diagram:



## 8.2 Layers Description:

1-

|  |  |  |
| --- | --- | --- |
| **Layer** | **Module** | **Module Description** |
| APPLICATION | TI\_LEFT MODULE | Responsible for continuously reading the TI LEFT switch status and MODE switch status |
| TI\_RIGHT MODULE | Responsible for continuously reading the TI RIGHT switch status and Mode switch status |
| TAIL MODULE | Responsible for continuously reading the TAIL switch status and MODE switch status |
| BACK LIGHTING MODULE | Responsible for analyzing all switches status and based on those inputs it will activate the TI\_LEDs and TAIL LEDs according to either MODE\_1 or MODE\_2 |

2-

|  |  |  |
| --- | --- | --- |
| **Layer** | **Module** | **Module Description** |
| ECUAL | SWITCH HANDLER | Responsible for initializing all the switches in the system as follows:   * Configuring the switches on specified pins. * Polling on the switch status flags. |
| LED HANDLER | Responsible for initializing all the LEDs in the system as follows:   * Configuring the LEDs on specified pins * Updating the LEDs state |

3-

|  |  |  |
| --- | --- | --- |
| **Layer** | **Module** | **Module Description** |
| LIBRARY | STD\_TYPES | Header file that contains all variable types that will be used in the system |
| BIT\_MATH | Header file that contains all common macros that are needed in the system |
| DELAY MODULE | This Module Contains a Delay Function that is used by the timer driver to execute a delay with a specified value |

4-

|  |  |  |
| --- | --- | --- |
| **Layer** | **Module** | **Module Description** |
| MCAL | GPIO DRIVER | Responsible for:   * Configuring the pins mode. * Writing/Reading from the pins. |
| TIMER DRIVER | Responsible for :   * Configuring the timer’s modes of the system. * Starting the timer’s |

# 9. Component API’s:

## 9.1 APPLICATION Layer Components:

### 9.1.1 MODE MODULE

1-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_001\_V01 |
| **Req. Coverage** | Req\_PO5\_LSAN\_SRS\_022\_V02 |
| **API Prototype** | ERROR\_STATUS Mode\_init(); |
| **Description** | This API is responsible for initializing the MODE switch at which pin |
| **Input Parameters** | Void |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

2-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_002\_V01 |
| **Req. Coverage** | Req\_PO5\_LSAN\_SRS\_002\_V02 , Req\_PO5\_LSAN\_SRS\_023\_V02 |
| **API Prototype** | ERROR\_STATUS Mode\_getStatus(u8\* Switch\_Status ); |
| **Description** | This API is responsible for getting the MODE switch state |
| **Input Parameters** | U8\* Switch\_Status {Pointer to a variable at which the status of the switch will be stored} |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

### 9.1.2 TI\_LEFT MODULE

1-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_003\_V01 |
| **Req. Coverage** | Req\_PO5\_LSAN\_SRS\_022\_V02 |
| **API Prototype** | ERROR\_STATUS TiLeft\_init(); |
| **Description** | This API is responsible for initializing the TI\_LEFT switch at which pin |
| **Input Parameters** | Void |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

2-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_004\_V01 |
| **Req. Coverage** | Req\_PO5\_LSAN\_SRS\_020\_V02 , Req\_PO5\_LSAN\_SRS\_021\_V02 |
| **API Prototype** | ERROR\_STATUS TiLeft\_getStatus(u8\* Switch\_Status ); |
| **Description** | This API is responsible for getting the TI\_LEFT switch state |
| **Input Parameters** | U8\* Switch\_Status {Pointer to a variable at which the status of the switch will be stored} |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

### 9.1.3 TI\_RIGHT MODULE

1-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_005\_V01 |
| **Req. Coverage** | Req\_PO5\_LSAN\_SRS\_022\_V02 |
| **API Prototype** | ERROR\_STATUS TiRight\_init(); |
| **Description** | This API is responsible for initializing the TI\_RIGHT switch at which pin |
| **Input Parameters** | Void |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

