# 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 |

# 3. Document Status:

|  |  |  |  |
| --- | --- | --- | --- |
| **Author** | **Version** | **Date** | **Status** |
| Walid Adel | 1.0 | 27/2/2020 | Draft |
| Walid Adel | 1.1 | 28/2/2020 | 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](#_Toc33789376)

[2. Table History: 1](#_Toc33789377)

[3. Document Status: 1](#_Toc33789378)

[4. Reference Documents: 1](#_Toc33789379)

[4. Table of Content: 2](#_Toc33789380)

[5. Static Architecture: 3](#_Toc33789381)

[5.1 Layered Architecture Diagram: 3](#_Toc33789382)

[5.2 Layers Description: 4](#_Toc33789383)

[6. Software Context Diagram: 5](#_Toc33789384)

[7. Component API’s: 5](#_Toc33789385)

[7.1 APPLICATION Layer Components: 5](#_Toc33789386)

[7.1.1 TI\_LEFT MODULE 6](#_Toc33789387)

[7.1.2 TI\_RIGHT MODULE 6](#_Toc33789388)

[7.1.3 TAIL MODULE 6](#_Toc33789389)

[7.1.4 BACK LIGHTING MODULE 7](#_Toc33789390)

[7.2 ECUAL: 7](#_Toc33789391)

[7.2.1 LED HANDLER 7](#_Toc33789392)

[7.2.2 SWITCH HANDLER 8](#_Toc33789393)

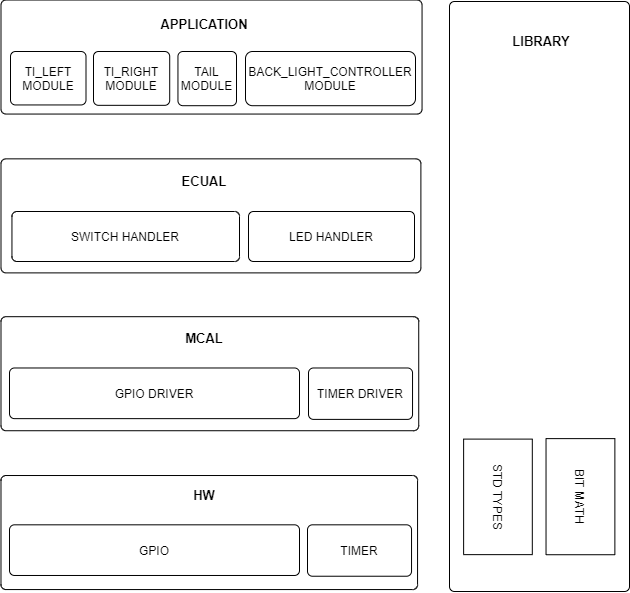
[7.3 MCAL: 8](#_Toc33789394)

[7.3.1 GPIO DRIVER 8](#_Toc33789395)

[7.3.2 TIMER DRIVER 9](#_Toc33789396)

# 5. Static Architecture:

## 5.1 Layered Architecture Diagram:



## 5.2 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 |

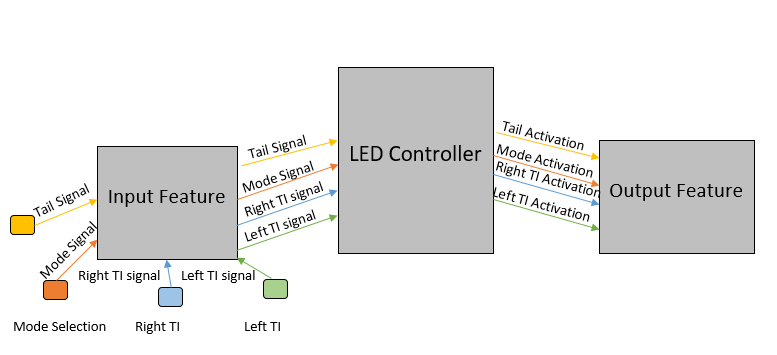
3-

|  |  |  |
| --- | --- | --- |
| **Layer** | **Module** | **Module Description** |
| LIBRARY | STD\_TYPES.h | Header file that contains all variable types that will be used in the system |
| BIT\_MATH.h | Header file that contains all common macros that are needed in the system |

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 |

# 6. Software Context Diagram:



# 7. Component API’s:

## 7.1 APPLICATION Layer Components:

### 7.1.1 TI\_LEFT MODULE

1-

|  |  |
| --- | --- |
| **API Prototype** | ERROR\_STATUS TiLeft\_init(); |
| **Description** | This API is responsible for initializing the TI\_LEFT switch at which pin by calling an API from Switch Handler called: ERROR\_STATUS SWITCH\_init(u8 Switch\_Num); |
| **Input Parameters** | Void |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

