On-demand Traffic light control

By:

Peter Magdy (peter.magdy.zaky@gmail.com)

Table of Contents

System Description:

The required system is a traffic light on-demand control system, with the goal of organizing the flow of vehicles and the flow of pedestrians that want to pass the street. The system allows the normal flow of vehicles by the mean of a normal traffic light (red, yellow and green light) until a pedestrian wants to cross the street, which is signaled by pressing a button. When the button is pressed the system starts to transition to allow for the passing of pedestrians by activating the pedestrian traffic light.

So the system consists of 2 traffic lights (3 LEDs each; red, yellow and green) and 1 button. In the normal mode the normal traffic light transitions from red to yellow to green to yellow to red on repeat where each LED stays on for 5 seconds. In case of the yellow LED it will blink for 5 seconds instead of constantly staying on. During this time the pedestrian red LED is the only one active.

When the pedestrian button is pressed the system engages in the pedestrian mode. The behavior in this mode is split into 2 parts. The first part is until the green pedestrian light turns on and it depends on the state of the system before the button was pressed. If the normal red light was active then the green pedestrian light activates and this state is maintained for 5 seconds. If the normal green light was active then the pedestrian red light remains active for 5 seconds then both traffic lights switch to blinking yellow for 5 seconds and finally the green pedestrian light and the red normal light activate for 5 seconds. In the case that the normal yellow light is active when the button is pressed then both yellow lights blink for 5 seconds and then the pedestrian green light and the normal red light activate for 5 seconds. This is all the cases for the first part.

The second part starts after the pedestrian green light has been on for 5 seconds. Both yellow lights will blink for 5 seconds WHILE the green pedestrian light remains on. Then the green pedestrian light turns off as well as both yellow lights and the red pedestrian light is active with the green normal light. After that the system reverts back to normal mode.

System Design:

High Level Design:

The high level design is in the form of a layered architecture. There are 4 layers from bottom to top: the Microcontroller layer, the Microcontroller Abstraction Layer (MCAL), Electronic Unit Abstraction Layer (ECUAL) and the Application layer.

The microcontroller layer is all the functions of the microcontroller which is an ATMega32 in this case. It includes setting the relevant registers to accomplish different functions such as controlling DIO pins or timers.

The **MCAL** abstracts dealing with the microcontroller into a set of easy to handle APIs within drivers:

DIO Driver:

Int initDIO(portNumber, pinNumber, state): This API takes the port and pin of a DIO and sets it to output or input according to the required state. The output would be 0 if the pin was set successfully or 1 if the port or pin or state was not valid.

Int enInterrupt(INTx): This API enables the required interrupt. Output is 0 for successful operation and 1 for failed operation (e.g.: due to invalid interrupt).

Int writeDIO(portNumber, pinNumber, value): This API sets an output DIO according to the required value. The output is 0 for a successful operation and 1 for any error.

Timers Driver:

Int setTimer0(ms): This API sets Timer 0 to the required milliseconds. Output is 0 for successful operation and 1 for any error.

The **ECUAL** abstracts dealing with any output devices by using the APIs provided by the MCAL using the following APIs: (All APIs have output of 0 for successful operation and output 1 for error)

LED Driver:

Int initLED(ledPort, ledPin): initializes the respective DIO pin as output.

Int onLED(ledPort, ledPin): sets the required LED to on by controlling the respective DIO pin.

Int offLED(ledPort, ledPin): sets the required LED to off by controlling the respective DIO pin.

Int blinkLED(ledPort, ledPin, time_per_blink, number_of_blinks): Turns the LED On then Off where the on time and off time is equal time_per_blink/2 each. This operation is repeated according to the number_of_blinks.

Button Driver:

Int initButton(INTx): initializes the respective DIO pin as input and activates the interrupt.

The application layer uses the APIs in the ECUAL layer to achieve the required function. The exact logic is discussed in the system flow chart section. It is divided into 2 APIs initApp() which initializes the button and 6 LEDs required and startApp() which achieves the required logic described in the system description.

Low Level Design:

initDIO(portNumber, pinNumber, state):

1- set the respective bit (according to pin number) in the respective DDRx register (according to portNumber) to 0 if the state is input and 1 if the state is output using a bitwise operation.

enInterrupt(INTx):

1- set the required ISC registers to rising edge operation according to the chosen interrupt (INT0, INT1, INT2)
2- enable the relevant bit in the GICR.

writeDIO(portNumber, pinNumber, value):

1- write 0 or 1 according to the required value on the respective bit in the respective PORTx register according to the pinNumber and PortNumber.

setTimer0(ms):

1- write 0 in WGM00 and WGM01 in register TCCR0 to initialize normal mode.
2- Set TCNT0 to the required number (calculated below).
3- Write 1 in TOIE0 in TIMSK register to activate interrupt.
4- Write 1 0 1 to CS02 CS01 CS00 to use clk/1024 prescaler.
5- Set TCNT0 to the required number again each time an interrupt happen until the required number of overflows is reached.
6- Write 0 0 0 to CS02 CS01 CS00 to turn off the clock.

