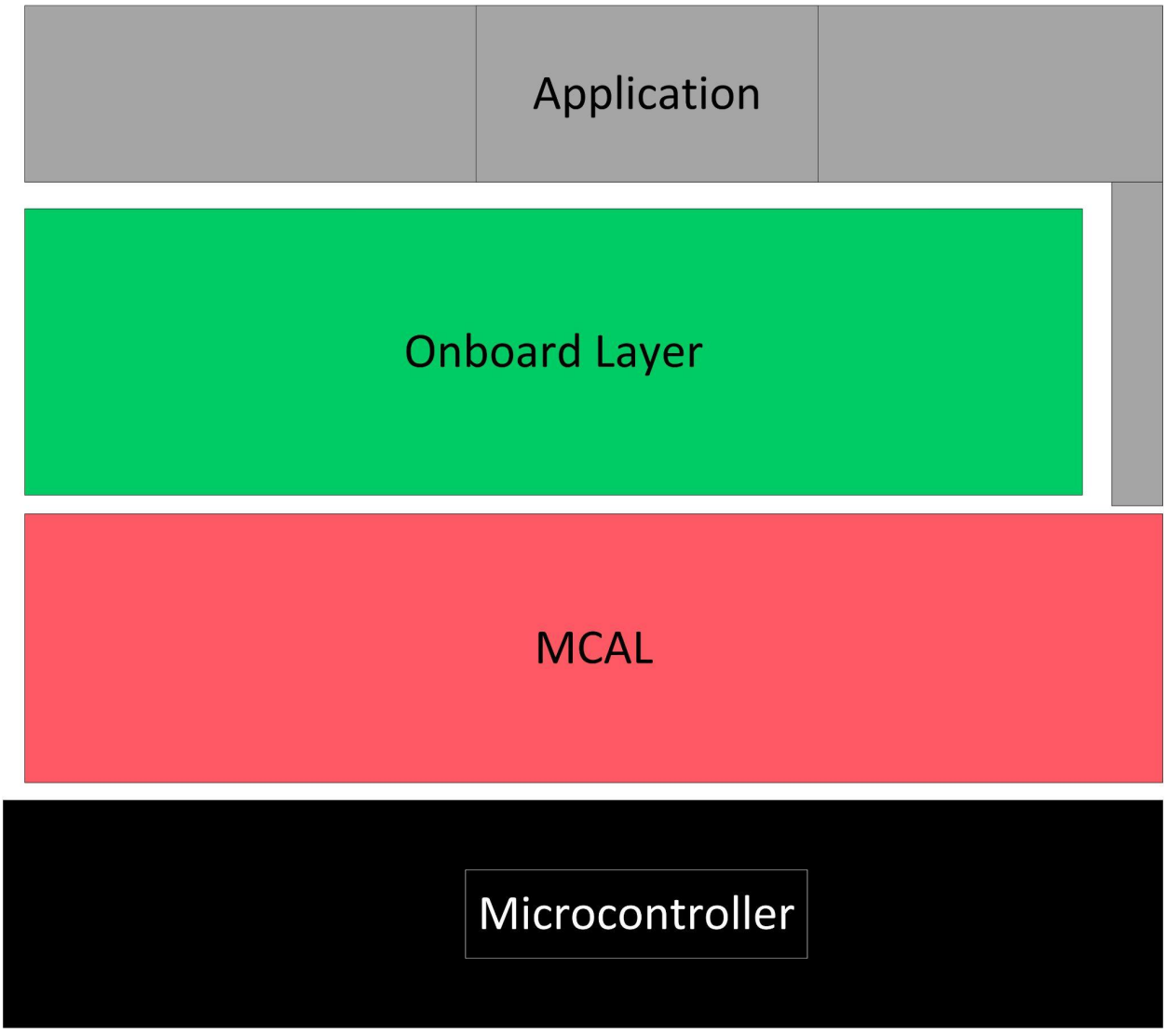
Moving Car System Design

By : Mahmoud Adel Matarawy
Email : mahmoudsarhan02672@gmail.com

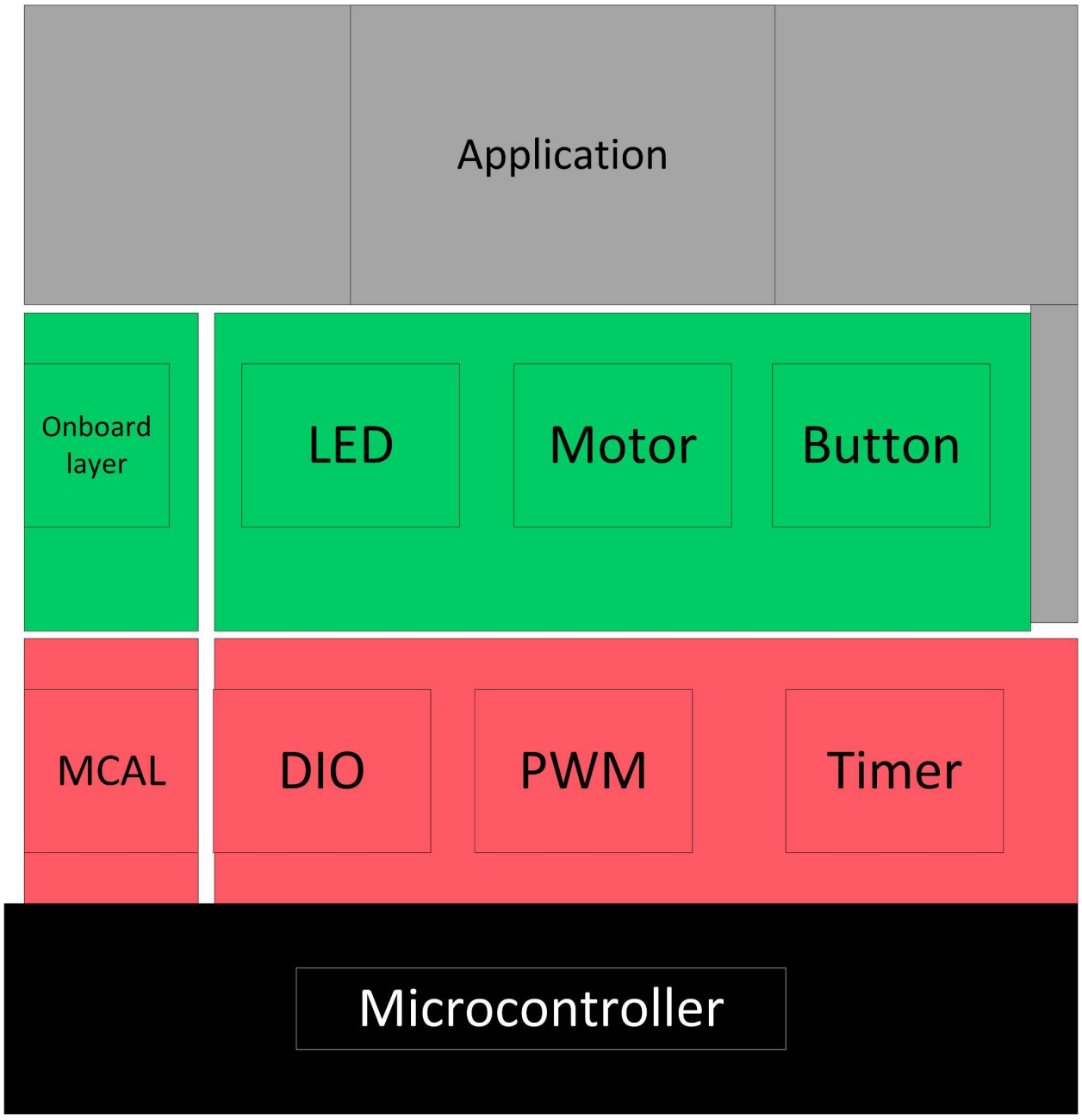
Detailed Requirements

1. Read System Requirement Specifications
    1. *Description*
        1. Car Components:
            1. Four motors (M1, M2, M3, M4)
            2. One button to start (PB1)
            3. One button for stop (PB2)
            4. Four LEDs (LED1, LED2, LED3, LED4)
        2. 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



System modules



APIs

MCAL APIs

DIO API :

Type definitions:

- Dio_ChannelType

Name	Dio_ChannelType
Type	Enumeration
Range	Shall contain all pins ID
Description	Dio_ChannelType
Available via	DIO_Config.h

- Dio_PortType

Name	Dio_PortType
Type	Enumeration
Range	Shall contain all ports ID
Description	Dio_PortType
Available via	DIO_Config.h

- DIO_Errors

Name	DIO_Errors		
Type	Enumeration		
Range	DIO_E_OK	0x00	DIO error OK
	DIO_InvalidPin	0x01	DIO error, invalid pin number.
Description	DIO Errors		
Available via	DIO.h		

- Dio_LevelType

Name	Dio_LevelType		
Type	Enumeration		
Range	STD_LOW	0x00	Physical state 0V
	STD_HIGH	0x01	Physical state 5V or 3.3V.
Description	Dio_LevelType		
Available via	DIO.h		

