Name: Abdelrhman Walaa Hussein

LED Sequence V2.0 Design

Outline

This document shows a design of an application that uses ATmega32 to control some LEDs lighting sequence according to button pressing.

Hardware Requirements:

- Four LEDs (LED0, LED1, LED2, LED3)
- One button (BUTTON0)

Software Requirements:

1. Initially, all LEDs are OFF
2. Once BUTTON0 is pressed, LED0 will be ON
3. Each press further will make another LED is ON
4. At the fifth press, LED0 will changed to be OFF
5. Each press further will make only one LED is OFF
6. This will be repeated forever
7. The sequence is described below
    a. Initially (OFF, OFF, OFF, OFF)
    b. Press 1 (ON, OFF, OFF, OFF)
    c. Press 2 (ON, ON, OFF, OFF)
    d. Press 3 (ON, ON, ON, OFF)
    e. Press 4 (ON, ON, ON, ON)
    f. Press 5 (OFF, ON, ON, ON)
    g. Press 6 (OFF, OFF, ON, ON)
    h. Press 7 (OFF, OFF, OFF, ON)
    i. Press 8 (OFF, OFF, OFF, OFF)
    j. Press 9 (ON, OFF, OFF, OFF)
8. Use *External Interrupts*

State Machine Diagram

Definition

A *state machine diagram* (*Figure 1*) models the behavior of a single object, specifying the sequence of events that an object goes through during its lifetime in response to events.

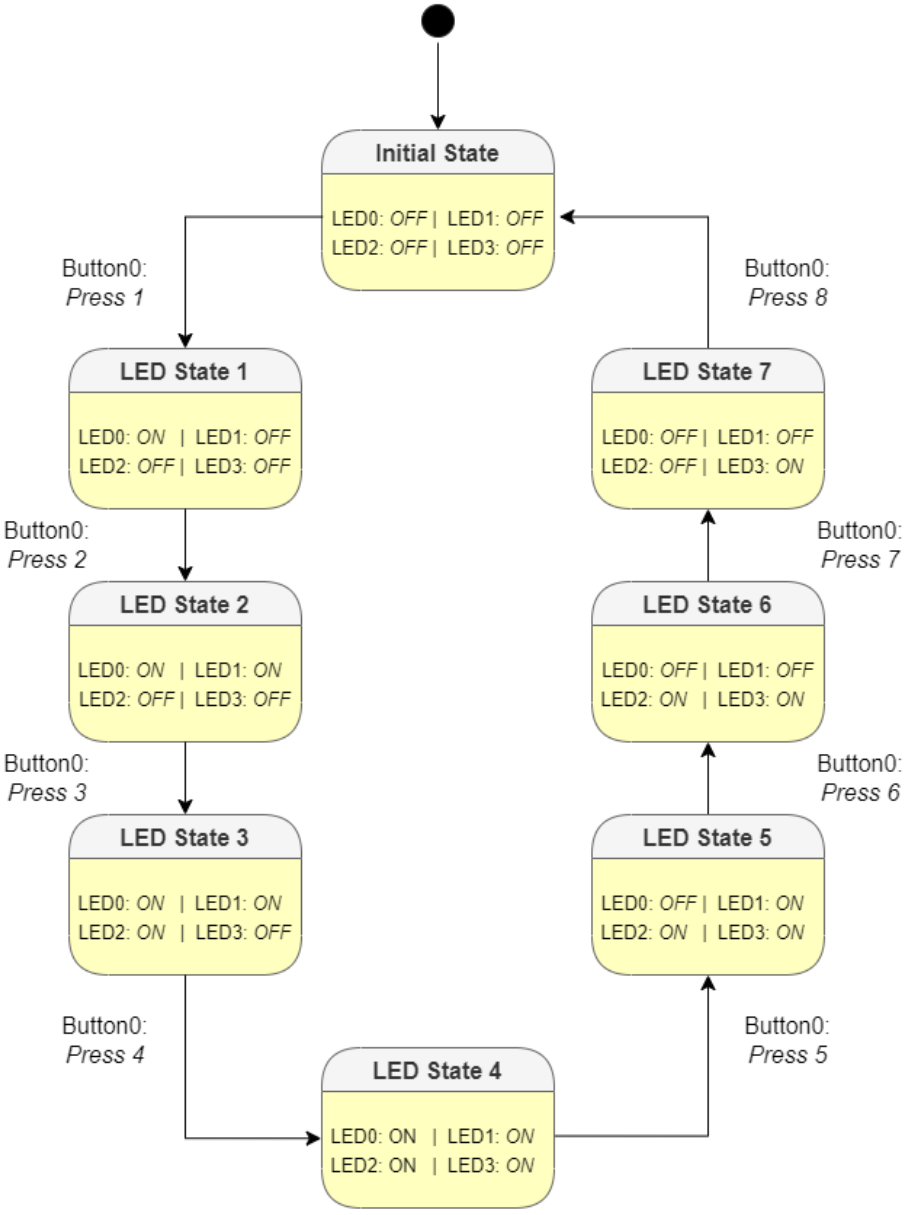


Figure 1. State Machine Diagram

Layered Architecture

Definition

Layered Architecture (*Figure 2*) describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software.

Microcontroller Abstraction Layer (*MCAL*) is a software module that directly accesses on-chip MCU peripheral modules and external devices that are mapped to memory, and makes the upper software layer independent of the MCU.

Hardware Abstraction Layer (*HAL*) is a layer of programming that allows a computer OS to interact with a hardware device at a general or abstract level rather than at a detailed hardware level.

Design

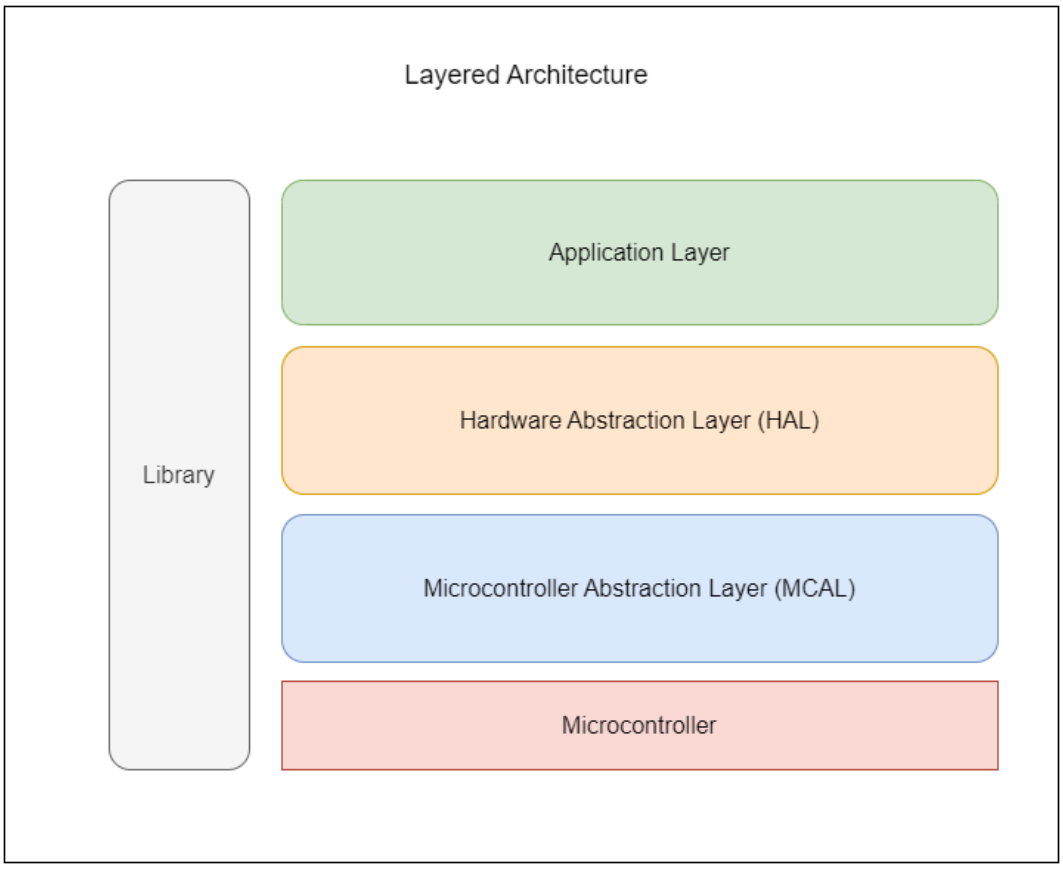


Figure 2. Layered Architecture Design

System Modules

Definition

A *Module* is a distinct assembly of components that can be easily added, removed or replaced in a larger system. Generally, a *Module* is not functional on its own.

In computer hardware, a *Module* is a component that is designed for easy replacement.

Design

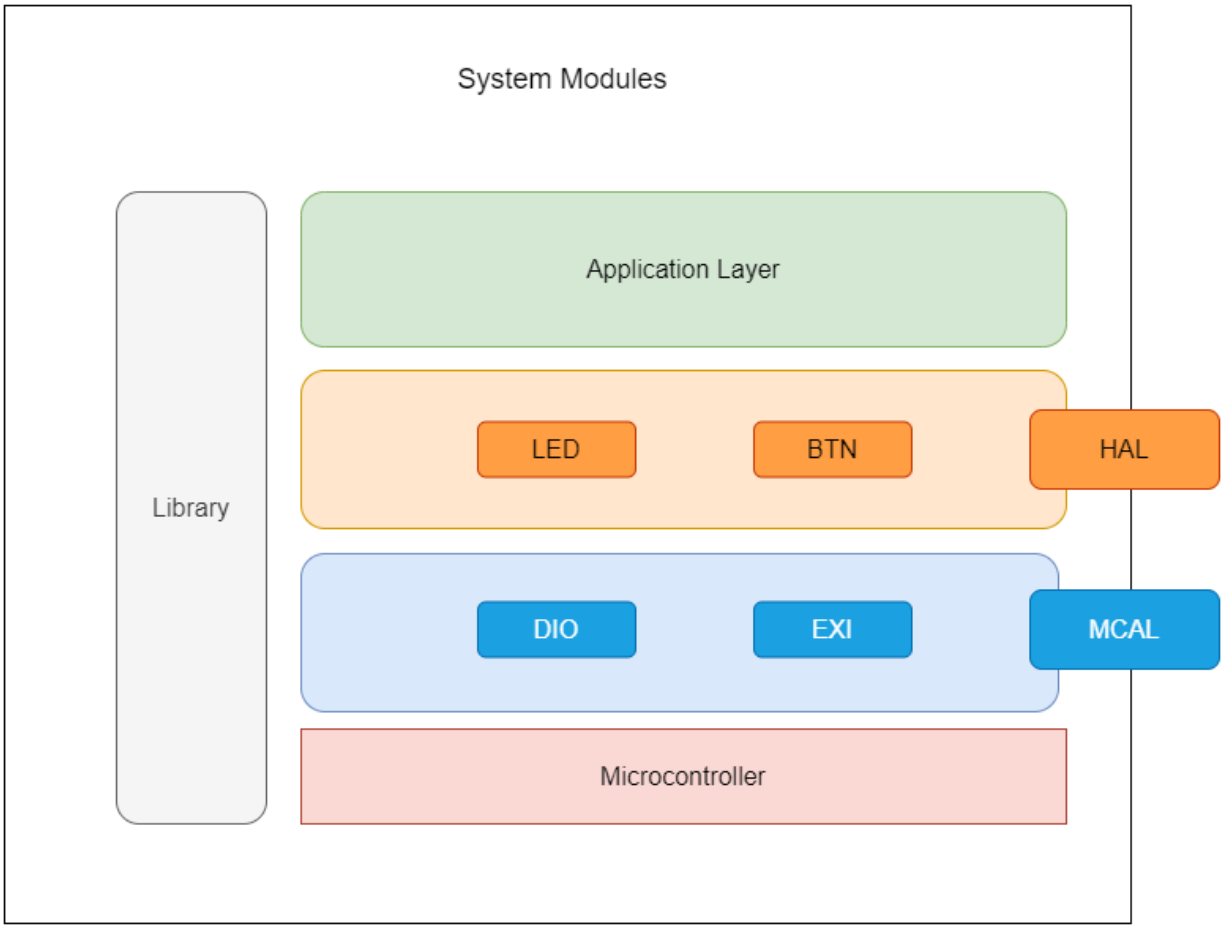


Figure 3. System Modules Design

Application Programming Interfaces (APIs)

Definition

An *API* is an *Application Programming Interface* that defines a set of *routines*, *protocols* and *tools* for creating an application. An *API* defines the high level interface of the behavior and capabilities of the component and its inputs and outputs.

An *API* should be created so that it is generic and implementation independent. This allows for the API to be used in multiple applications with changes only to the implementation of the API and not the general interface or behavior.

MCAL APIs

A. DIO APIs

vd DIO_vdInitialization (void)

Name: DIO_vdInitialization
Input: void
Output: void
Description: Function to initialize DIO peripheral.

u8 DIO_u8SetPinDirection (u8 Cpy_u8PortId, u8 Cpy_u8PinId, u8 Cpy_u8PinDirection)

Name: DIO_u8SetPinDirection
Input: u8 PortId, u8 PinId, and u8 PinDirection
Output: u8 Error or No Error
Description: Function to set Pin direction.

u8 DIO_u8SetPinValue (u8 Cpy_u8PortId, u8 Cpy_u8PinId, u8 Cpy_u8PinValue)

Name: DIO_u8SetPinValue
Input: u8 PortId, u8 PinId, and u8 PinValue
Output: u8 Error or No Error
Description: Function to set Pin value.

*u8 DIO_u8GetPinValue (u8 Cpy_u8PortId, u8 Cpy_u8PinId, u8 *Cpy_pu8ReturnedPinValue)*

Name: DIO_u8GetPinValue
Input: u8 PortId, u8 PinId, and Pointer to u8 ReturnedPinValue
Output: u8 Error or No Error
Description: Function to get Pin value.

u8 DIO_u8TogglePinValue (u8 Cpy_u8PortId, u8 Cpy_u8PinId)

Name: DIO_u8TogglePinValue
Input: u8 PortId and u8 PinId
Output: u8 Error or No Error
Description: Function to toggle Pin value.

B. *EXI APIs*

u8 EXI_u8EnablePIE (u8 Cpy_u8InterruptId, u8 Cpy_u8SenseControl)

Name: EXI_u8EnablePIE
Input: u8 InterruptId and u8 SenseControl
Output: u8 Error or No Error
Description: Function to enable and configure Peripheral Interrupt Enable (PIE), by setting relevant bit for each interrupt in GICR register, then configuring Sense Control in MCUCR (case interrupt 0 or 1) or MCUCSR (case interrupt 2) registers.

u8 EXI_u8DisablePIE (u8 Cpy_u8InterruptId)

Name: EXI_u8DisablePIE
Input: u8 InterruptId
Output: u8 Error or No Error
Description: Function to disable Peripheral Interrupt Enable (PIE), by clearing relevant bits for each interrupt in GICR register.

*u8 EXI_u8INTSetCallBack (u8 Cpy_u8InterruptId, void (*Cpy_pfINTInterruptAction) (void))*

Name: EXI_u8SetCallBack
Input: u8 InterruptId and Pointer to Function that takes void and returns void
Output: u8 Error or No Error
Description: Function to receive an address of a function (in APP Layer) to be called back in ISR function of the passed Interrupt (InterruptId), the address is passed through a pointer to function (INTInterruptAction), and then pass this address to ISR function.

HAL APIs

A. *LED APIs*

vd LED_vdInitialization(void)

Name: LED_vdInitialization
Input: void
Output: void
Description: Function to initialize LED peripheral, by initializing DIO peripheral.

u8 LED_u8SetLEDPin (u8 Cpy_u8LEDId, u8 Cpy_u8Operation)

Name: LED_u8SetLEDPin
Input: u8 LedId and u8 Operation
Output: u8 Error or No Error
Description: Function to switch LED on, off, or toggle.

B. *BTN APIs*

*u8 BTN_u8GetBTNState (u8 Cpy_u8BTNId, u8 *Cpy_pu8ReturnedBTNState)*

Name: BTN_u8GetBTNState
Input: u8 BTNId and Pointer to u8 ReturnedBTNState
Output: u8 Error or No Error
Description: Function to get BTN state.

Flowchart Diagram

Definition

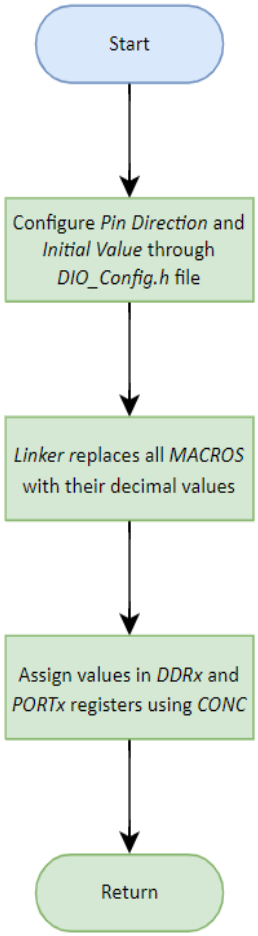
A *Flowchart* (or Flow Chart) is a diagram that shows the steps in a process.

Flowcharts are often used for visualizing the sequence of actions or information needed for training, documenting, planning, and decision-making. They often use symbols, shapes, and arrows to illustrate how one step leads to another.

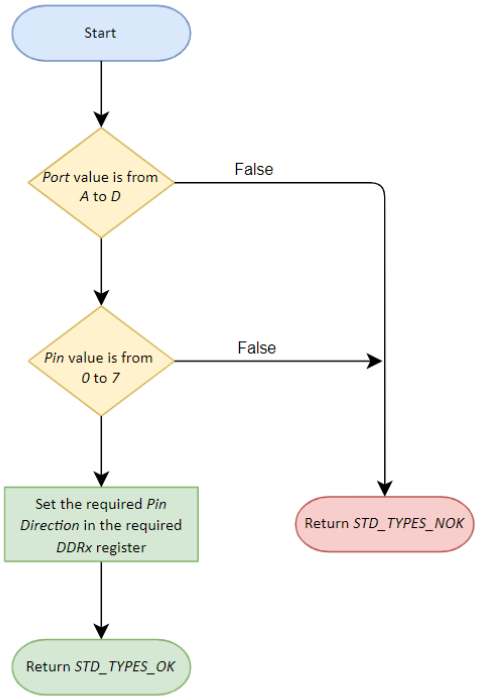
MCAL Flowcharts

A. DIO Flowcharts

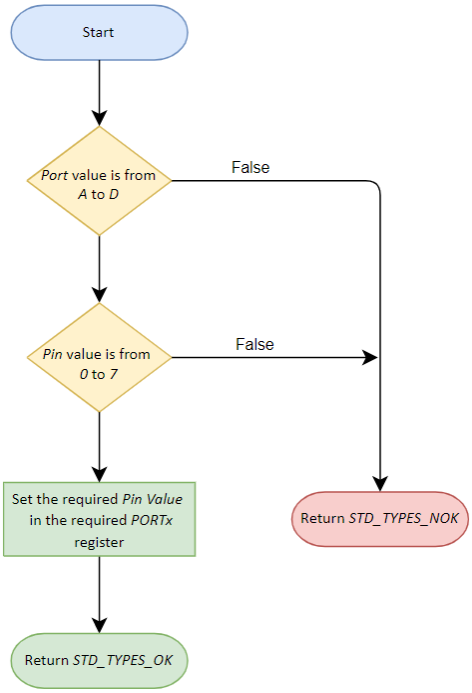
a. DIO_vdInitialization



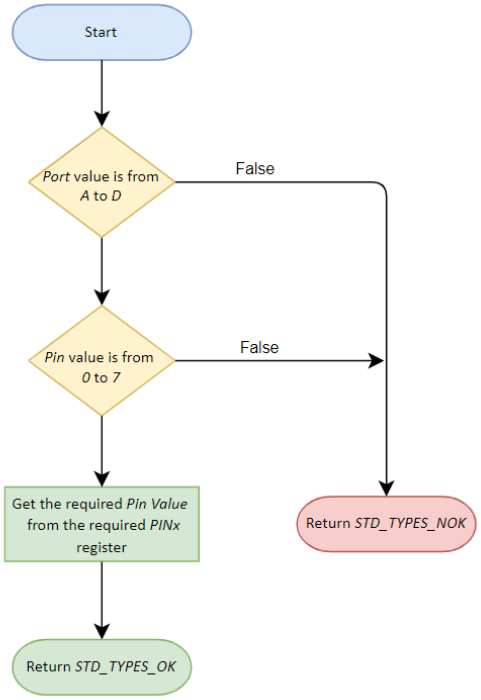
b. *DIO_u8SetPinDirection*



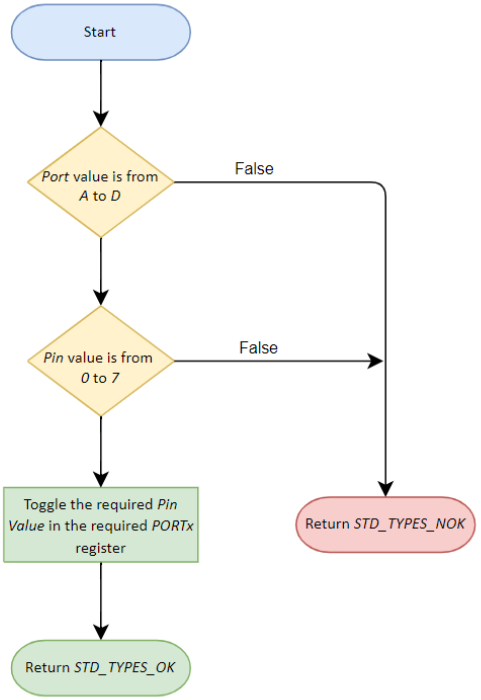
c. *DIO_u8SetPinValue*



d. *DIO_u8GetPinValue*

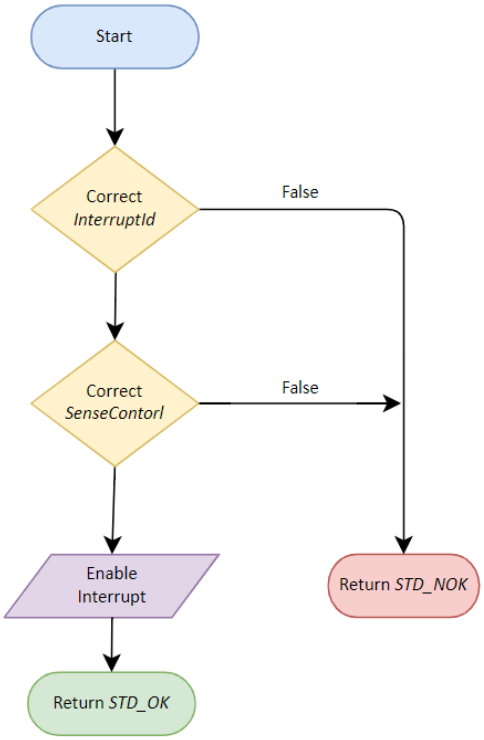


e. *DIO_u8TogglePinValue*

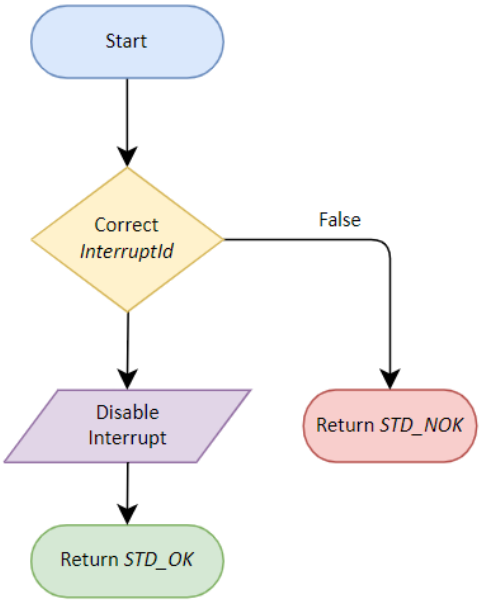


B. EXI Flowcharts

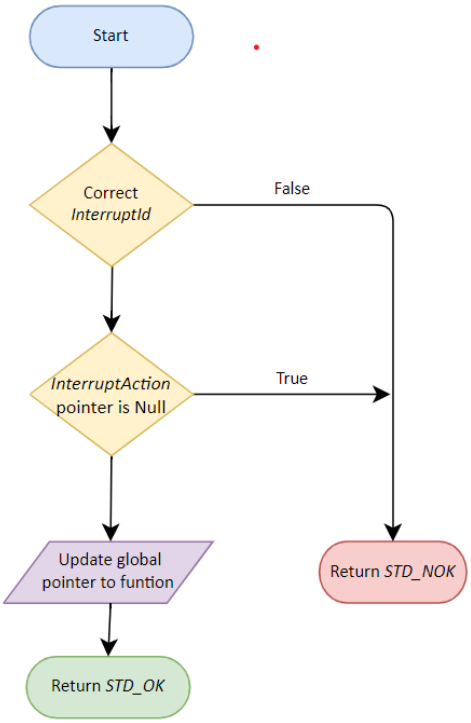
a. EXI_u8EnablePIE



b. EXI_u8DisablePIE



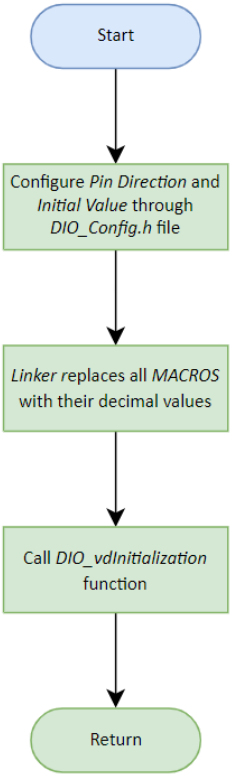
c. *EXI_u8INTSetCallBack*



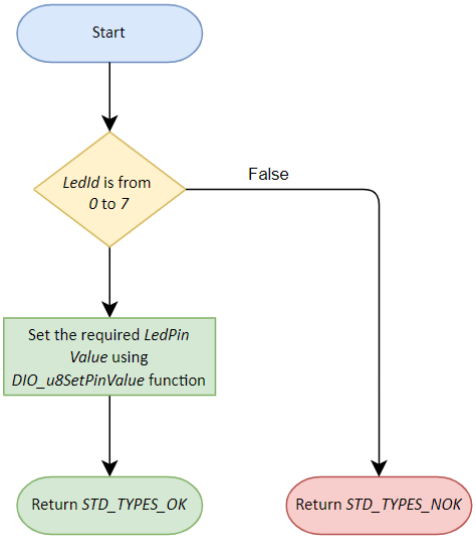
HAL Flowcharts

