On-Demand Traffic Light Control

Delievered By: Ahmed Genina

➢ Table of Contents

➢ Introduction

Traffic lights are signaling devices positioned at road intersections, pedestrian crossings, and other locations to control the flow of traffic.

Traffic lights normally consist of three signals, transmitting meaning to drivers and riders through colors and symbols including arrows and bicycles.

The regular traffic light colors are red, yellow, and green arranged vertically or horizontally in that order.

Although this is internationally standardized, variations exist on national and local scales as to traffic light sequences and laws.

The system is a demonstration of traffic lights system with an on-demand crosswalk button.

Crosswalk buttons let the signal operations know that someone is planning to cross the street, so the light adjusts, giving the pedestrian enough time to get across.

System Description

1. ATmega32 microcontroller

2. One push button connected to INT0 pin for pedestrian
3. Three LEDs for cars - Green, Yellow, and Red, connected on port A, pins 0, 1, and 2
4. Three LEDs for pedestrians - Green, Yellow, and Red, connected on port B, pins 0, 1, and 2 The system is divided into 2 modes:

 Normal Mode:

1. Cars' LEDs will be changed every five seconds starting from Green then yellow then red then yellow then Green.
2. The Yellow LED will blink for five seconds before moving to Green or Red LEDs.

- Change from normal mode to pedestrian mode when the pedestrian button is pressed

- **Pedestrian Mode:**

1. If pressed when the cars' Red LED is on, the pedestrian's Green LED and the cars' Red LEDs will be on for five seconds, this means that pedestrians can cross the street while the pedestrian's Green LED is on.
2. If pressed when the cars' Green LED is on or the cars' Yellow LED is blinking, the pedestrian Red LED will be on then both Yellow LEDs start to blink for five

seconds, then the cars' Red LED and pedestrian Green LEDs are on for five seconds, this means that pedestrian must wait until the Green LED is on.

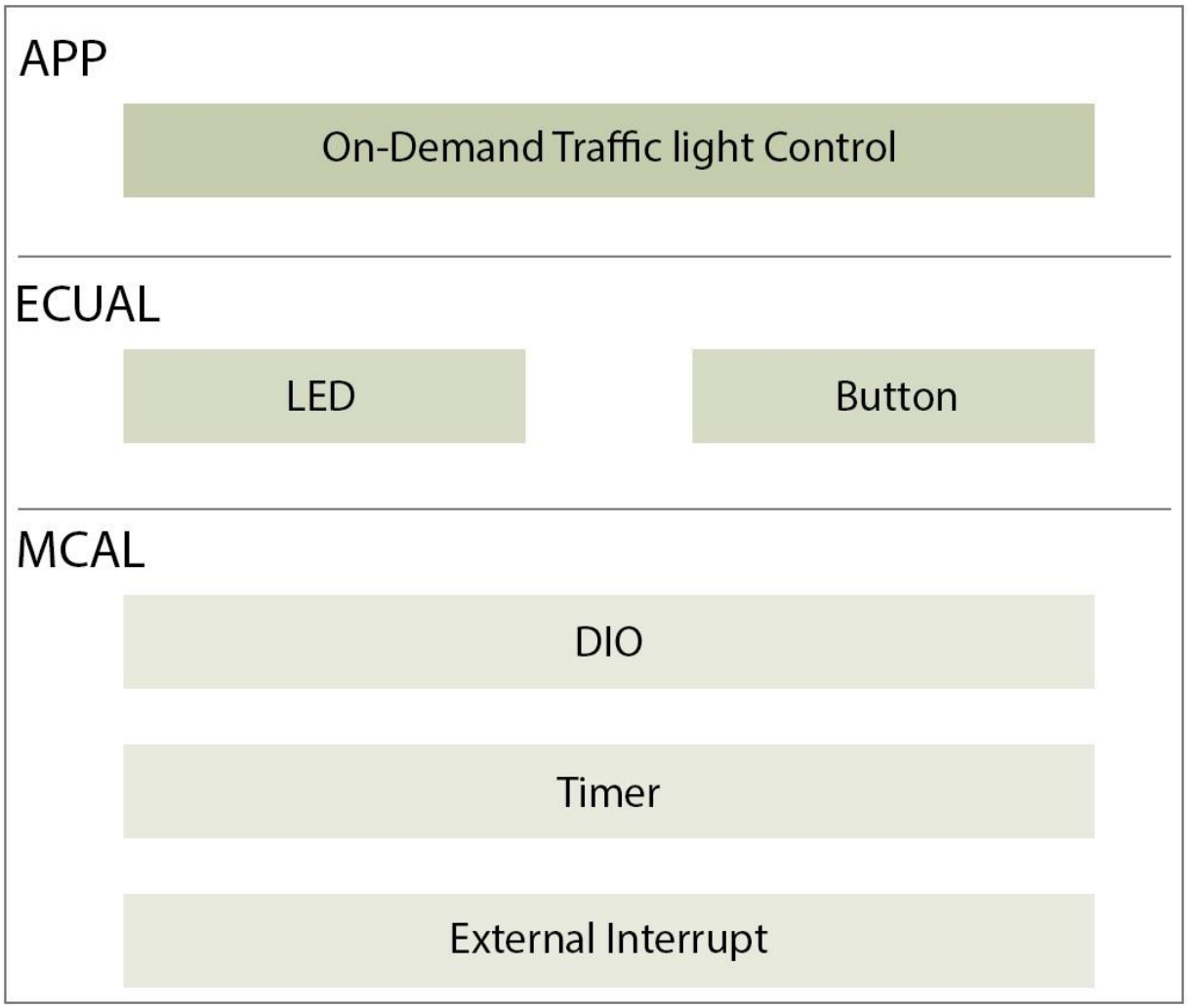
3. At the end of the two states, the cars' Red LED will be off and both Yellow LEDs start blinking for 5 seconds and the pedestrian's Green LED is still on.
4. After the five seconds the pedestrian Green LED will be off and both the pedestrian Red LED and the cars' Green LED will be on.
5. Traffic lights signals are going to the normal mode again.