- Dio_DIRType

Name	Dio_DIRType		
Type	Enumeration		
Range	STD_INPUT	0x00	Set pin as input pin
	STD_OUTPUT	0x01	Set pin as output pin
Description	Dio_DIRType		
Available via	DIO.h		

Services affecting the hardware unit:

- Dio_ReadChannel

Service name	Dio_ReadChannel		
Syntax	DIO_Errors Dio_ReadChannel(Dio_ChannelType ChannelId, Dio_LevelType* level);		
Parameters (in)	ChannelId	Channel ID	
	level	Pointer to store the level	STD_HIGH
			STD_LOW
Return	DIO_Errors	DIO_E_OK	
		DIO_InvalidPin	
Description	This Function gets the level of the pin		

  - This function shall return DIO_InvalidPin if pin number is invalid.

- Dio_WriteChannel

Service name	Dio_WriteChannel		
Syntax	DIO_Errors Dio_WriteChannel(Dio_ChannelType ChannelId, Dio_LevelType level);		
Parameters (in)	ChannelId	Channel ID	
	level	Value to be set	STD_HIGH
			STD_LOW
Return	DIO_Errors	DIO_E_OK	
		DIO_InvalidPin	
Description	This Function gets the level of the pin		

- This function shall return DIO_InvalidPin if pin number is invalid.

- Dio_ChannelSetDIR

Service name	Dio_ChannelSetDIR		
Syntax	DIO_Errors Dio_ChannelSetDIR(Dio_ChannelType ChannelId, Dio_DIRType dir);		
Parameters (in)	ChannelId	Channel ID	
	dir	Value to be set	STD_INPUT
			STD_OUTPUT
Return	DIO_Errors	DIO_E_OK	
		DIO_InvalidPin	
Description	This Function sets the Direction of the pin		

- This function shall return DIO_InvalidPin if pin number is invalid.

Timer API :

Type definitions:

- Timer_config

Name	Timer_configType
Type	Structure
Description	This is the type of the external data structure containing the overall initialization data for the Timer driver
Available via	timer.h

- Timer_Status

Name	Timer_Status
Type	Enumeration

Range	Timer_S_Ready | 0x00 | Timer state Ready Timer_S_UnInit | 0x01 | Timer state UnInit
Description	Timer state
Available via	timer.h

- Timer_Errors

Name	Timer_Errors
Type	Enumeration
Range	Timer_E_OK | 0x00 | Timer error OK Timer_E_TRANSITION | 0x01 | Timer error TRANSITION Timer_E_PARAM_POINTER | 0x02 | Timer error Parameter Pointer Timer_E_INIT_FAILED | 0x03 | Timer error INIT FAILED Timer_E_InvalidValue | 0x04 | Timer error Invalid value
Description	Timer Errors
Available via	timer.h

Services affecting the hardware unit

- Timer_Init

Service name	Timer_Init	
Syntax	Timer_Errors Timer_Init(Timer_configType* config);	
Parameters (in)	config	Pointer to driver configuration
Return	Timer_Errors	Timer_E_OK

	Timer_E_TRANSITION Timer_E_PARAM_POINTER Timer_E_INIT_FAILED
Description	This Function Initialize the driver

- This function shall return Timer_E_TRANSITION if timer status is Timer_S_Ready.
- This function shall return Timer_E_PARAM_POINTER if the config pointer is NULL.

- Timer_Set

Service name	Timer_Set	
Syntax	Timer_Errors Timer_Set(Timer_Number Timer_Num, uint16_t Timer_value);	
Parameters (in)	Timer_Num	Timer Number
	Timer_value	Value will be stored in timer counter register
Return	Timer_Errors	Timer_E_OK Timer_E_TRANSITION Timer_E_InvalidValue
Description	This Function Set timer counter with value	

- This function shall return Timer_E_TRANSITION if timer status is Timer_S_UnInit
- This function shall return Timer_E_InvalidValue if the passed value is more than timer capacity.

- Timer_DeInit

Service name	Timer_DeInit
Syntax	Timer_Errors Timer_DeInit(

	Timer_Number Timer_Num);	
Parameters (in)	Timer_Num	Timer Number
Return	Timer_Errors	Timer_E_OK Timer_E_TRANSITION
Description	This Function DeInitialize the driver	

- This function shall return Timer_E_TRANSITION if timer status is Timer_S_UnInit.