A. LED Flowcharts

a. LED_vdInitialization

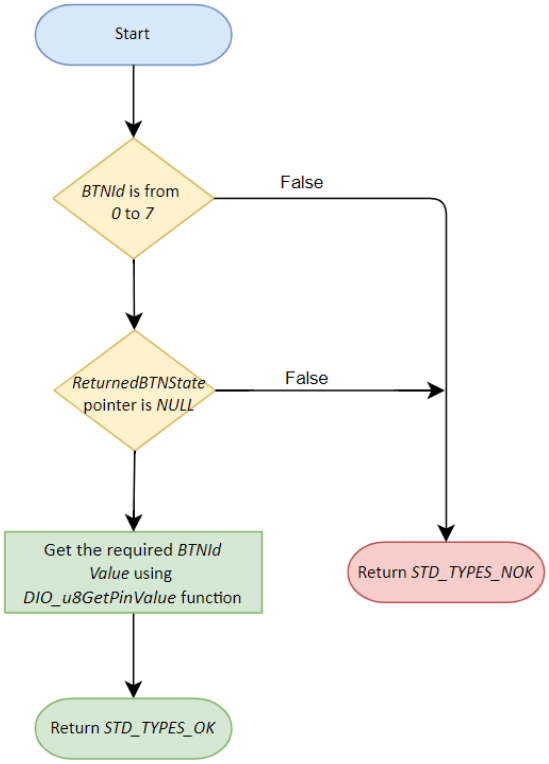


b. LED_u8SetLEDPin



B. BTN Flowcharts

a. BTN_u8GetBTNState



References:

1. *Diagrams.net*
2. *State Machine Diagram - UML 2 Tutorial | Sparx Systems*
3. *Layered Architecture | Baeldung on Computer Science*
4. *Microcontroller Abstraction Layer (MCAL) | Renesas*
5. *Hardware Abstraction Layer - an overview | ScienceDirect Topics*
6. *What is a module in software, hardware and programming?*
7. *Embedded Basics – API's vs HAL's*