System Design

I made a full static architecture for my system which divides the system into 3 layers:

- Application Layer
- ECUAL (ECU Abstraction Layer)

- MCAL (Microcontroller Abstraction Layer) The design follows SOLID principles.



- **Application Layer:**

Contains two files : app.c , app.h

Application contains the following functions:

1. EN_APPState_t **APP_VOIDInit**(**void**);

Which initializes all drivers in order by checking the error state of each initialization and returns APP_OK if initialization is done succesfuly & APP_NOT_OK if any error occurred during initialization.

2. EN_APPState_t **APP_VOIDStart**(**void**);

Which starts the application by only running the normal mode by calling the Normal Mode function, as the pedestrian mode only executed when an interrupt is occurred on the INT0 external interrupt pin.

3. **void APP_VOIDNormalMode**(**void**);

Which runs the normal operation of the traffic light system for cars which is defined in the system description, it also stores the value of the current car traffic light state to be used in the pedestrian mode function.

4. **void APP_VOIDPedestrianMode(void);**

Which runs the pedestrian operation of the traffic light system for pedestrians which is defined in the system description, it uses the state of the car traffic light to decide which sequence it will perform.

- **ECUAL Layer:**

Contains two Modules: LED & Button LED

contains the following functions:

1. En_LEDState_t **HLED_VOIDInit**(uint8_t Copy_U8Port, uint8_t Copy_U8Pin);

Which takes Port & Pin input to configure this specific pin inside this specific port to be output for LEDs and return the state of the function as LED_OK if nothing went wrong & LED_NOT_OK otherwise.

2. En_LEDState_t **HLED_VOIDON**(uint8_t Copy_U8Port, uint8_t Copy_U8Pin);

Which takes Port & Pin input to set this specific pin inside this specific port to be ON and return the state of the function as LED_OK if nothing went wrong & LED_NOT_OK otherwise.

3. En_LEDState_t **HLED_VOIDOFF**(uint8_t Copy_U8Port, uint8_t Copy_U8Pin);

Which takes Port & Pin input to set this specific pin inside this specific port to be OFF and return the state of the function as LED_OK if nothing went wrong & LED_NOT_OK otherwise.

4. En_LEDState_t **HLED_VOIDToggle**(uint8_t Copy_U8Port, uint8_t Copy_U8Pin);

Which takes Port & Pin input to toggle this specific pin inside this specific port and return the state of the function as LED_OK if nothing went wrong & LED_NOT_OK otherwise.

Button contains the following functions:

1. En_ButtonState_t **HBTN_VOIDInit**(uint8_t Copy_U8Port, uint8_t Copy_U8Pin);

Which takes Port & Pin input to configure this specific pin inside this specific port to be input for Buttons and return the state of the function as BUTTON_OK if nothing went wrong & BUTTON _NOT_OK otherwise.

- **MCAL Layer:**

Contains three Modules: Timer, External Interrupt, DIO

Timer contains the following functions:

1. En_TIMERState_t **MTMR0_VOIDCTCInit** (**void**);

Which initialize the timer in CTC mode and return the state of the function as TIMER_OK if nothing went wrong & TIMER_NOT_OK otherwise.

