Name: Abdelrhman Walaa Hussein

Moving Car System Design

Outline

This document shows a design of a four-driving wheel robot that moves in a rectangular shape.

Car Components:

- Four motors (M1, M2, M3, M4)
- One button to start (PB1)
- One button for stop (PB2)
- Four LEDs (LED1, LED2, LED3, LED4)

System Requirements:

1. The car starts initially from 0 speed.
2. When PB1 is pressed, the car will move forward after 1 second.
3. The car will move forward to create the longest side of the rectangle for 3 seconds with 50% of its maximum speed.
4. After finishing the first longest side the car will stop for 0.5 seconds, rotate 90 degrees to the right, and stop for 0.5 second.
5. The car will move to create the short side of the rectangle at 30% of its speed for 2 seconds.
6. After finishing the shortest side the car will stop for 0.5 seconds, rotate 90 degrees to the right, and stop for 0.5 second.
7. Steps 3 to 6 will be repeated infinitely until you press the stop button (PB2)
8. PB2 acts as a sudden break, and it has the highest priority.

Layered Architecture

Definition

Layered Architecture (*Figure 1*) 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

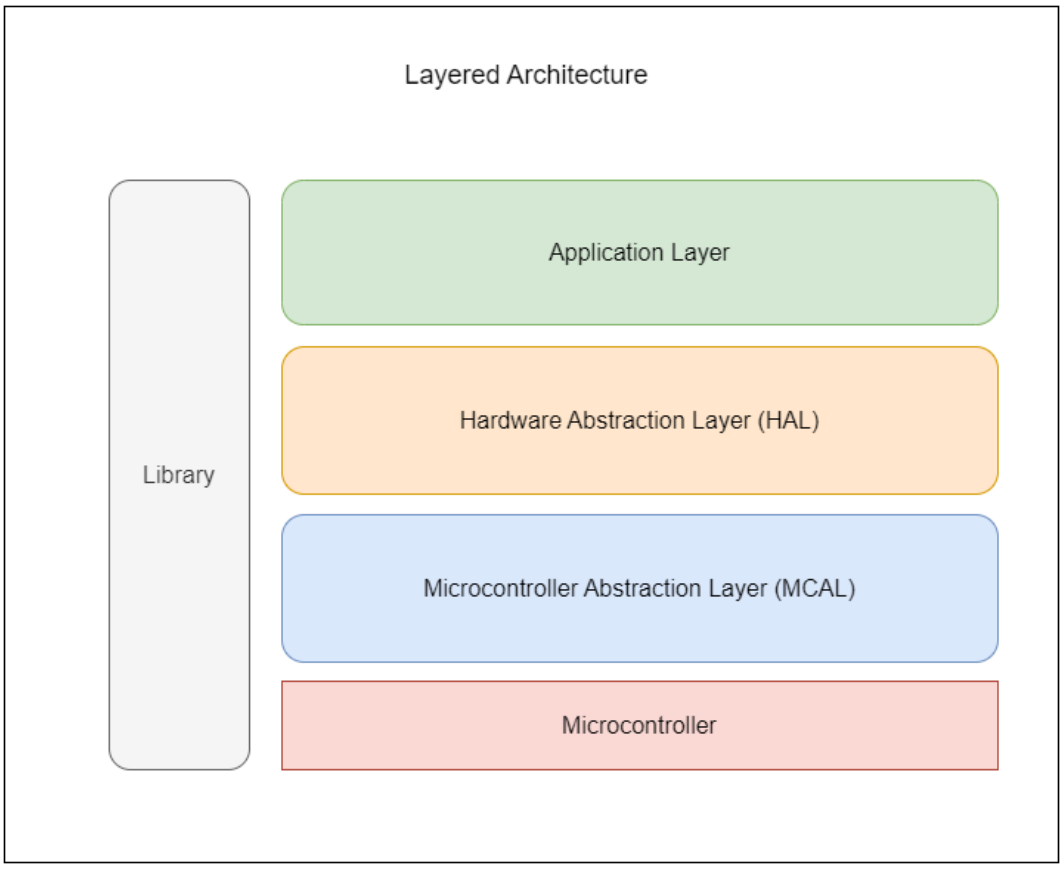

Figure 1. Layered Architecture Design

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

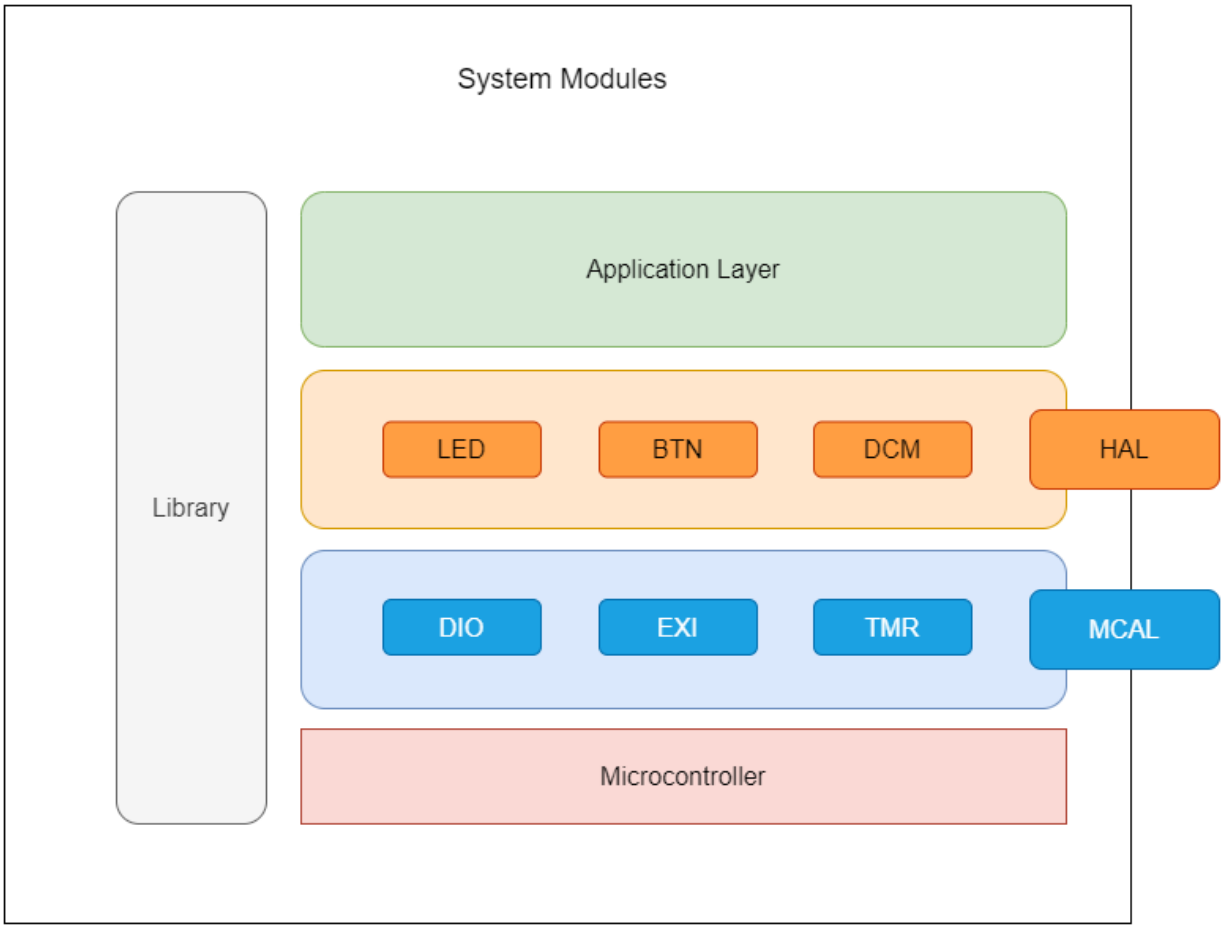

Figure 2. System Modules Design

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 bits for each interrupt, then configuring Sense Control.

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.

*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.

C. TMR APIs

vd TMR_vdTMR0Initialization (void)

Name: TMR_vdTMR0Initialization
Input: void
Output: void
Description: Function to Initialize TMR0 peripheral.

u8 TMR_u8EnableTMR (u8 Cpy_u8TimerId)