2-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_006\_V01 |
| **Req. Coverage** | Req\_PO5\_LSAN\_SRS\_015\_V02 , Req\_PO5\_LSAN\_SRS\_019\_V02 |
| **API Prototype** | ERROR\_STATUS TiRight\_getStatus(u8\* Switch\_Status ); |
| **Description** | This API is responsible for getting the TI\_RIGHT switch state |
| **Input Parameters** | U8\* Switch\_Status {Pointer to a variable at which the status of the switch will be stored} |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

### 9.1.4 TAIL MODULE

1-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_007\_V01 |
| **Req. Coverage** | Req\_PO5\_LSAN\_SRS\_022\_V02 |
| **API Prototype** | ERROR\_STATUS Tail\_init(); |
| **Description** | This API is responsible for initializing the TAIL switch at which pin |
| **Input Parameters** | Void |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

2-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_008\_V01 |
| **Req. Coverage** | Req\_PO5\_LSAN\_SRS\_002\_V02 , Req\_PO5\_LSAN\_SRS\_012\_V02 |
| **API Prototype** | ERROR\_STATUS Tail\_getStatus(u8\* Switch\_Status ); |
| **Description** | This API is responsible for getting the TAIL switch state |
| **Input Parameters** | U8\* Switch\_Status {Pointer to a variable at which the status of the switch will be stored} |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

### 9.1.5 BACK LIGHTING CONTROLLER MODULE

1-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_009\_V01 |
| **Req. Coverage** | Req\_PO5\_LSAN\_SRS\_003\_V02 , Req\_PO5\_LSAN\_SRS\_011\_V02 , Req\_PO5\_LSAN\_SRS\_013\_V02 , Req\_PO5\_LSAN\_SRS\_014\_V02 , Req\_PO5\_LSAN\_SRS\_016\_V02 , Req\_PO5\_LSAN\_SRS\_017\_V02 |
| **API Prototype** | void BackLightingController(); |
| **Description** | This API is responsible for Starting a runnable that integrate the whole Back Lighting System by:   * Analyzing the input signals from other modules to choose which mode (MODE\_1, MODE\_2) at start-up will execute * Updating the Back Lights LED’s status |
| **Input Parameters** | Void |
| **Output Parameters** | Void |

## 9.2 ECUAL Components:

### 9.2.1 LED HANDLER

1-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_010\_V01 |
| **Req. Coverage** | Req\_PO5\_LSAN\_SRS\_022\_V02 |
| **API Prototype** | ERROR\_STATUS LED\_Init(u8 LED\_Num); |
| **Description** | This API is responsible for Initializing the pins direction at which the LED is connected at |
| **Input Parameters** | U8 LED\_Num { LED\_1, LED\_2, etc.} |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

2-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_011\_V01 |
| **Req. Coverage** | Req\_PO5\_LSAN\_SRS\_003\_V02 , Req\_PO5\_LSAN\_SRS\_011\_V02 , Req\_PO5\_LSAN\_SRS\_013\_V02 , Req\_PO5\_LSAN\_SRS\_014\_V02 , Req\_PO5\_LSAN\_SRS\_016\_V02 , Req\_PO5\_LSAN\_SRS\_017\_V02 |
| **API Prototype** | ERROR\_STATUS LED\_updateStatus (u8 LED\_Num , u8 status); |
| **Description** | This API is responsible for Setting the LED status to be ON/OFF |
| **Input Parameters** | 1. U8 LED\_Num {LED\_1, LED\_2, etc.} 2. U8 status {ON = 1, OFF = 0} |
| **Output Parameters** | 1. ERROR\_STATUS {OK = 1, NOK = 0} |

### 9.2.2 SWITCH HANDLER

1-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_012\_V01 |
| **Req. Coverage** | Req\_PO5\_LSAN\_SRS\_022\_V02 |
| **API Prototype** | ERROR\_STATUS SWITCH\_Init(u8 Switch\_Num); |
| **Description** | This API is responsible Initializing the pins direction at which each of the switch will be connected at |
| **Input Parameters** | U8 Switch\_Num |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

