

On-Demand Traffic Control

Generated by Doxygen 1.9.4

1 System static architecture	1
1.1 system description	1
1.1.1 Hardware	1
1.1.2 Software & IDEs	1
1.2 system design	1
1.3 State machine	2
2 Module Index	3
2.1 Modules	3
3 File Index	5
3.1 File List	5
4 Module Documentation	7
4.1 Application layer	7
4.1.1 Detailed Description	8
4.1.2 Macro Definition Documentation	9
4.1.2.1 CARS_GREEN_LED	9
4.1.2.2 CARS_RED_LED	9
4.1.2.3 CARS_YELLOW_LED	9
4.1.2.4 PEDESTRIANS_GREEN_LED	9
4.1.2.5 PEDESTRIANS_RED_LED	9
4.1.2.6 PEDESTRIANS_YELLOW_LED	10
4.1.3 Enumeration Type Documentation	10
4.1.3.1 EN_TrafficStates_t	10
4.1.4 Function Documentation	10
4.1.4.1 app_start()	10
4.1.4.2 carsGreenState()	11
4.1.4.3 carsRedState()	11
4.1.4.4 carsYellowState()	12
4.1.4.5 pedestriansGreenState()	12
4.1.4.6 pedestriansRedState()	12
4.1.4.7 pedestriansYellowState()	13
4.1.4.8 trafficLightOperation()	13
4.2 ECUAL layer	14
4.2.1 Detailed Description	14
4.3 Button driver	14
4.3.1 Detailed Description	14
4.3.2 Function Documentation	14
4.3.2.1 buttonInit()	15
4.3.2.2 buttonRead()	16
4.4 LED driver	16
4.4.1 Detailed Description	17

4.4.2 Function Documentation	17
4.4.2.1 ledInit()	17
4.4.2.2 ledOff()	17
4.4.2.3 ledOn()	18
4.4.2.4 ledToggle()	18
4.5 MCAL layer	19
4.5.1 Detailed Description	19
4.6 DIO driver	19
4.6.1 Detailed Description	20
4.6.2 Function Documentation	20
4.6.2.1 DIO_pinInit()	20
4.6.2.2 DIO_pinRead()	20
4.6.2.3 DIO_pinToggle()	21
4.6.2.4 DIO_pinWrite()	21
4.7 ATMEGA32 external interrupts driver	22
4.7.1 Detailed Description	23
4.7.2 Macro Definition Documentation	23
4.7.2.1 INT0_PIN	24
4.7.2.2 INT1_PIN	24
4.7.2.3 INT2_PIN	24
4.7.2.4 ISC00	24
4.7.2.5 ISC01	24
4.7.2.6 ISC10	25
4.7.2.7 ISC11	25
4.7.2.8 ISC2	25
4.7.3 Enumeration Type Documentation	25
4.7.3.1 EN_interruptError_t	25
4.7.3.2 EN_interruptNum_t	26
4.7.3.3 EN_interruptSenseControl_t	26
4.7.4 Function Documentation	27
4.7.4.1 Ext_interruptInit()	27
4.8 Interrupts driver	27
4.8.1 Detailed Description	27
4.9 ATMEGA32 interrupts definitions	27
4.9.1 Detailed Description	28
4.9.2 Macro Definition Documentation	28
4.9.2.1 ADC	28
4.9.2.2 ANA_COMP	29
4.9.2.3 cli	29
4.9.2.4 EE_RDY	29
4.9.2.5 EXT_INT0	29
4.9.2.6 EXT_INT1	30

4.9.2.7 EXT_INT2	30
4.9.2.8 ISR	30
4.9.2.9 sei	30
4.9.2.10 SPI_STC	31
4.9.2.11 SPM_RDY	31
4.9.2.12 TIM0_COMP	31
4.9.2.13 TIM0_OVF	31
4.9.2.14 TIM1_CAPT	31
4.9.2.15 TIM1_COMPA	32
4.9.2.16 TIM1_COMPB	32
4.9.2.17 TIM1_OVF	32
4.9.2.18 TIM2_COMP	32
4.9.2.19 TIM2_OVF	32
4.9.2.20 TWI	33
4.9.2.21 USART_RXC	33
4.9.2.22 USART_TXC	33
4.9.2.23 USART_UDRE	33
4.10 Timers driver	33
4.10.1 Detailed Description	34
4.11 Timer0 driver	34
4.11.1 Detailed Description	36
4.11.2 Macro Definition Documentation	36
4.11.2.1 CLR_TIMER0_CLK_SRC	36
4.11.2.2 CLR_TIMER0_MODE	36
4.11.2.3 CS00	36
4.11.2.4 CS01	37
4.11.2.5 CS02	37
4.11.2.6 OCF0	37
4.11.2.7 OCIE0	37
4.11.2.8 TIMER0_NUM_OF_TICKS	37
4.11.2.9 TOIE0	38
4.11.2.10 TOV0	38
4.11.2.11 WGM00	38
4.11.2.12 WGM01	38
4.11.3 Enumeration Type Documentation	38
4.11.3.1 EN_Timer0_clkSource_t	38
4.11.3.2 En_Timer0_Error_t	39
4.11.3.3 EN_Timer0_Mode_t	39
4.11.3.4 TIMER0_interrupt_t	40
4.11.4 Function Documentation	41
4.11.4.1 Timer0_delay_ms()	41
4.11.4.2 Timer0_init()	41

4.11.4.3 Timer0_interruptDiable()	42
4.11.4.4 Timer0_interruptEnable()	42
4.11.4.5 Timer0_reset()	43
4.11.4.6 Timer0_start()	43
4.11.4.7 Timer0_stop()	44
4.12 MCU ports	44
4.12.1 Detailed Description	45
4.12.2 Macro Definition Documentation	45
4.12.2.1 PORTA_OFFSET	45
4.12.2.2 PORTB_OFFSET	45
4.12.2.3 PORTC_OFFSET	45
4.12.2.4 PORTD_OFFSET	46
4.12.3 Enumeration Type Documentation	46
4.12.3.1 EN_pinDirection_t	46
4.12.3.2 EN_pinErro_t	46
4.12.3.3 EN_pinNum_t	46
4.12.3.4 EN_pinState_t	47
4.13 Bit math	48
4.13.1 Detailed Description	48
4.13.2 Macro Definition Documentation	48
4.13.2.1 clrBit	48
4.13.2.2 getBit	49
4.13.2.3 setBit	49
4.13.2.4 toggleBit	50
4.14 Definition of data types	50
4.14.1 Detailed Description	50
4.14.2 Typedef Documentation	51
4.14.2.1 float128_t	51
4.14.2.2 float32_t	51
4.14.2.3 float64_t	51
4.14.2.4 sint16_t	51
4.14.2.5 sint32_t	51
4.14.2.6 sint8_t	52
4.14.2.7 uint16_t	52
4.14.2.8 uint32_t	52
4.14.2.9 uint8_t	52
4.15 Service layer	52
4.15.1 Detailed Description	53
4.16 MCU Registers	53
4.16.1 Detailed Description	53
4.17 I/O registers	53
4.17.1 Detailed Description	53

4.18 Port A registers	54
4.18.1 Detailed Description	54
4.18.2 Macro Definition Documentation	54
4.18.2.1 DDRA	54
4.18.2.2 PINA	54
4.18.2.3 PORTA	55
4.19 Port B registers	55
4.19.1 Detailed Description	55
4.19.2 Macro Definition Documentation	55
4.19.2.1 DDRB	55
4.19.2.2 PINB	56
4.19.2.3 PORTB	56
4.20 Port C registers	56
4.20.1 Detailed Description	56
4.20.2 Macro Definition Documentation	56
4.20.2.1 DDRC	57
4.20.2.2 PINC	57
4.20.2.3 PORTC	57
4.21 Port D registers	58
4.21.1 Detailed Description	58
4.21.2 Macro Definition Documentation	58
4.21.2.1 DDRD	58
4.21.2.2 PIND	58
4.21.2.3 PORTD	59
4.22 Interrupt registers	59
4.22.1 Detailed Description	59
4.22.2 Macro Definition Documentation	59
4.22.2.1 GICR	59
4.22.2.2 GIFR	60
4.22.2.3 MCUCR	60
4.22.2.4 MCUCSR	61
4.23 Timers Registers	61
4.23.1 Detailed Description	61
4.24 Timer0 Registers	61
4.24.1 Detailed Description	61
4.24.2 Macro Definition Documentation	62
4.24.2.1 OCR0	62
4.24.2.2 TCCR0	62
4.24.2.3 TCNT0	64
4.25 General Timers registers	65
4.25.1 Detailed Description	65
4.25.2 Macro Definition Documentation	65

4.25.2.1 TIFR	65
4.25.2.2 TIMSK	66
5 File Documentation	67
5.1 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/App/app.c File Reference	67
5.1.1 Macro Definition Documentation	68
5.1.1.1 SYSTEM_CLK	68
5.1.2 Function Documentation	68
5.1.2.1 ISR()	68
5.1.3 Variable Documentation	68
5.1.3.1 system_CurrentState	68
5.2 app.c	69
5.3 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/App/app.h File Reference	71
5.4 app.h	73
5.5 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/Button driver/Button.c File Reference	73
5.6 Button.c	74
5.7 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/Button driver/Button.h File Reference	74
5.8 Button.h	74
5.9 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/LED driver/LED.c File Reference	75
5.10 LED.c	75
5.11 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/LED driver/LED.h File Reference	75
5.12 LED.h	76
5.13 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/IncFile1.h File Reference	76
5.14 IncFile1.h	76
5.15 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/main.c File Reference	77
5.15.1 Function Documentation	77
5.15.1.1 main()	77
5.16 main.c	77
5.17 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Dio driver/DIO.c File Reference	77
5.18 DIO.c	78
5.19 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Dio driver/DIO.h File Reference	81
5.19.1 Detailed Description	81
5.20 DIO.h	81
5.21 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Ext interrupt driver/Ext interrupt.c File Reference	82

5.22 Ext interrupt.c	82
5.23 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Ext interrupt driver/Ext interrupt.h File Reference	83
5.24 Ext interrupt.h	85
5.25 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Interrupt/Interrupt.h File Reference	85
5.26 Interrupt.h	86
5.27 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Timer driver/Timer_0.c File Reference	87
5.27.1 Function Documentation	87
5.27.1.1 ISR()	87
5.27.2 Variable Documentation	87
5.27.2.1 Timer0_globalClkSource	88
5.27.2.2 Timer0_globalNumOfOverFlows	88
5.27.2.3 Timer0_globalOverFlowTime	88
5.27.2.4 Timer0_globalTickTime	88
5.28 Timer_0.c	89
5.29 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Timer driver/Timer_0.h File Reference	90
5.30 Timer_0.h	92
5.31 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/ATmega32Port.h File Reference	94
5.32 ATmega32Port.h	94
5.33 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/BitMath.h File Reference	95
5.34 BitMath.h	95
5.35 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/dataTypes.h File Reference	96
5.36 dataType.h	96
5.37 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/RegisterFile.h File Reference	96
5.38 RegisterFile.h	97
Index	99

Chapter 1

System static architecture

1.1 system description

- This system consist of two modes normal mode and pedestrians mode.
- It controls two traffic lights one for cars and one for pedestrians.
- It has button if the button pressed the system will change from normal mode to pedestrians mode.

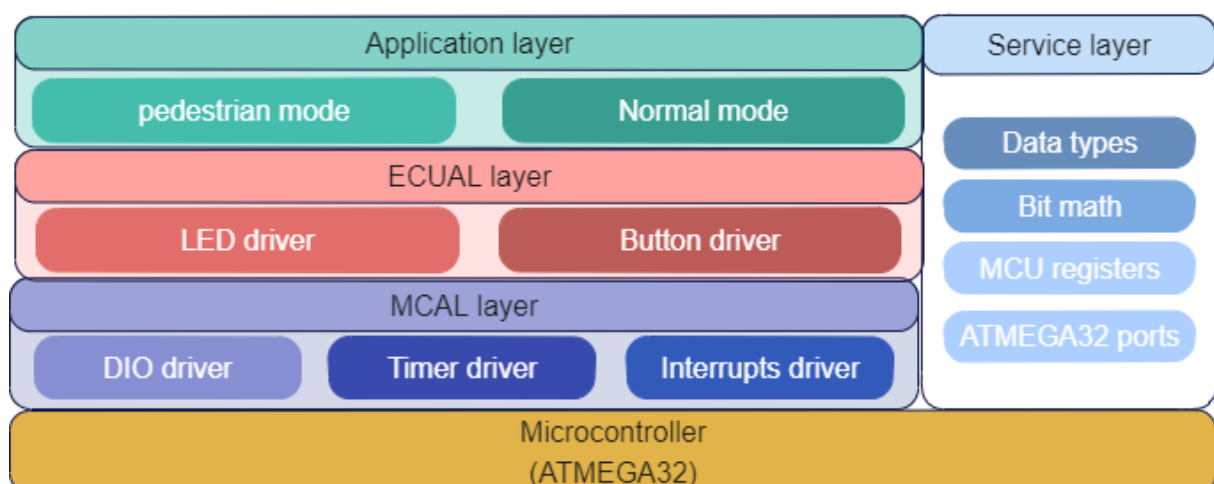
1.1.1 Hardware

- 6 LEDs (3 for cars and 3 for pedestrians).
- 1 push button.
- ATmega32 MCU.

1.1.2 Software & IDEs

- Proteus (used for system simulation).
- Microchip studio (used for development and debugging).

1.2 system design

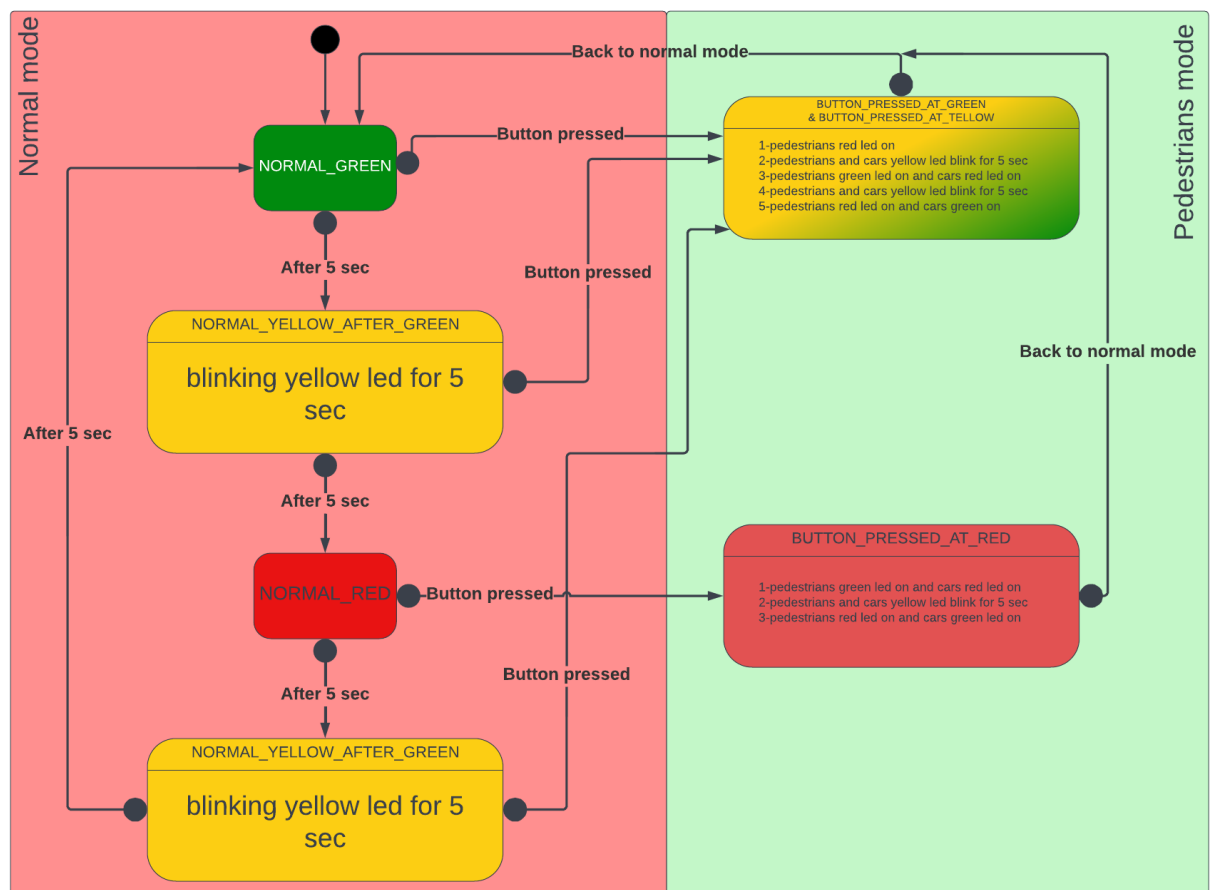


- This system consist of 4 layers (MCAL layer,ECUAL layer,Application layer,Service layer)

1.3 State machine

- This state machine describes the flow of the system
- It consist of 7 main states

NORMAL_GREEN
 NORMAL_YELLOW_AFTER_GREEN
 NORMAL_RED
 NORMAL_YELLOW_AFTER_RED
 BUTTON_PRESSED_AT_GREEN
 BUTTON_PRESSED_AT_YELLOW
 BUTTON_PRESSED_AT_RED



Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Application layer	7
ECUAL layer	14
Button driver	14
LED driver	16
MCAL layer	19
DIO driver	19
Interrupts driver	27
ATMEGA32 external interrupts driver	22
ATMEGA32 interrupts definitions	27
Timers driver	33
Timer0 driver	34
Service layer	52
MCU ports	44
Bit math	48
Definition of data types	50
MCU Registers	53
I/O registers	53
Port A registers	54
Port B registers	55
Port C registers	56
Port D registers	58
Interrupt registers	59
Timers Registers	61
Timer0 Registers	61
General Timers registers	65

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ IncFile1.h	76
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ main.c	77
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/App/ app.c	67
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/App/ app.h	71
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/Button driver/ Button.c	73
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/Button driver/ Button.h	74
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/LED driver/ LED.c	75
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/LED driver/ LED.h	75
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Dio driver/ DIO.c	77
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Dio driver/ DIO.h	81
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Ext interrupt driver/ Ext interrupt.c	82
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Ext interrupt driver/ Ext interrupt.h	83
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Interrupt/ Interrupt.h	85
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Timer driver/ Timer_0.c	87
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Timer driver/ Timer_0.h	90
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/ ATmega32Port.h	94
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/ BitMath.h	95
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/ dataTypes.h	96
E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/ RegisterFile.h	96

Chapter 4

Module Documentation

4.1 Application layer

Enumerations

- enum [EN_TrafficStates_t](#) {
 [NORMAL_GREEN](#) , [NORMAL_YELLOW_AFTER_GREEN](#) , [NORMAL_RED](#) , [NORMAL_YELLOW_AFTER_RED](#)
 ,
 [BUTTON_PRESSED_AT_GREEN](#) , [BUTTON_PRESSED_AT_RED](#) , [BUTTON_PRESSED_AT_YELLOW](#) }

This enum contains the values for each state of the system.