Name: TMR_u8EnableTMR
Input: u8 TimerId
Output: u8 Error or No Error
Description: Function to Enable TMR peripheral.

u8 TMR_u8DisableTMR (u8 Cpy_u8TimerId)

Name: TMR_u8DisableTMR
Input: u8 TimerId
Output: u8 Error or No Error
Description: Function to Disable TMR peripheral.

*u8 TMR_u8OVFSetCallBack (u8 Cpy_u8TimerId, void (*Cpy_pfOVFInterruptAction) (void))*

Name: TMR_u8OVFSetCallBack
Input: u8 TimerId and Pointer to function OVFInterruptAction taking void and returning 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 Timer (TimerId), the address is passed through a pointer to function (OVFInterruptAction), and then pass this address to ISR function.

*u8 TMR_u8COMPSetCallBack (u8 Cpy_u8TimerId, u8 Cpy_u8Timer1ChannelId, void (*Cpy_pfCOMPInterruptAction) (void))*

Name: TMR_u8GetOVFFlagStatus
Input: u8 TimerId, u8 Timer1ChannelId, and Pointer to function COMPInterruptAction taking void and returning 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 Timer (TimerId), the address is passed through a pointer to function (COMPInterruptAction), and then pass this address to ISR function.

HAL APIs

A. LED APIs

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.

C. DCM APIs

u8 DCM_u8RotateDCMInOneDirection (u16 Cpy_u16RotateSpeed)

Name: DCM_vdRotateDCMInOneDirection
Input: u16 RotateSpeed
Output: u8 Error or No Error
Description: Function to Rotate DCM in One Direction only.

u8 DCM_u8SetDutyCycleOfPWM (u16 Cpy_u16DutyCycleValue)

Name: DCM_u8SetDutyCycleOfPWM
Input: u8 DutyCycleValue
Output: u8 Error or No Error
Description: Function to Set Duty Cycle Value of DCM PWM.

vd DCM_vdStopDCM (void)

Name: DCM_vdStopMotor
Input: void
Output: void
Description: Function to Stop DCM.

References:

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