2-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_013\_V01 |
| **Req. Coverage** | Req\_PO5\_LSAN\_SRS\_002\_V02 , Req\_PO5\_LSAN\_SRS\_012\_V02 , Req\_PO5\_LSAN\_SRS\_015\_V02 , Req\_PO5\_LSAN\_SRS\_019\_V02 , Req\_PO5\_LSAN\_SRS\_020\_V02 , Req\_PO5\_LSAN\_SRS\_021\_V02 , Req\_PO5\_LSAN\_SRS\_023\_V02 |
| **API Prototype** | ERROR\_STATUS SWITCH\_getStatus (u8 Switch\_Num ,u8\* Status); |
| **Description** | This API is responsible Reading the status of the switch |
| **Input Parameters** | 1. U8 Switch\_Num 2. U8 \*Status {Pointer to a variable at which the status of the switch will be stored} |
| **Output Parameters** | 1. ERROR\_STATUS {OK = 1, NOK = 0} |

## 9.3 MCAL Components:

### 9.3.1 GPIO DRIVER

1-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_014\_V01 |
| **Req. Coverage** | Req\_PO5\_LSAN\_SRS\_022\_V02 |
| **API Prototype** | ERROR\_STATUS GPIO\_Init(u8 Port, u8 Pin, u8 Direction); |
| **Description** | This API is responsible for initializing the GPIO.   1. Initializing The Direction of The Specified Pin by Writing on the DDRx |
| **Input Parameters** | 1. U8 Port {“A”, “B”, “C”, “D”, “E”, “F”} 2. U8 Pin {PIN\_0 = 0, PIN\_1 = 1, PIN\_2 = 2, etc.} 3. U8 Direction {1 = Output ,0 = Input} |
| **Output Parameters** | 1. ERROR\_STATUS {OK = 1, NOK = 0} |

2-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_015\_V01 |
| **Req. Coverage** | N/A |
| **API Prototype** | ERROR\_STATUS GPIO\_writePin(u8 Port, u8 Pin, u8 Value); |
| **Description** | This API is responsible for writing on A specified GPIO pin.   1. Writing on the PORTx register Using the Value Entered |
| **Input Parameters** | 1. U8 Port {“A”, “B”, “C”, “D”} 2. U8 Pin {PIN\_0 = 0, PIN\_1 = 1, PIN\_2 = 2, etc.} 3. U8 Value {1 = HIgh , 0 = Input} |
| **Output Parameters** | 1. ERROR\_STATUS {OK = 1, NOK = 0} |

3-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_016\_V01 |
| **Req. Coverage** | N/A |
| **API Prototype** | ERROR\_STATUS GPIO\_readPin(u8 Port, u8 Pin, u8 \*Value); |
| **Description** | This API is responsible for writing on a specified GPIO pin.   1. Reading From the PORTx register |
| **Input Parameters** | 1. U8 Port {“A”, “B”, “C”, “D”, “E”, “F”} 2. U8 Pin {PIN\_0 = 0, PIN\_1 = 1, PIN\_2 = 2, etc.} 3. U8\* Value {Ptr to hold the Value Read From The Pin} |
| **Output Parameters** | 1. ERROR\_STATUS {OK = 1, NOK = 0} |

### 9.3.2 TIMER DRIVER

1-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_017\_V01 |
| **Req. Coverage** | N/A |
| **API Prototype** | ERROR\_STATUS TIMER\_Init(u8 Timer, u8 Mode, u8 Pre scalar, u8 Preload\_Value); |
| **Description** | This Api is responsible for initializing a specified timer.   1. Configuring the timer mode (PWM, Overflow, CTC) 2. Setting the pre scalar 3. Setting the preload value |
| **Input Parameters** | 1. u8 Timer {TIMER\_0, TIMER\_1, TIMER\_2} 2. u8 Mode {PWM, OVERFLOW, CTC} 3. u8 Pre scalar {NO\_PRESCALAR, 2, 4, 64,etc.} 4. u8 Preload\_Value {0,1, etc.} |
| **Output Parameters** | 1. ERROR\_STATUS {OK = 1, NOK = 0} |

2-

|  |  |
| --- | --- |
| **Req. ID** | Req\_PO5\_LSAN\_GDD\_018\_V01 |
| **Req. Coverage** | N/A |
| **API Prototype** | ERROR\_STATUS TIMER\_start(); |
| **Description** | This Api is responsible for starting the timer |
| **Input Parameters** | Void |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |