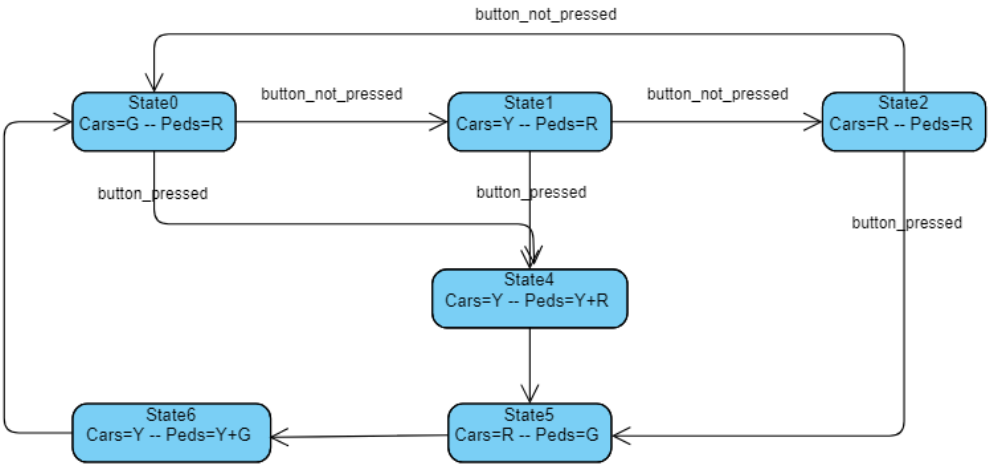
Project documentation

System Description

The system consists of the MCU and 3 peripherals that interact with it: two sets of LEDs and a button. The application is written such that the MCU drives the LEDs by default, which is the “normal mode”. When the button is pressed the MCU handles the event by entering a certain series of application states before returning to normal mode again.

System state machine

- States 0,1,2 → the normal mode
- States 4,5,6 → the pedestrian mode



System Design: static architecture of the system

System layers

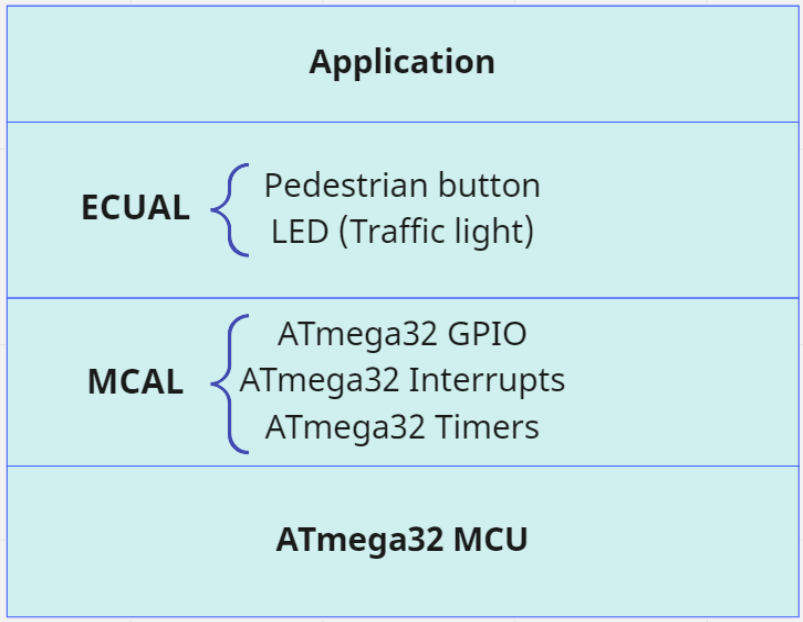
- Application
- Electronic Unit Abstraction Layer (ECUAL)
- Microcontroller Abstraction Layer (MCAL)
- Microcontroller

System drivers

- Pedestrian button
- LED (Traffic light)
- ATmega32 GPIO

- ATmega32 Interrupts
- ATmega32 Timers

Drivers' places in the layers



APIs for each driver

ATmega32 GPIO

- Init: choose the pin, make it an input or output
    - Input: port, pin number, pin direction
    - Output: nothing
    - Return: error_state
- Set_value: put HIGH on the connected pin OR put LOW on the connected pin
    - Input: port, pin number, desired value
    - Output: nothing
    - Return: error_state
- Get_value: read value on the pin
    - Input: port, pin number
    - Output: read value
    - Return: error_state
- Toggle: change the state of the pin (HIGH → LOW OR LOW → HIGH)
    - Input: port, pin number
    - Output: nothing
    - Return: error_state
- Driver data types
    - Enum Port: PORT_A, PORT_B

Pedestrian button

- Init: choose the pin, make it an input
    - Input: port, pin number
    - Output: nothing
    - Return: error_state
- Read_state: read value on the pin that's connected to the button
    - Input: port, pin number
    - Output: pin value
    - Return: error_state
- Driver data types
    - None

LED (Traffic light)

- Init: choose the pin, make it an output
    - Input: port, pin number
    - Output: nothing
    - Return: error_state
- On: put HIGH on the connected pin
    - Input: port, pin number
    - Output: nothing
    - Return: error_state
- Off: put LOW on the connected pin
    - Input: port, pin number
    - Output: nothing
    - Return: error_state
- Toggle_led: change the state of the LED (ON → OFF OR OFF → ON)
    - Input: port, pin number
    - Output: nothing
    - Return: error_state
- Driver data types:
    - None

ATmega32 Timers

- Init: set mode of operation, set timer initial value, enable/disable interrupts, "*choose*" the prescaler value, compare_match value,
    - Input: mode of operation (normal/CTC/…), initial value, prescaler value, interrupt configuration
    - Output: nothing
    - Return: error_state
- Start: set the chosen prescaler value to start counting
    - Input: prescaler value
    - Output: nothing
    - Return: error_state
- Stop: stop the timer

        - Input: nothing
        - Output: nothing
        - Return: error_state
- Read_state: read the status flags
    - Input: the desired flag to be known (timer mode)
    - Output: flag value
    - Return: error_state
- Driver data types
    - Enum: Timer_Mode: normal, CTC, …

ATmega32 Interrupts

- interrupt_en: enables interrupts from a certain source and with a certain trigger
    - Input: interrupt source and trigger type
    - Output:
    - Return:
- Driver data types
    - Enum: three source of external interrupts `{INT_0, INT_1, INT_2}`
    - Enum: types of triggers `{LOW_LVL, ANY_CHNG, FAL_EDG, RIS_EDG}`

System constraints

There are no explicit constraints (power, cost, …) except the timing between application states. However, the timing is not so critical (no hard deadlines). Working with the AVR ATmega32 specifically can also be considered a constraint.