2. En_TIMERState_t **MTMR0_VOIDCTCStart** (uint16_t Copy_U16Second);

Which takes Second input argument which defines the number of seconds that the user wants this function to delay the program with, and return the state of the function as TIMER_COMPLETE if timer reached the required value of time delay & TIMER_NOT_COMPLETE otherwise.
Prescaler value: /256
Value in OCR0 register = 125
MCU frequency = 8Mhz Calculations:
Tov = (125 * 256)/(8000000) = 4ms
Required time = “Second”/ 4ms = “Second” * 125 So, the delay will be complete if a counter which starts at 0 reaches the value = “Second” * 125

External Interrupt contains the following functions:

1. En_EXINTState_t **MEXINT_VOIDGIEEnable** (**void**);

Which enables the General Interrupts by setting the I-bit to 1 in SREG register and return the state of the function as

EXINT_OK if nothing went wrong & EXINT_NOT_OK otherwise.

2. En_EXINTState_t **MEXINT_VOIDGIEDisable** (**void**);

Which disables the General Interrupts by clearing the I-bit to 0 in SREG register and return the state of the function as EXINT_OK if nothing went wrong & EXINT_NOT_OK otherwise.

3. En_EXINTState_t **MEXINT0_VOIDInit** (**void**);

Which initialize the external interrupts on INT0 pin by setting bits ISC01 ISC00 to 11 to configure the interrupt sense to rising edge, and return the state of the function as EXINT_OK if nothing went wrong & EXINT_NOT_OK otherwise.

4. En_EXINTState_t **MEXINT0_VOIDEnableInt** (**void**);

Which enables the INT0 interrupt by GIE Enable function and Peripheral interrupt, and return the state of the function as EXINT_OK if nothing went wrong & EXINT_NOT_OK otherwise.

5. En_EXINTState_t **MEXINT0_VOIDEnableInt** (**void**);

Which disables the INT0 interrupt by GIE Disable function and Peripheral interrupt, and return the state of the function as EXINT_OK if nothing went wrong & EXINT_NOT_OK otherwise.

6. En_EXINTState_t **MEXINT_VOIDINT0CallBackFunction** (**void** (* Copy_VOIDCallBackPtr) (**void**));

Which takes a pointer to function as input and store its address in a global pointer to function, this global pointer is used after that in the ISR handler of the INT0 interrupt, its purpose is to use the App Pedestrian mode function in the ISR of a lower layer module, and return the state of the function as EXINT_OK if nothing went wrong & EXINT_NOT_OK otherwise.

DIO contains the following functions:

1. En_DIOState_t **MDIO_VOIDSetPinDirection** (uint8_t Copy_U8Port, uint8_t Copy_U8Pin, uint8_t Copy_U8Direction);

Which takes the Port, Pin & Direction as inputs and configure this specific pin to be input/output depending on the direction, and returns DIO_OK if nothing went wrong and DIO_NOT_OK otherwise.

2. En_DIOState_t **MDIO_VOIDSetPortDirection**

(uint8_t Copy_U8Port, uint8_t Copy_U8Direction);

Which takes the Port & Direction as inputs and configure this port to be input/output depending on the combination defined in Direction, and returns DIO_OK if nothing went wrong and DIO_NOT_OK otherwise.

3. En_DIOState_t **MDIO_VOIDSetPinValue** (uint8_t Copy_U8Port, uint8_t Copy_U8Pin, uint8_t Copy_U8Value);

Which takes the Port, Pin & Value as inputs and sets the value of this specific pin to be HIGH/LOW depending on the Value, and returns DIO_OK if nothing went wrong and DIO_NOT_OK otherwise.

4. En_DIOState_t **MDIO_VOIDSetPortValue** (uint8_t Copy_U8Port, uint8_t Copy_U8Value);

Which takes the Port & Value as inputs and sets this port to be high/low depending on the combination defined in Value, and returns DIO_OK if nothing went wrong and DIO_NOT_OK otherwise.

5. En_DIOState_t **MDIO_VOIDTogglePinValue** (uint8_t Copy_U8Port, uint8_t Copy_U8Pin);

Which takes the Port, Pin as inputs and toggles the value of this specific pin to be HIGH/LOW depending on its previous value,

and returns DIO_OK if nothing went wrong and DIO_NOT_OK otherwise.

6. En_DIOState_t **MDIO_U8GetPinValue** (uint8_t Copy_U8Port, uint8_t Copy_U8Pin);

Which takes the Port, Pin as inputs and reads the value of this specific pin and returns DIO_HIGH when value is HIGH and DIO_LOW when value is LOW.

➢ System Flow Chart