PWM API :

Services affecting the hardware unit:

- Set_Duty

Service name	Set_Duty	
Syntax	Timer_Errors Set_Duty(Timer_Number Timer_Num, uint16_t duty);	
Parameters (in)	Timer_Num	Timer Number
	duty	Value will be stored in timer output compare register
Return	Timer_Errors	Timer_E_OK Timer_E_TRANSITION Timer_E_InvalidValue
Description	This Function Set duty cycle in percentage.	

- This function shall return Timer_E_TRANSITION if timer status is Timer_S_UnInit
- This function shall return Timer_E_InvalidValue if the passed value is more than timer capacity.

Onboard APIs

LED API:

No APIs needed for the current requirements.

Motor API:

Type definitions:

- MOTOR_ID_Type

Name	MOTOR_ID_Type		
Type	Enumeration		
Range	MOTORS_RIGHT	0x00	2 Motors in right side
	MOTORS_LEFT	0x01	2 Motors in left side
Description	MOTOR ID Enum		
Available via	motor.h		

- MOTOR_DIR_Type

Name	MOTOR_DIR_Type		
Type	Enumeration		
Range	MOTOR_FORWARD	0x00	Forward Direction
	MOTOR_BACKWARD	0x01	Backward Direction
Description	MOTOR ID Enum		
Available via	motor.h		

Services affecting the hardware unit:

- motorStart

Service name	motorStart	
Syntax	void motorStart(MOTOR_ID_Type motor);	
Parameters (in)	motor	Right 0x00, Left 0x01
Return	NONE	
Description	This Function Starts The motor.	

- motorStop

Service name	motorStop	
Syntax	void motorStop(MOTOR_ID_Type motor);	
Parameters (in)	motor	Right 0x00, Left 0x01
Return	NONE	
Description	This Function Stops The motor.	

- motorSet_dir

Service name	motorSet_dir	
Syntax	void motorSet_dir(MOTOR_ID_Type motor, MOTOR_DIR_Type dir);	
Parameters (in)	motor	Right 0x00, Left 0x01
	dir	Forward 0x00, Backward 0x01
Return	NONE	
Description	This Function Sets the direction of The motor.	

- motorSet_speed

Service name	motorSet_speed

Syntax	void motorSet_speed(MOTOR_ID_Type motor, uint8_t speed);	
Parameters (in)	motor	Right 0x00, Left 0x01
	speed	Speed in percentage
Return	NONE	
Description	This Function Sets the speed of The motor.	

- motor_RotateLeft

Service name	motor_RotateLeft
Syntax	void motor_RotateLeft(void);
Parameters (in)	NONE
Return	NONE
Description	This Function Rotate to left.

- motor_RotateRight

Service name	motor_RotateRight
Syntax	void motor_RotateRight(void);
Parameters (in)	NONE
Return	NONE
Description	This Function Rotate to right.

Button API:

Type definitions:

- Button_configType

Name	Button_configType
Type	Structure
Description	This is the type of the external data structure containing the overall configuration data for the Button API
Available via	Button_Types.h

- Button_LevelType

Name	Button_LevelType		
Type	Enumeration		
Range	BT_PUSH_LEVEL	0x00	Push Level
	BT_RELEASE_LEVEL	0x01	Release Level
Description	Button Level Enum		
Available via	Button_Types.h		

- Button_StateType

Name	Button_StateType		
Type	Enumeration		
Range	BT_PRE_PUSH	0x00	Pre Push Level
	BT_PUSHED	0x01	Pushed Level
	BT_PRE_HOLD	0x02	Pre Hold Level
	BT_HOLD	0x03	Hold Level
	BT_PRE_RELEASE	0x04	Pre Release Level
	BT_RELEASED	0x05	Released Level
	BT_UNDEFINED	0x06	Undefined
Description	Button state Enum		
Available via	Button_Types.h		

- Button_IdType

Name	Button_IdType		
Type	Enumeration		
Range	Button_Start	0x00	Start Button
	Button_Stop	0x01	Stop Button
Description	Button Id Enum		
Available via	Button_Types.h		

Services affecting the hardware unit:

- getButtonState

Service name	getButtonState	
Syntax	Button_StateTyp getButtonState(Button_IdType enmButtonId);	
Parameters (in)	enmButtonId	Start 0x00, Stop 0x01
Return	Button_StateTyp	BT_PRE_PUSH
		BT_PUSHED
		BT_PRE_HOLD
		BT_HOLD
		BT_PRE_RELEASE
		BT_RELEASED
		BT_UNDEFINED
Description	This Function gets the Button state.	

App APIs

App API:

Services affecting the hardware unit:

- appStart

Service name	appStart
Syntax	void appStart(void);
Parameters (in)	NONE
Return	NONE
Description	This Function Start the application.