Functions

- void [app_start](#) (void)
app_start this function used to initialize the MCU for the application
- void [trafficLightOperation](#) (void)

Cars led control function

These function controls the leds of the cars. \It is used in normal mode states.

Parameters

in	<i>None</i>	this function doesn't take any input parameters.
out	<i>None</i>	this function doesn't take any output parameters.

Return values

<i>None</i>	This function doesn't return anything.
-------------	--

- void [carsGreenState](#) (void)
carsGreenState

- void `carsYellowState` (void)
carsYellowState
- void `carsRedState` (void)
carsRedState

Pedestrians led control function

These function controls the leds of the pedestrians. \t is used in pedestrians mode states.

Parameters

in	<i>None</i>	this function doesn't take any input parameters.
out	<i>None</i>	this function doesn't take any output parameters.

Return values

<i>None</i>	This function doesn't return anything.
-------------	--

- void `pedestriansGreenState` (void)
pedestriansGreenState
- void `pedestriansYellowState` (void)
pedestriansYellowState
- void `pedestriansRedState` (void)
pedestriansRedState

Cars LEDS Macros

These macros represent the pins which cars leds connected to.

- `#define CARS_GREEN_LED PA0`
- `#define CARS_YELLOW_LED PA1`
- `#define CARS_RED_LED PA2`

Pedestrians LEDS Macros

These macros represent the pins which pedestrians leds connected to.

- `#define PEDESTRIANS_GREEN_LED PB0`
- `#define PEDESTRIANS_YELLOW_LED PB1`
- `#define PEDESTRIANS_RED_LED PB2`

4.1.1 Detailed Description

- This layer contains all the function to control the flow of the system.

4.1.2 Macro Definition Documentation

4.1.2.1 CARS_GREEN_LED

```
#define CARS_GREEN_LED PA0
```

This is a cars green led pin

Definition at line 23 of file [app.h](#).

4.1.2.2 CARS_RED_LED

```
#define CARS_RED_LED PA2
```

This is a cars red led pin

Definition at line 25 of file [app.h](#).

4.1.2.3 CARS_YELLOW_LED

```
#define CARS_YELLOW_LED PA1
```

This is a cars yellow led pin

Definition at line 24 of file [app.h](#).

4.1.2.4 PEDESTRIANS_GREEN_LED

```
#define PEDESTRIANS_GREEN_LED PB0
```

This is a pedestrians green led pin

Definition at line 34 of file [app.h](#).

4.1.2.5 PEDESTRIANS_RED_LED

```
#define PEDESTRIANS_RED_LED PB2
```

This is a pedestrians red led pin

Definition at line 36 of file [app.h](#).

4.1.2.6 PEDESTRIANS_YELLOW_LED

```
#define PEDESTRIANS_YELLOW_LED PB1
```

This is a pedestrians yellow led pin

Definition at line 35 of file [app.h](#).

4.1.3 Enumeration Type Documentation

4.1.3.1 EN_TrafficStates_t

```
enum EN_TrafficStates_t
```

This enum contains the values for each state of the system.

Enumerator

NORMAL_GREEN	This enum value is for normal green state
NORMAL_YELLOW_AFTER_GREEN	This enum value is for normal yellow state after green finished
NORMAL_RED	This enum value is for normal state
NORMAL_YELLOW_AFTER_RED	This enum value is for normal yellow state after red finished
BUTTON_PRESSED_AT_GREEN	This enum value is for the button if it pressed when the system state is green
BUTTON_PRESSED_AT_RED	This enum value is for the button if it pressed when the system state is red
BUTTON_PRESSED_AT_YELLOW	This enum value is for the button if it pressed when the system state is yellow

Definition at line 42 of file [app.h](#).

4.1.4 Function Documentation

4.1.4.1 app_start()

```
void app_start (
    void )
```

`app_start` this function used to initialize the MCU for the application

- 1.It initialize cars and pedestrians leds as output.
- 2.It initialize external interrupt 0 ([INT0](#)) to trigger at falling edge.
- 3.It initialize Timer 0 as normal mode with prescaler 1024.

Parameters

in	<i>None</i>	this function doesn't take any input parameters.
out	<i>None</i>	this function doesn't take any output parameters.

Return values

<i>None</i>	This function doesn't return anything.
-------------	--

Definition at line 21 of file [app.c](#).

4.1.4.2 carsGreenState()

```
void carsGreenState (  
    void )
```

carsGreenState

This function responsible for making the cars green led on for 5 sec.

Definition at line 41 of file [app.c](#).

4.1.4.3 carsRedState()

```
void carsRedState (  
    void )
```

carsRedState

This function responsible for making the cars red led on for 5 sec.

Parameters

in	<i>None</i>	this function doesn't take any input parameters.
out	<i>None</i>	this function doesn't take any output parameters.

Return values

<i>None</i>	This function doesn't return anything.
-------------	--

Definition at line 64 of file [app.c](#).

4.1.4.4 carsYellowState()

```
void carsYellowState (  
    void )
```

carsYellowState

This function responsible for making the cars yellow led to blink for 5 sec.

Parameters

in	<i>None</i>	this function doesn't take any input parameters.
out	<i>None</i>	this function doesn't take any output parameters.

Return values

<i>None</i>	This function doesn't return anything.
-------------	--

Definition at line [48](#) of file [app.c](#).

4.1.4.5 pedestriansGreenState()

```
void pedestriansGreenState (  
    void )
```

pedestriansGreenState

- This function responsible for making the cars red led on and pedestrians green led on for 5 sec.
- This function responsible for making the cars red led off and pedestrians red led off for 5 sec.

Parameters

in	<i>None</i>	this function doesn't take any input parameters.
out	<i>None</i>	this function doesn't take any output parameters.

Return values

<i>None</i>	This function doesn't return anything.
-------------	--

Definition at line [148](#) of file [app.c](#).

4.1.4.6 pedestriansRedState()

```
void pedestriansRedState (  
    void )
```

pedestriansRedState

- This function responsible for making the pedestrians red led on and cars green led on for 5 sec.
- This function responsible for making the cars red led off and pedestrians green led off for 5 sec.

Parameters

in	<i>None</i>	this function doesn't take any input parameters.
out	<i>None</i>	this function doesn't take any output parameters.

Return values

<i>None</i>	This function doesn't return anything.
-------------	--

Definition at line 172 of file [app.c](#).

4.1.4.7 pedestriansYellowState()

```
void pedestriansYellowState (  
    void )
```

pedestriansYellowState

- This function responsible for making the cars yellow and pedestrians yellow led blink for 5 sec.

Parameters

in	<i>None</i>	this function doesn't take any input parameters.
out	<i>None</i>	this function doesn't take any output parameters.

Return values

<i>None</i>	This function doesn't return anything.
-------------	--

Definition at line 157 of file [app.c](#).

4.1.4.8 trafficLightOperation()

```
void trafficLightOperation (  
    void )
```

@brief trafficLightOperation

- This function used as a main API for the system.
- It is used to switch between the system states and decide which state will run.

Parameters

in	<i>None</i>	this function doesn't take any input parameters.
out	<i>None</i>	this function doesn't take any output parameters.

Return values

<i>None</i>	This function doesn't return anything.
-------------	--

Definition at line 71 of file [app.c](#).

4.2 ECUAL layer

Modules

- [Button driver](#)
- [LED driver](#)

4.2.1 Detailed Description

This layer contains all the drivers for the external devices that connected to the MCU.

4.3 Button driver

Functions

- [EN_pinErro_t buttonInit](#) ([EN_pinNum_t](#) buttonPin)
initialize the button pin.
- [EN_pinErro_t buttonRead](#) ([EN_pinNum_t](#) buttonPin, [EN_pinState_t](#) *pinState)
reads the value of the button.

4.3.1 Detailed Description

This driver contains all the function that controls the buttons connected to the MCU.

4.3.2 Function Documentation

4.3.2.1 buttonInit()

```
EN_pinErro_t buttonInit (  
    EN_pinNum_t buttonPin )
```

initialize the button pin.

`buttonInit` function:

- This function makes the button pin as Input.

Parameters

in	<i>buttonPin</i>	it is the pin which the button is connected to,it may be (PA0 to PD7).
out	<i>none</i>	no output arguments

Return values

<i>WRONG_PIN_NUM</i>	if the pinNum is wrong.
<i>OK</i>	if the pinNum is correct.

Definition at line 11 of file [Button.c](#).

4.3.2.2 buttonRead()

```
EN_pinErro_t buttonRead (
    EN_pinNum_t buttonPin,
    EN_pinState_t * pinState )
```

reads the value of the button.

[buttonRead](#) function:

- It reads the value of the connected pin to the button.
- It store the value in the pinState pointer.

Parameters

in	<i>buttonPin</i>	it is the pin which the button is connected to,it may be (PA0 to PD7).
out	<i>pinState</i>	the function store the value of the button in that pointer.

Return values

<i>WRONG_PIN_NUM</i>	if the pinNum is wrong.
<i>OK</i>	if the pinNum is correct.

Definition at line 16 of file [Button.c](#).