2-

|  |  |
| --- | --- |
| **API Prototype** | ERROR\_STATUS TiLeft\_getStatus(u8\* Switch\_Status ); |
| **Description** | This API is responsible for getting the TI\_LEFT switch state by calling an API from Switch Handler called:  ERROR\_STATUS SWITCH\_getSwitchState(u8\* Switch\_Num); |
| **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} |

### 7.1.2 TI\_RIGHT MODULE

1-

|  |  |
| --- | --- |
| **API Prototype** | ERROR\_STATUS TiRight\_init(); |
| **Description** | This API is responsible for initializing the TI\_RIGHT switch at which pin by calling an API from Switch Handler called:  ERROR\_STATUS SWITCH\_init(u8 Switch\_Num); |
| **Input Parameters** | Void |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

2-

|  |  |
| --- | --- |
| **API Prototype** | ERROR\_STATUS TiRight\_getStatus(u8\* Switch\_Status ); |
| **Description** | This API is responsible for getting the TI\_RIGHT switch state by calling an API from Switch Handler called:  ERROR\_STATUS SWITCH\_getSwitchState(u8\* Switch\_Num); |
| **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} |

### 7.1.3 TAIL MODULE

1-

|  |  |
| --- | --- |
| **API Prototype** | ERROR\_STATUS Tail\_init(); |
| **Description** | This API is responsible for initializing the TAIL switch at which pin by calling an API from Switch Handler called: ERROR\_STATUS SWITCH\_init(u8 Switch\_Num); |
| **Input Parameters** | Void |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

2-

|  |  |
| --- | --- |
| **API Prototype** | ERROR\_STATUS Tail\_getStatus(u8\* Switch\_Status ); |
| **Description** | This API is responsible for getting the TAIL switch state by calling an API from Switch Handler called:  ERROR\_STATUS SWITCH\_getSwitchState(u8\* Switch\_Num); |
| **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} |

### 7.1.4 BACK LIGHTING MODULE

1-

|  |  |
| --- | --- |
| **API Prototype** | void BackLightingRunnable(); |
| **Description** | This API is responsible for Starting a runnable that integrate the whole Back Lighting System by:   * Calling API’s From TI\_LEFT, TI\_RIGHT and TAIL Modules called:  1. ERROR\_STATUS TiLeft\_getStatus(u8\* Switch\_Status ); 2. ERROR\_STATUS TiRight\_getStatus(u8\* Switch\_Status ); 3. ERROR\_STATUS Tail\_getStatus(u8\* Switch\_Status );  * Analyzing the input signals from the above modules to choose which mode (MODE\_1, MODE\_2) at start-up will execute * Updating the Back Lights LED’s status by calling API’s from LED HANDLER Module called: ERROR\_ STATUS LED\_updateStatus(u8 Status); |
| **Input Parameters** | Void |
| **Output Parameters** | Void |

## 7.2 ECUAL:

### 7.2.1 LED HANDLER

1-

|  |  |
| --- | --- |
| **API Prototype** | ERROR\_STATUS LED\_Init(void); |
| **Description** | This API is responsible for Initializing the pins direction at which the LED is connected at using an API from the GPIO Module called :  ERROR\_STATUS GPIO\_Init(LED\_PORT, LED\_PIN, LED\_MODE); |
| **Input Parameters** | Void |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

2-

|  |  |
| --- | --- |
| **API Prototype** | ERROR\_STATUS LED\_updateStatus (u8 status); |
| **Description** | This API is responsible for Setting the LED status to be ON/OFF using an API from the GPIO Module called :  ERROR\_STATUS GPIO\_writePin(LED\_PORT,LED\_PIN,LED\_ON); |
| **Input Parameters** | Void |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

### 7.2.2 SWITCH HANDLER

1-

|  |  |
| --- | --- |
| **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 using an API from the GPIO Module called:  ERROR\_STATUS GPIO\_Init(SWITCH\_PORT, SWITCH\_PIN, SWITCH\_MODE); |
| **Input Parameters** | U8 Switch\_Num |
| **Output Parameters** | ERROR\_STATUS {OK = 1, NOK = 0} |

2-

|  |  |
| --- | --- |
| **API Prototype** | ERROR\_STATUS SWITCH\_getStatus (u8 Switch\_Num ,u8\* Status); |
| **Description** | This API is responsible Reading the status of the switch using an API from the GPIO module called:  ERROR\_STATUS GPIO\_readPin(SWITCH\_PORT, SWITCH\_PIN, &Status); |
| **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} |

## 7.3 MCAL:

### 7.3.1 GPIO DRIVER

1-

|  |  |
| --- | --- |
| **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-

|  |  |
| --- | --- |
| **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-

|  |  |
| --- | --- |
| **API Prototype** | Error\_S 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} |

### 7.3.2 TIMER DRIVER

1-

|  |  |
| --- | --- |
| **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-

|  |  |
| --- | --- |
| **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} |