Calculations:

Ttick = 1024/(16*10^6);
Tmax_delay = 2^8 * Ttick;
N_overflows = ceil(ms*10^-3/Tmax_delay);
Timer_initial_value = 2^8 – (ms/Ttick)/N_overflows

initLed(ledPort, ledPin): call initDIO(ledPort, ledPin, OUTPUT)

onLED(ledPort, ledPin): call writeDIO(ledPort, ledPin, HIGH)

offLED(ledPort, ledPin): call writeDIO(ledPort, ledPin, LOW)

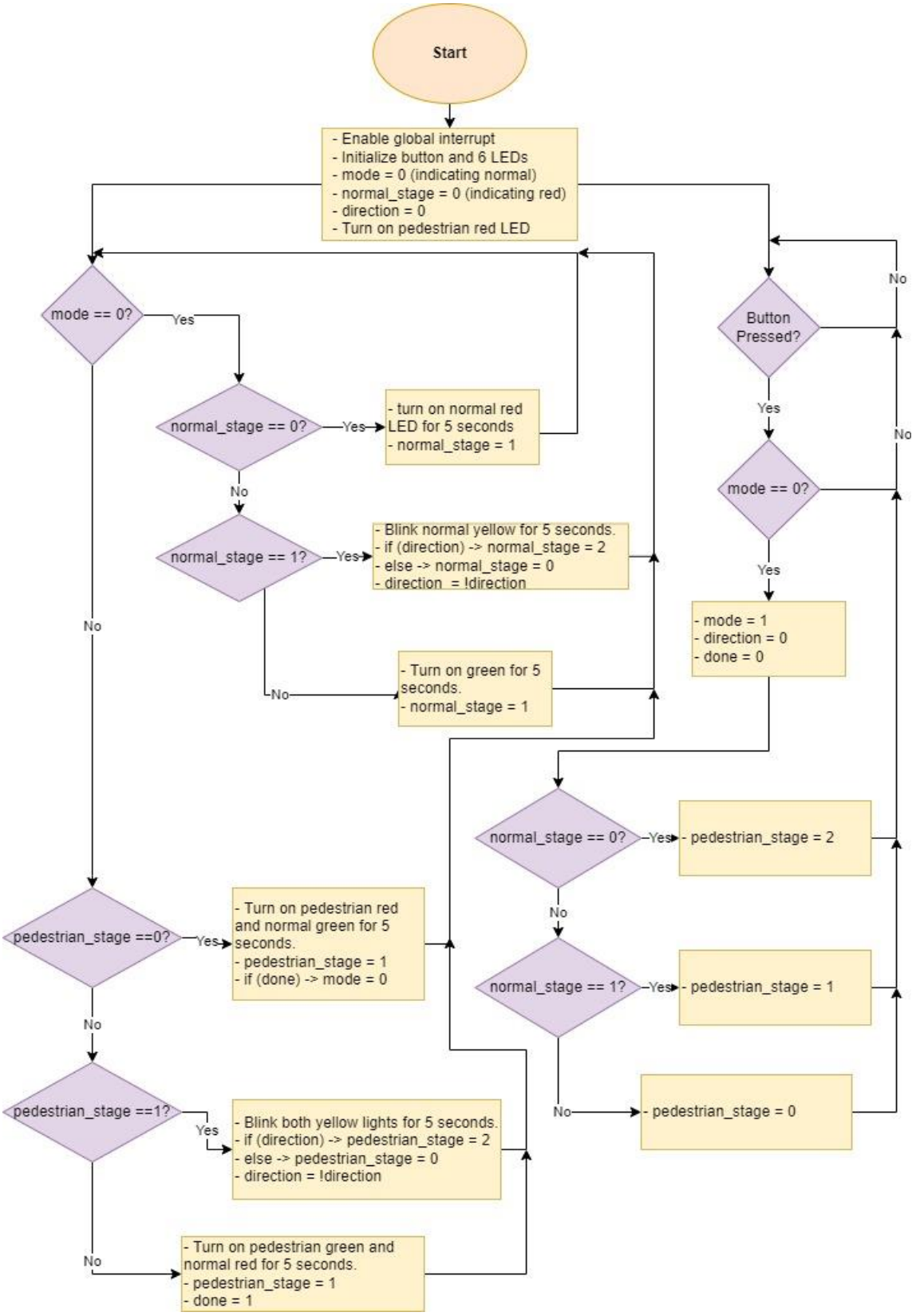
blinkLED(ledPort, ledPin, time_per_blink, number_of_blinks):

1- Call onLED(ledPort, ledPin)
2- Call setTimer0(time_per_blink/2)
3- Call offLED(ledPort, ledPin)
4- Call setTimer0(time_per_blink/2)
5- Repeat steps 1-4 by number_of_blinks.

initButton(INTx):

1- Call initDIO(buttonPort, buttonPin, INPUT) – buttonPort and buttonPin can be obtained through switch cases from INTx.
2- Call enInterrupt(INTx)

System Flow Chart:



System Constraints:

The system requires additional error handling conditions.

The system requires hardware considerations such as debouncing of the button.

The application layer had to communicate with the timer in the MCAL.