4.4 LED driver**Functions**

- [EN_pinErro_t ledInit](#) ([EN_pinNum_t](#) ledPin)

- initialize the led pin.*
- `EN_pinErro_t ledOn (EN_pinNum_t ledPin)`
turn the led on.
- `EN_pinErro_t ledOff (EN_pinNum_t ledPin)`
turn the led off.
- `EN_pinNum_t ledToggle (EN_pinNum_t ledPin)`
toggle the led state.

4.4.1 Detailed Description

This driver contains all the function that controls the LEDs connected to the MCU.

4.4.2 Function Documentation

4.4.2.1 ledInit()

```
EN_pinErro_t ledInit (
    EN_pinNum_t ledPin )
```

initialize the led pin.

`ledInit` function:

- This function initialize the led pin as output.

Parameters

in	<i>ledPin</i>	it is the pin which the led is connected to,it may be (PA0 to PD7).
out	<i>none</i>	no output arguments

Return values

<i>WRONG_PIN_NUM</i>	if the pinNum is wrong.
<i>OK</i>	if the pinNum is correct.

Definition at line 10 of file [LED.c](#).

4.4.2.2 ledOff()

```
EN_pinErro_t ledOff (
    EN_pinNum_t ledPin )
```

turn the led off.

ledOff function:

- This function turns the led off by writing low to the pin.

Parameters

in	<i>ledPin</i>	it is the pin which the led is connected to,it may be (PA0 to PD7).
out	<i>none</i>	no output arguments

Return values

<i>WRONG_PIN_NUM</i>	if the pinNum is wrong.
<i>OK</i>	if the pinNum is correct.

Definition at line 20 of file [LED.c](#).

4.4.2.3 ledOn()

```
EN_pinErro_t ledOn (
    EN_pinNum_t ledPin )
```

turn the led on.

ledOn function:

- This function turns the led on by writing high to the pin.

Parameters

in	<i>ledPin</i>	it is the pin which the led is connected to,it may be (PA0 to PD7).
out	<i>none</i>	no output arguments

Return values

<i>WRONG_PIN_NUM</i>	if the pinNum is wrong.
<i>OK</i>	if the pinNum is correct.

Definition at line 15 of file [LED.c](#).

4.4.2.4 ledToggle()

```
EN_pinNum_t ledToggle (
```

```
EN_pinNum_t ledPin )
```

toggle the led state.

[ledToggle](#) function:

- This function toggle the led state.
- It makes the led on if the led was off.
- It makes the led off if the led was on.

Parameters

in	<i>ledPin</i>	it is the pin which the led is connected to,it may be (PA0 to PD7).
out	<i>none</i>	no output arguments

Return values

<i>WRONG_PIN_NUM</i>	if the pinNum is wrong.
<i>OK</i>	if the pinNum is correct.

Definition at line 25 of file [LED.c](#).

4.5 MCAL layer

Modules

- [DIO driver](#)
- [Interrupts driver](#)
- [Timers driver](#)

4.5.1 Detailed Description

This layer contains all the driver related to the MCU.

4.6 DIO driver

Functions

- [EN_pinErro_t DIO_pinInit](#) ([EN_pinNum_t](#) pinNum, [EN_pinDirection_t](#) pinDirection)
Set the direction of the pin.
- [EN_pinErro_t DIO_pinWrite](#) ([EN_pinNum_t](#) pinNum, [EN_pinState_t](#) pinState)
This function writes High or Low on the pin.
- [EN_pinErro_t DIO_pinToggle](#) ([EN_pinNum_t](#) pinNum)
This function toggles the state of the pin.
- [EN_pinErro_t DIO_pinRead](#) ([EN_pinNum_t](#) pinNum, [EN_pinState_t](#) *pinState)
This function reads the state of the pin.

4.6.1 Detailed Description

This contains all the function needed to configure and manipulate the MCU ports.

4.6.2 Function Documentation

4.6.2.1 DIO_pinInit()

```
EN_pinErro_t DIO_pinInit (
    EN_pinNum_t pinNum,
    EN_pinDirection_t pinDirection )
```

Set the direction of the pin.

DIO_pinInit

- This function makes pin input or output.
- it makes the pinNum Output by setting the pinNum in the DDRx (x:A,B,C or D) register.
- it makes the pinNum Input by clearing the pinNum in the DDRx (x:A,B,C or D) register.

Parameters

in	<i>pinNum</i>	it represent the pin number (PA0 to PD7).
in	<i>pinDirection</i>	it represent the pin direction it may be (Input or Output).
out	<i>none</i>	no output arguments

Return values

<i>WRONG_PIN_NUM</i>	if the pinNum is wrong.
<i>WRONG_PIN_DIR</i>	if the pinDirection is wrong.
<i>OK</i>	if the pinNum and the pinDirection are correct.

Definition at line 12 of file [DIO.c](#).

4.6.2.2 DIO_pinRead()

```
EN_pinErro_t DIO_pinRead (
    EN_pinNum_t pinNum,
    EN_pinState_t * pinState )
```

This function reads the state of the pin.

DIO_pinRead

- It reads the bit relative to the pinNum in the register PINx (A,B,C or D).

Parameters

in	<i>pinNum</i>	it represent the pin number (PA0 to PD7).
out	<i>pinState</i>	this is a pointer to store the state of the pin (High or Low).

Return values

<i>WRONG_PIN_NUM</i>	if the pinNum is wrong.
<i>OK</i>	if the pinNum is correct.

Definition at line 166 of file [DIO.c](#).

4.6.2.3 DIO_pinToggle()

```
EN_pinErro_t DIO_pinToggle (  
    EN_pinNum_t pinNum )
```

This function toggles the state of the pin.

[DIO_pinToggle](#)

- if the current state of the pin is High it will make it Low.
- if the current state of the pin is Low it will make it High.

Parameters

in	<i>pinNum</i>	it represent the pin number (PA0 to PD7).
out	<i>none</i>	no output arguments

Return values

<i>WRONG_PIN_NUM</i>	if the pinNum is wrong.
<i>OK</i>	if the pinNum is correct.

Definition at line 198 of file [DIO.c](#).

4.6.2.4 DIO_pinWrite()

```
EN_pinErro_t DIO_pinWrite (  
    EN_pinNum_t pinNum,  
    EN_pinState_t pinState )
```

This function writes High or Low on the pin.

DIO_pinWrite

- it writes High to the pinNum by setting the pinNum in the PORTx (x:A,B,C or D) register.
- it writes Low to the pinNum by clearing the pinNum in the PORTx (x:A,B,C or D) register.

Parameters

in	<i>pinNum</i>	it represent the pin number (PA0 to PD7).
in	<i>pinState</i>	it represent the pin state it may be (High or Low).
out	<i>none</i>	no output arguments

Return values

<i>WRONG_PIN_NUM</i>	if the pinNum is wrong.
<i>WRONG_PIN_STATE</i>	if the pinState is wrong.
<i>OK</i>	if the pinNum and the pinState are correct.

Definition at line 90 of file [DIO.c](#).

4.7 ATMEGA32 external interrupts driver

External interrupts driver.

Enumerations

- enum [EN_interruptNum_t](#) { [INT2](#) = 5 , [INT0](#) , [INT1](#) }
External interrupt number.
- enum [EN_interruptSenseControl_t](#) { [LOW_LEVEL](#) , [ANY_LOGICAL_CHANGE](#) , [FALLING_EDGE](#) , [RISING_EDGE](#) }
External interrupt sense control.
- enum [EN_interruptError_t](#) { [INT_OK](#) , [WRONG_INT_NUM](#) , [WRONG_SENSE_CONTROL](#) }
External interrupt errors.

Functions

- [EN_interruptError_t](#) [Ext_interruptInit](#) ([EN_interruptNum_t](#) interruptNum, [EN_interruptSenseControl_t](#) interruptSenseControl)
External interrupt init.

External interrupts pins

- These are the pins which connected to each interrupt.
- It should be configured as [Input](#).
- #define [INT0_PIN](#) ([PD2](#) - [PORTD_OFFSET](#))
- #define [INT1_PIN](#) ([PD3](#) - [PORTD_OFFSET](#))
- #define [INT2_PIN](#) ([PB2](#) - [PORTB_OFFSET](#))

INT0 sense control

- These two bits `ISC00` and `ISC01` which located in `MCUCR` register control the `INT0` sense control.

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

- `#define ISC00 0`
- `#define ISC01 1`

INT1 sense control

- These two bits `ISC10` and `ISC11` which located in `MCUCR` register control the `INT1` sense control.

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

- `#define ISC10 2`
- `#define ISC11 3`

INT2 sense control

- This bit `ISC2` which located in `MCUCSR` register control the `INT2` sense control.

ISC2	Description
0	The falling edge on INT2 activates the interrupt request.
1	The rising edge on INT2 activates the interrupt request.

- `#define ISC2 6`

4.7.1 Detailed Description

External interrupts driver.

4.7.2 Macro Definition Documentation

4.7.2.1 INT0_PIN

```
#define INT0_PIN (PD2 - PORTD_OFFSET)
```

This Pin connected to INT0 interrupt

Definition at line 29 of file [Ext interrupt.h](#).

4.7.2.2 INT1_PIN

```
#define INT1_PIN (PD3 - PORTD_OFFSET)
```

This Pin connected to INT1 interrupt

Definition at line 30 of file [Ext interrupt.h](#).

4.7.2.3 INT2_PIN

```
#define INT2_PIN (PB2 - PORTB_OFFSET)
```

This Pin connected to INT2 interrupt

Definition at line 31 of file [Ext interrupt.h](#).

4.7.2.4 ISC00

```
#define ISC00 0
```

Interrupt Sense Control 0 Bit 0

Definition at line 46 of file [Ext interrupt.h](#).

4.7.2.5 ISC01

```
#define ISC01 1
```

Interrupt Sense Control 0 Bit 1

Definition at line 47 of file [Ext interrupt.h](#).

4.7.2.6 ISC10

```
#define ISC10 2
```

Interrupt Sense Control 1 Bit 0

Definition at line 63 of file [Ext interrupt.h](#).

4.7.2.7 ISC11

```
#define ISC11 3
```

Interrupt Sense Control 1 Bit 1

Definition at line 64 of file [Ext interrupt.h](#).

4.7.2.8 ISC2

```
#define ISC2 6
```

Interrupt Sense Control 2 Bit 6

Definition at line 78 of file [Ext interrupt.h](#).

4.7.3 Enumeration Type Documentation

4.7.3.1 EN_interruptError_t

```
enum EN\_interruptError\_t
```

External interrupt errors.

- This enum contains the values for interrupt errors.

Enumerator

INT_OK	enum value shows that INTx parameters is right.
WRONG_INT_NUM	enum value shows that INTx number is wrong.
WRONG_SENSE_CONTROL	enum value shows that INTx sense control is wrong.

Definition at line 117 of file [Ext interrupt.h](#).

4.7.3.2 EN_interruptNum_t

enum [EN_interruptNum_t](#)

External interrupt number.

- This enum contains the bit number for each interrupt in [GICR](#) register.
- Setting these bits will enables the interrupts.
- Clearing these bits will disables the interrupts.

Enumerator

INT2	enum value for external interrupt 2
INT0	enum value for external interrupt 0
INT1	enum value for external interrupt 1

Definition at line 89 of file [Ext interrupt.h](#).

4.7.3.3 EN_interruptSenseControl_t

enum [EN_interruptSenseControl_t](#)

External interrupt sense control.

- This enum contains the values for interrupt sense control.
- each value represent the exact value that should be written in the [MCUCR](#) register this for [INT0](#) and [INT1](#) and [MCUCSR](#) register for [INT2](#).

Note

- [INT2](#) has just rising and falling edge sense control.

Enumerator

LOW_LEVEL	The low level generates an interrupt request.
ANY_LOGICAL_CHANGE	Any logical change generates an interrupt request.
FALLING_EDGE	The falling edge generates an interrupt request
RISING_EDGE	The rising edge generates an interrupt request

Definition at line 104 of file [Ext interrupt.h](#).

4.7.4 Function Documentation

4.7.4.1 Ext_interruptInit()

```
EN_interruptError_t Ext_interruptInit (
    EN_interruptNum_t interruptNum,
    EN_interruptSenseControl_t interruptSenseControl )
```

External interrupt init.

- This function configures INTx sense control.
- This function enables INTx.

Parameters

in	<i>interruptNum</i>	This is the interrupt number that needed to be enabled.
in	<i>interruptSenseControl</i>	This is the value of the interrupt sense control which the interrupt will activated at it.

Return values

<i>INT_OK</i>	If interruptNum and interruptSenseControl are corrects.
<i>WRONG_INT_NUM</i>	If interruptNum is wrong.
<i>WRONG_SENSE_CONTROL</i>	If interruptSenseControl is wrong.

Definition at line 9 of file [Ext interrupt.c](#).

4.8 Interrupts driver

Modules

- [ATMEGA32 external interrupts driver](#)
External interrupts driver.
- [ATMEGA32 interrupts definitions](#)
Interrupts request handlers.

4.8.1 Detailed Description

4.9 ATMEGA32 interrupts definitions

Interrupts request handlers.

Macros

- `#define sei() __asm__ __volatile__ ("sei" ::: "memory")`
- `#define cli() __asm__ __volatile__ ("cli" ::: "memory")`
- `#define EXT_INT0 __vector_1`
- `#define EXT_INT1 __vector_2`
- `#define EXT_INT2 __vector_3`
- `#define TIM2_COMP __vector_4`
- `#define TIM2_OVF __vector_5`
- `#define TIM1_CAPT __vector_6`
- `#define TIM1_COMPA __vector_7`
- `#define TIM1_COMPB __vector_8`
- `#define TIM1_OVF __vector_9`
- `#define TIM0_COMP __vector_10`
- `#define TIM0_OVF __vector_11`
- `#define SPI_STC __vector_12`
- `#define USART_RXC __vector_13`
- `#define USART_UDRE __vector_14`
- `#define USART_TXC __vector_15`
- `#define ADC __vector_16`
- `#define EE_RDY __vector_17`
- `#define ANA_COMP __vector_18`
- `#define TWI __vector_19`
- `#define SPM_RDY __vector_20`
- `#define ISR(INT_VECT)`

interrupt service routine Macro.

4.9.1 Detailed Description

Interrupts request handlers.

This section contains:

- Macros for Interrupts request handlers in ATmega32.
- Macros for enabling and disabling global interrupt.
- ISR Macro which defines interrupt service routine function.

4.9.2 Macro Definition Documentation

4.9.2.1 ADC

```
#define ADC __vector_16
```

This Macro defines ADC Conversion Complete Handler

Definition at line 63 of file [Interrupt.h](#).

4.9.2.2 ANA_COMP

```
#define ANA_COMP __vector_18
```

This Macro defines Analog Comparator Handler

Definition at line 65 of file [Interrupt.h](#).

4.9.2.3 cli

```
#define cli( ) __asm__ __volatile__ ("cli" ::: "memory")
```

- Disables all interrupts by clearing the global interrupt mask.
- This function actually compiles into a single line of assembly, so there is no function call overhead.
- However, the macro also implies a **memory barrier** which can cause additional loss of optimization.

Definition at line 46 of file [Interrupt.h](#).

4.9.2.4 EE_RDY

```
#define EE_RDY __vector_17
```

This Macro defines EEPROM Ready Handler

Definition at line 64 of file [Interrupt.h](#).

4.9.2.5 EXT_INT0

```
#define EXT_INT0 __vector_1
```

This Macro defines IRQ0 Handler

Definition at line 48 of file [Interrupt.h](#).

4.9.2.6 EXT_INT1

```
#define EXT_INT1 __vector_2
```

This Macro defines IRQ1 Handler

Definition at line 49 of file [Interrupt.h](#).

4.9.2.7 EXT_INT2

```
#define EXT_INT2 __vector_3
```

This Macro defines IRQ2 Handler

Definition at line 50 of file [Interrupt.h](#).

4.9.2.8 ISR

```
#define ISR(  
    INT_VECT )
```

Value:

```
void INT_VECT(void) __attribute__((signal,used));\  
void INT_VECT(void)
```

interrupt service routine Macro.

- Introduces an interrupt handler function (interrupt service routine) that runs with global interrupts initially disabled by default with no attributes specified.

Precondition

`vector` must be one of the interrupt vector names that are valid for the particular MCU type.

Definition at line 78 of file [Interrupt.h](#).

4.9.2.9 sei

```
#define sei( ) __asm__ __volatile__ ("sei" ::: "memory")
```

- Disables all interrupts by clearing the global interrupt mask.
- This function actually compiles into a single line of assembly, so there is no function call overhead.
- However, the macro also implies a **memory barrier** which can cause additional loss of optimization.

Definition at line 35 of file [Interrupt.h](#).

4.9.2.10 SPI_STC

```
#define SPI_STC __vector_12
```

This Macro defines SPI Transfer Complete Handler

Definition at line 59 of file [Interrupt.h](#).

4.9.2.11 SPM_RDY

```
#define SPM_RDY __vector_20
```

This Macro defines Store Program Memory Ready Handler

Definition at line 67 of file [Interrupt.h](#).

4.9.2.12 TIM0_COMP

```
#define TIM0_COMP __vector_10
```

This Macro defines Timer0 Compare Handler

Definition at line 57 of file [Interrupt.h](#).

4.9.2.13 TIM0_OVF

```
#define TIM0_OVF __vector_11
```

This Macro defines Timer0 Overflow Handler

Definition at line 58 of file [Interrupt.h](#).

4.9.2.14 TIM1_CAPT

```
#define TIM1_CAPT __vector_6
```

This Macro defines Timer1 Capture Handler

Definition at line 53 of file [Interrupt.h](#).

4.9.2.15 TIM1_COMPA

```
#define TIM1_COMPA __vector_7
```

This Macro defines Timer1 CompareA Handler

Definition at line 54 of file [Interrupt.h](#).

4.9.2.16 TIM1_COMPB

```
#define TIM1_COMPB __vector_8
```

This Macro defines Timer1 CompareB Handler

Definition at line 55 of file [Interrupt.h](#).

4.9.2.17 TIM1_OVF

```
#define TIM1_OVF __vector_9
```

This Macro defines Timer1 Overflow Handler

Definition at line 56 of file [Interrupt.h](#).

4.9.2.18 TIM2_COMP

```
#define TIM2_COMP __vector_4
```

This Macro defines Timer2 Compare Handler

Definition at line 51 of file [Interrupt.h](#).

4.9.2.19 TIM2_OVF

```
#define TIM2_OVF __vector_5
```

This Macro defines Timer2 Overflow Handler

Definition at line 52 of file [Interrupt.h](#).

4.9.2.20 TWI

```
#define TWI __vector_19
```

This Macro defines Two-wire Serial Interface Handler

Definition at line 66 of file [Interrupt.h](#).

4.9.2.21 USART_RXC

```
#define USART_RXC __vector_13
```

This Macro defines USART RX Complete Handler

Definition at line 60 of file [Interrupt.h](#).

4.9.2.22 USART_TXC

```
#define USART_TXC __vector_15
```

This Macro defines USART TX Complete Handler

Definition at line 62 of file [Interrupt.h](#).

4.9.2.23 USART_UDRE

```
#define USART_UDRE __vector_14
```

This Macro defines UDR Empty Handler

Definition at line 61 of file [Interrupt.h](#).

4.10 Timers driver

Modules

- [Timer0 driver](#)

4.10.1 Detailed Description

This contains the drivers for Atmega32 Timers

4.11 Timer0 driver

Macros

- `#define TIMER0_NUM_OF_TICKS 256`
System clock Macro.
- `#define CLR_TIMER0_CLK_SRC 0xF8`
- `#define CLR_TIMER0_MODE 0xB7`

Enumerations

- enum `EN_Timer0_Mode_t` { `NORMAL = 0` , `PWM_PHASE_CORRECR = 8` , `CTC = 64` , `FAST_PWM = 72` }
- enum `EN_Timer0_clkSource_t` {
 `NO_CLOCK_SOURCE` , `clk_No_DIVISON` , `clk_DIVISION_BY_8` , `clk_DIVISION_BY_64` ,
 `clk_DIVISION_BY_256` , `clk_DIVISION_BY_1024` , `EXTERNAL_CLOCK_FALLING_EDGE` , `EXTERNAL_CLOCK_RISING_E`
}
- enum `En_Timer0_Error_t` { `TIMER0_OK` , `TIMER0_WRONG_MODE` , `TIMER0_WRONG_CLK_SOURCE` ,
 `TIMER0_WRONG_INT` }

Functions

- `En_Timer0_Error_t Timer0_interruptEnable (TIMER0_interrupt_t Timer0_interrupt)`
- `En_Timer0_Error_t Timer0_interruptDiable (TIMER0_interrupt_t Timer0_interrupt)`
- `En_Timer0_Error_t Timer0_init (EN_Timer0_Mode_t Timer0_mode, EN_Timer0_clkSource_t Timer0_clk↔
 Source)`
- `void Timer0_start (void)`
- `void Timer0_stop (void)`
- `void Timer0_reset (void)`
- `void Timer0_delay_ms (uint32_t delay_ms)`

Timer/Counter0 Interrupts Enable

- These bits enable and disable the interrupts of the counter and located in `TIMSK`.
- enum `TIMER0_interrupt_t` { `TIMER0_OVER_FLOW_INT` , `TIMER0_OUT_CMP_MATCH_INT` }
- `#define TOIE0 0`
- `#define OCIE0 1`

Bit 2:0 - CS02:0: Clock Select

- The three Clock Select bits select the clock source to be used by the Timer/Counter and located in `TCCR0`.

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clkI/O/(No prescaling).
0	1	0	clkI/O/8 (From prescaler).
0	1	1	clkI/O/64 (From prescaler).
1	0	0	clkI/O/256 (From prescaler).
1	0	1	clkI/O/1024 (From prescaler).
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.
Clock Select Bit Description			

Note

If external pin modes are used for the Timer/Counter0, transitions on the T0 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

- #define [CS00](#) 0
- #define [CS01](#) 1
- #define [CS02](#) 2

Bit 6, 3 - WGM01:0: Waveform Generation Mode

- These bits control the counting sequence of the counter and located in [TCCR0](#).
- the source for the maximum (TOP) counter value, and what type of Waveform Generation to be used.

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX
Waveform Generation Mode Bit Description						

- #define [WGM00](#) 3
- #define [WGM01](#) 6

Timer/Counter0 Interrupts Flags

- These bits are flags for interrupts of the Timer 0 and located in [TIFR](#).
- #define [TOV0](#) 0
- #define [OCF0](#) 1

4.11.1 Detailed Description

4.11.2 Macro Definition Documentation

4.11.2.1 CLR_TIMER0_CLK_SRC

```
#define CLR_TIMER0_CLK_SRC 0xF8
```

Clear Timer 0 clock source

- This macro used to clear Timer 0 clock source.
- Anding the register [TCCR0](#) by the [CLR_TIMER0_CLK_SRC](#) (0b1111 1000) will result clearing the three bits [CS00](#), [CS01](#) and [CS02](#).

Definition at line [80](#) of file [Timer_0.h](#).

4.11.2.2 CLR_TIMER0_MODE

```
#define CLR_TIMER0_MODE 0xB7
```

Clear Timer 0 Mode

- This macro used to clear Timer 0 mode.
- Anding the register [TCCR0](#) by the [CLR_TIMER0_MODE](#) (0b1011 0111) will result clearing the two bits [WGM00](#) and [WGM01](#).

Definition at line [107](#) of file [Timer_0.h](#).

4.11.2.3 CS00

```
#define CS00 0
```

Definition at line [68](#) of file [Timer_0.h](#).

4.11.2.4 CS01

```
#define CS01 1
```

Definition at line 69 of file [Timer_0.h](#).

4.11.2.5 CS02

```
#define CS02 2
```

Definition at line 70 of file [Timer_0.h](#).

4.11.2.6 OCF0

```
#define OCF0 1
```

Bit 1 - OCF0: Output Compare Flag

Definition at line 136 of file [Timer_0.h](#).

4.11.2.7 OCIE0

```
#define OCIE0 1
```

Bit 1 - OCIE0: Timer/Counter0 Output Compare Match Interrupt Enable

Definition at line 116 of file [Timer_0.h](#).

4.11.2.8 TIMER0_NUM_OF_TICKS

```
#define TIMER0_NUM_OF_TICKS 256
```

System clock Macro.

- This Macro is the system clock that the user used.
- It should provided in hertz like this

```
#define SYSTEM_CLK 1000000UL
```

.
- Default value is 1 MHz.

Number of Ticks.

- This Macro is the Number of Ticks for Timer 0.
- It the number of ticks for one over flow.
- It can be calculated by $2^{\text{num of bits}}$, Timer 0 is 8 bit timer so number of ticks for Timer 0 are $2^8 = 256$.

Definition at line 46 of file [Timer_0.h](#).

4.11.2.9 TOIE0

```
#define TOIE0 0
```

Bit 0 - TOIE0: Timer/Counter0 Overflow Interrupt Enable

Definition at line 115 of file [Timer_0.h](#).

4.11.2.10 TOV0

```
#define TOV0 0
```

Bit 0 - TOV0: Timer/Counter0 Overflow Flag

Definition at line 135 of file [Timer_0.h](#).

4.11.2.11 WGM00

```
#define WGM00 3
```

Definition at line 96 of file [Timer_0.h](#).

4.11.2.12 WGM01

```
#define WGM01 6
```

Definition at line 97 of file [Timer_0.h](#).

4.11.3 Enumeration Type Documentation

4.11.3.1 EN_Timer0_clkSource_t

```
enum EN_Timer0_clkSource_t
```

Timer 0 clock source

- This enum contains the values for Timer0 clock source that needed to be written in [TCCR0](#) register.
 - There is no need to check for each source, just (orring) | the [TCCR0](#) register with enum value shifted left by [CS00](#) will do the job.
 - After setting the clock source the timer will start automatically.
-
- ```
TCCR0 |= clkI_No_DIVISON«CS00; //this will make the timer clock sorce as the system clock (No prescaling)
```



## Enumerator

|                             |                                                            |
|-----------------------------|------------------------------------------------------------|
| NO_CLOCK_SOURCE             | No clock source<br>(Timer/Counter stopped).                |
| clk_No_DIVISON              | clk/O/(No prescaling).                                     |
| clk_DIVISION_BY_8           | clk/O/8 (From prescaler).                                  |
| clk_DIVISION_BY_64          | clk/O/64 (From prescaler).                                 |
| clk_DIVISION_BY_256         | clk/O/256 (From prescaler).                                |
| clk_DIVISION_BY_1024        | clk/O/1024 (From prescaler).                               |
| EXTERNAL_CLOCK_FALLING_EDGE | External clock source on T0 pin.<br>Clock on falling edge. |
| EXTERNAL_CLOCK_RISING_EDGE  | External clock source on T0 pin.<br>Clock on rising edge.  |

Definition at line 187 of file [Timer\\_0.h](#).

## 4.11.3.2 En\_Timer0\_Error\_t

```
enum En_Timer0_Error_t
```

## Timer 0 errors

- This enum contains the values for Timer 0 errors.

## Enumerator

|                         |                                                         |
|-------------------------|---------------------------------------------------------|
| TIMER0_OK               | enum value shows that timer 0 parameters are correct    |
| TIMER0_WRONG_MODE       | enum value shows that timer 0 mode is wrong             |
| TIMER0_WRONG_CLK_SOURCE | enum value shows that timer 0 clock source is wrong     |
| TIMER0_WRONG_INT        | enum value shows that timer 0 interrupt number is wrong |

Definition at line 204 of file [Timer\\_0.h](#).

## 4.11.3.3 EN\_Timer0\_Mode\_t

```
enum EN_Timer0_Mode_t
```

## Timer 0 Modes

- This enum contains the exact value for each mode that needed to be written in [TCCR0](#) register.
- There is no need to check for each mode, just (orring) | the [TCCR0](#) register with enum value will do the job.

### example

- `TCCR0 |= FAST_PWM; //this will make the timer work in fast PWM mode.`

### enum representation in [TCCR0](#) register:

| FOC0        | WGM00 | COM01 | COM00 | WGM01 | CS02 | CS01 | CS00 |
|-------------|-------|-------|-------|-------|------|------|------|
| x           | 0     | x     | x     | 0     | x    | x    | x    |
| Normal mode |       |       |       |       |      |      |      |

| FOC0                   | WGM00 | COM01 | COM00 | WGM01 | CS02 | CS01 | CS00 |
|------------------------|-------|-------|-------|-------|------|------|------|
| x                      | 0     | x     | x     | 1     | x    | x    | x    |
| PWM phase correct mode |       |       |       |       |      |      |      |

| FOC0                        | WGM00 | COM01 | COM00 | WGM01 | CS02 | CS01 | CS00 |
|-----------------------------|-------|-------|-------|-------|------|------|------|
| x                           | 1     | x     | x     | 0     | x    | x    | x    |
| clear timer on compare mode |       |       |       |       |      |      |      |

| FOC0          | WGM00 | COM01 | COM00 | WGM01 | CS02 | CS01 | CS00 |
|---------------|-------|-------|-------|-------|------|------|------|
| x             | 1     | x     | x     | 1     | x    | x    | x    |
| fast PWM mode |       |       |       |       |      |      |      |

### Enumerator

|                   |                                                    |
|-------------------|----------------------------------------------------|
| NORMAL            | enum value for Timer 0 Normal mode                 |
| PWM_PHASE_CORRECR | enum value for Timer 0 PWM phase correct mode      |
| CTC               | enum value for Timer 0 clear timer on compare mode |
| FAST_PWM          | enum value for Timer 0 fast PWM mode               |

Definition at line 170 of file [Timer\\_0.h](#).

#### 4.11.3.4 TIMERO\_interrupt\_t

enum [TIMERO\\_interrupt\\_t](#)

### Timer 0 interrupts choice

- This enum contains the values for Timer0 interrupts.

## Enumerator

|                          |                                               |
|--------------------------|-----------------------------------------------|
| TIMER0_OVER_FLOW_INT     | Timer/Counter0 Overflow Interrupt             |
| TIMER0_OUT_CMP_MATCH_INT | Timer/Counter0 Output Compare Match Interrupt |

Definition at line 122 of file [Timer\\_0.h](#).

## 4.11.4 Function Documentation

### 4.11.4.1 Timer0\_delay\_ms()

```
void Timer0_delay_ms (
 uint32_t delay_ms)
```

#### Timer 0 delay

- This function generates a delay in mile seconds using Timer 0.

## Parameters

|     |                 |                             |
|-----|-----------------|-----------------------------|
| in  | <i>delay_ms</i> | Delay time in mile seconds. |
| out | <i>void</i>     | No output arguments.        |

## Return values

|             |                                        |
|-------------|----------------------------------------|
| <i>void</i> | This function doesn't return anything. |
|-------------|----------------------------------------|

Definition at line 127 of file [Timer\\_0.c](#).

### 4.11.4.2 Timer0\_init()

```
En_Timer0_Error_t Timer0_init (
 EN_Timer0_Mode_t Timer0_mode,
 EN_Timer0_clkSource_t Timer0_clkSource)
```

#### Timer0 init

- This function initialize Timer 0.
- It configures Timer 0 mode.
- It configures Timer 0 clock source.

## Parameters

|    |                         |                                                                                              |
|----|-------------------------|----------------------------------------------------------------------------------------------|
| in | <i>Timer0_mode</i>      | The mode for Timer 0 it can be selected from <a href="#">EN_Timer0_Mode_t</a> .              |
| in | <i>Timer0_clkSource</i> | The clock source for Timer 0 it can be selected from <a href="#">EN_Timer0_clkSource_t</a> . |

## Return values

|                         |                                   |
|-------------------------|-----------------------------------|
| <i>TIMER0_OK</i>        | If timer 0 parameters are correct |
| <i>WRONG_MODE</i>       | If timer 0 mode is wrong          |
| <i>WRONG_CLK_SOURCE</i> | If timer 0 clock source is wrong  |

Definition at line 42 of file [Timer\\_0.c](#).

#### 4.11.4.3 Timer0\_interruptDiable()

```
En_Timer0_Error_t Timer0_interruptDiable (
 TIMER0_interrupt_t Timer0_interrupt)
```

##### Timer0 interrupt disable

- This function disables Timer 0 interrupt.

## Parameters

|     |                         |                           |
|-----|-------------------------|---------------------------|
| in  | <i>TIMER0_interrupt</i> | Timer 0 interrupt number. |
| out | <i>void</i>             | No output arguments.      |

## Return values

|                         |                                       |
|-------------------------|---------------------------------------|
| <i>TIMER0_OK</i>        | If timer 0 parameters are correct.    |
| <i>TIMER0_WRONG_INT</i> | If timer 0 interrupt number is wrong. |

Definition at line 98 of file [Timer\\_0.c](#).

#### 4.11.4.4 Timer0\_interruptEnable()

```
En_Timer0_Error_t Timer0_interruptEnable (
 TIMER0_interrupt_t Timer0_interrupt)
```

##### Timer0 interrupt enable

- This function enables Timer 0 interrupt.

## Parameters

|     |                         |                           |
|-----|-------------------------|---------------------------|
| in  | <i>TIMER0_interrupt</i> | Timer 0 interrupt number. |
| out | <i>void</i>             | No output arguments.      |

## Return values

|                         |                                       |
|-------------------------|---------------------------------------|
| <i>TIMER0_OK</i>        | If timer 0 parameters are correct.    |
| <i>TIMER0_WRONG_INT</i> | If timer 0 interrupt number is wrong. |

Definition at line 112 of file [Timer\\_0.c](#).

#### 4.11.4.5 Timer0\_reset()

```
void Timer0_reset (
 void)
```

**Timer0 reset**

- This function resets Timer 0 without stopping it.

## Parameters

|     |             |                      |
|-----|-------------|----------------------|
| in  | <i>void</i> | No input arguments.  |
| out | <i>void</i> | No output arguments. |

## Return values

|             |                                        |
|-------------|----------------------------------------|
| <i>void</i> | This function doesn't return anything. |
|-------------|----------------------------------------|

Definition at line 92 of file [Timer\\_0.c](#).

#### 4.11.4.6 Timer0\_start()

```
void Timer0_start (
 void)
```

**Timer0 start**

- This function starts Timer 0.

**Parameters**

|     |             |                      |
|-----|-------------|----------------------|
| in  | <i>void</i> | No input arguments.  |
| out | <i>void</i> | No output arguments. |

**Return values**

|             |                                        |
|-------------|----------------------------------------|
| <i>void</i> | This function doesn't return anything. |
|-------------|----------------------------------------|

Definition at line 77 of file [Timer\\_0.c](#).

**4.11.4.7 Timer0\_stop()**

```
void Timer0_stop (
 void)
```

**Timer0 stop**

- This function stops Timer 0.

**Parameters**

|     |             |                      |
|-----|-------------|----------------------|
| in  | <i>void</i> | No input arguments.  |
| out | <i>void</i> | No output arguments. |

**Return values**

|             |                                        |
|-------------|----------------------------------------|
| <i>void</i> | This function doesn't return anything. |
|-------------|----------------------------------------|

Definition at line 85 of file [Timer\\_0.c](#).

**4.12 MCU ports****Macros**

- #define [PORTA\\_OFFSET](#) 0
- #define [PORTB\\_OFFSET](#) 8
- #define [PORTC\\_OFFSET](#) 16
- #define [PORTD\\_OFFSET](#) 24

## Enumerations

- enum `EN_pinNum_t` {  
    `PA0` , `PA1` , `PA2` , `PA3` ,  
    `PA4` , `PA5` , `PA6` , `PA7` ,  
    `PB0` , `PB1` , `PB2` , `PB3` ,  
    `PB4` , `PB5` , `PB6` , `PB7` ,  
    `PC0` , `PC1` , `PC2` , `PC3` ,  
    `PC4` , `PC5` , `PC6` , `PC7` ,  
    `PD0` , `PD1` , `PD2` , `PD3` ,  
    `PD4` , `PD5` , `PD6` , `PD7` }
- enum `EN_pinState_t` { `Low` , `High` }
- enum `EN_pinDirection_t` { `Input` , `Output` }
- enum `EN_pinErro_t` { `OK` , `WRONG_PIN_NUM` , `WRONG_PIN_DIR` , `WRONG_PIN_STATE` }

### 4.12.1 Detailed Description

This contains all the definition for MCU pins, input and output pins values and pins errors.

### 4.12.2 Macro Definition Documentation

#### 4.12.2.1 PORTA\_OFFSET

```
#define PORTA_OFFSET 0
```

This macro defines the start of the PORTA pins

Definition at line 62 of file [ATmega32Port.h](#).

#### 4.12.2.2 PORTB\_OFFSET

```
#define PORTB_OFFSET 8
```

This macro defines the start of the PORTB pins

Definition at line 63 of file [ATmega32Port.h](#).

#### 4.12.2.3 PORTC\_OFFSET

```
#define PORTC_OFFSET 16
```

This macro defines the start of the PORTC pins

Definition at line 64 of file [ATmega32Port.h](#).

#### 4.12.2.4 PORTD\_OFFSET

```
#define PORTD_OFFSET 24
```

This macro defines the start of the PORTD pins

Definition at line 65 of file [ATmega32Port.h](#).

### 4.12.3 Enumeration Type Documentation

#### 4.12.3.1 EN\_pinDirection\_t

```
enum EN_pinDirection_t
```

Enumerator

|        |                                 |
|--------|---------------------------------|
| Input  | enum value for input direction  |
| Output | enum value for output direction |

Definition at line 72 of file [ATmega32Port.h](#).

#### 4.12.3.2 EN\_pinErro\_t

```
enum EN_pinErro_t
```

Enumerator

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| OK              | enum value that defines that the pin parameters are ok  |
| WRONG_PIN_NUM   | enum value that defines that the pin number is wrong    |
| WRONG_PIN_DIR   | enum value that defines that the pin direction is wrong |
| WRONG_PIN_STATE | enum value that defines that the pin state is wrong     |

Definition at line 77 of file [ATmega32Port.h](#).

#### 4.12.3.3 EN\_pinNum\_t

```
enum EN_pinNum_t
```

This enum contains the value for all pins of the MCU of the four ports (PORTA,PORTB,PORTC,PORTD)



## Enumerator

|     |                            |
|-----|----------------------------|
| PA0 | enum value for PORTA pin 0 |
| PA1 | enum value for PORTA pin 1 |
| PA2 | enum value for PORTA pin 2 |
| PA3 | enum value for PORTA pin 3 |
| PA4 | enum value for PORTA pin 4 |
| PA5 | enum value for PORTA pin 5 |
| PA6 | enum value for PORTA pin 6 |
| PA7 | enum value for PORTA pin 7 |
| PB0 | enum value for PORTB pin 0 |
| PB1 | enum value for PORTB pin 1 |
| PB2 | enum value for PORTB pin 2 |
| PB3 | enum value for PORTB pin 3 |
| PB4 | enum value for PORTB pin 4 |
| PB5 | enum value for PORTB pin 5 |
| PB6 | enum value for PORTB pin 6 |
| PB7 | enum value for PORTB pin 7 |
| PC0 | enum value for PORTC pin 0 |
| PC1 | enum value for PORTC pin 1 |
| PC2 | enum value for PORTC pin 2 |
| PC3 | enum value for PORTC pin 3 |
| PC4 | enum value for PORTC pin 4 |
| PC5 | enum value for PORTC pin 5 |
| PC6 | enum value for PORTC pin 6 |
| PC7 | enum value for PORTC pin 7 |
| PD0 | enum value for PORTD pin 0 |
| PD1 | enum value for PORTD pin 1 |
| PD2 | enum value for PORTD pin 2 |
| PD3 | enum value for PORTD pin 3 |
| PD4 | enum value for PORTD pin 4 |
| PD5 | enum value for PORTD pin 5 |
| PD6 | enum value for PORTD pin 6 |
| PD7 | enum value for PORTD pin 7 |

Definition at line 22 of file [ATmega32Port.h](#).

#### 4.12.3.4 EN\_pinState\_t

enum [EN\\_pinState\\_t](#)

## Enumerator

|      |                            |
|------|----------------------------|
| Low  | enum value for Low output  |
| High | enum value for high output |

Definition at line 67 of file [ATmega32Port.h](#).

## 4.13 Bit math

### Macros

- #define [setBit](#)(reg, bitNum) reg |= (1<<bitNum)  
*this Macro writes 1 to the bit.*
- #define [clrBit](#)(reg, bitNum) reg &= (~(1<<bitNum))  
*this Macro clear the bit.*
- #define [toggleBit](#)(reg, bitNum) reg ^= (1<<bitNum)  
*This Macro toggle the bit logic.*
- #define [getBit](#)(reg, bitNum) ((reg>>bitNum) & 0x01)  
*This Macro read this bit value.*

### 4.13.1 Detailed Description

Author : Ehab Omara

Date : 8/10/2022 12:46:40 PM

File name: [BitMath.h](#)

This contains all the bit math macros that manipulates the registers values.

### 4.13.2 Macro Definition Documentation

#### 4.13.2.1 clrBit

```
#define clrBit(
 reg,
 bitNum) reg &= (~(1<<bitNum))
```

this Macro clear the bit.

[clrBit](#) function

- this function takes register (reg) and bit number (bitNum).
- it make the required bit in the register Low(0).

#### Parameters

|    |               |                                                                    |
|----|---------------|--------------------------------------------------------------------|
| in | <i>reg</i>    | this is register that needed to be changed.                        |
| in | <i>bitNum</i> | this is bit number that needed to be written to 0 in the register. |

Definition at line 37 of file [BitMath.h](#).

#### 4.13.2.2 getBit

```
#define getBit(
 reg,
 bitNum) ((reg>>bitNum) & 0x01)
```

This Macro read this bit value.

[getBit](#) function

- this function takes register (*reg*) and bit number (*bitNum*).
- it returns the state of the required bit in the register.
- if the required bit is Low(0) it will return 0.
- if the required bit is High(1) it will return 1.

##### Parameters

|    |               |                                                    |
|----|---------------|----------------------------------------------------|
| in | <i>reg</i>    | This is register where it reads the value from it. |
| in | <i>bitNum</i> | This is the bit number that needed to be read.     |

Definition at line 62 of file [BitMath.h](#).

#### 4.13.2.3 setBit

```
#define setBit(
 reg,
 bitNum) reg |= (1<<bitNum)
```

this Macro writes 1 to the bit.

[setBit](#) function

- this function takes register (*reg*) and bit number (*bitNum*).
- it make the required bit in the register High(1).

##### Parameters

|    |               |                                                                    |
|----|---------------|--------------------------------------------------------------------|
| in | <i>reg</i>    | this is register that needed to be changed.                        |
| in | <i>bitNum</i> | this is bit number that needed to be written to 1 in the register. |

Definition at line 26 of file [BitMath.h](#).

#### 4.13.2.4 toggleBit

```
#define toggleBit(
 reg,
 bitNum) reg ^= (1<<bitNum)
```

This Macro toggle the bit logic.

#togBit function

- this function takes register (*reg*) and bit number (*bitNum*).
- it toggle the state of the required bit in the register.
- if the required bit is Low(0) it makes it High(1).
- if the required bit is High(1) it makes it Low(0).

##### Parameters

|    |               |                                                               |
|----|---------------|---------------------------------------------------------------|
| in | <i>reg</i>    | this is register that needed to be changed.                   |
| in | <i>bitNum</i> | this is bit number that needed to be changed in the register. |

Definition at line 50 of file [BitMath.h](#).

## 4.14 Definition of data types

### Typedefs

- typedef unsigned char [uint8\\_t](#)
- typedef signed char [sint8\\_t](#)
- typedef unsigned short int [uint16\\_t](#)
- typedef signed short int [sint16\\_t](#)
- typedef unsigned long int [uint32\\_t](#)
- typedef signed long int [sint32\\_t](#)
- typedef float [float32\\_t](#)
- typedef double [float64\\_t](#)
- typedef long double [float128\\_t](#)

#### 4.14.1 Detailed Description

This file contains all the data types definitions that needed in this project.

## 4.14.2 Typedef Documentation

### 4.14.2.1 float128\_t

```
typedef long double float128_t
```

This is define a memory size of 16 byte float

Definition at line 23 of file [dataTypes.h](#).

### 4.14.2.2 float32\_t

```
typedef float float32_t
```

This is define a memory size of 4 byte float

Definition at line 21 of file [dataTypes.h](#).

### 4.14.2.3 float64\_t

```
typedef double float64_t
```

This is define a memory size of 8 byte float

Definition at line 22 of file [dataTypes.h](#).

### 4.14.2.4 sint16\_t

```
typedef signed short int sint16_t
```

This is define a memory size of 2 byte signed

Definition at line 18 of file [dataTypes.h](#).

### 4.14.2.5 sint32\_t

```
typedef signed long int sint32_t
```

This is define a memory size of 4 byte signed

Definition at line 20 of file [dataTypes.h](#).

#### 4.14.2.6 sint8\_t

```
typedef signed char sint8_t
```

This is define a memory size of 1 byte signed

Definition at line 16 of file [dataTypes.h](#).

#### 4.14.2.7 uint16\_t

```
typedef unsigned short int uint16_t
```

This is define a memory size of 2 byte

Definition at line 17 of file [dataTypes.h](#).

#### 4.14.2.8 uint32\_t

```
typedef unsigned long int uint32_t
```

This is define a memory size of 4 byte

Definition at line 19 of file [dataTypes.h](#).

#### 4.14.2.9 uint8\_t

```
typedef unsigned char uint8_t
```

This is define a memory size of 1 byte

Definition at line 15 of file [dataTypes.h](#).

## 4.15 Service layer

### Modules

- [MCU ports](#)
- [Bit math](#)
- [Definition of data types](#)
- [MCU Registers](#)

### 4.15.1 Detailed Description

This layer contains all the common services that the other layers need like data types, MCU registers, bit math and MCU ports.

## 4.16 MCU Registers

### Modules

- [I/O registers](#)
- [Interrupt registers](#)
- [Timers Registers](#)

### 4.16.1 Detailed Description

This contains all the MCU registers definition and description for each register.

## 4.17 I/O registers

### Modules

- [Port A registers](#)
- [Port B registers](#)
- [Port C registers](#)
- [Port D registers](#)

### 4.17.1 Detailed Description

This contains all I/O registers that controls the functionality of the MCU ports.

#### Note

x may be (A,B,C, or D) and n from 0 to 7.

- Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. The DDxn bits are accessed at the DDRx I/O address, the PORTxn bits at the PORTx I/O address, and the PINxn bits at the PINx I/O address.
- The DDxn bit in the DDRx Register selects the direction of this pin. If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.
- If PORTxn is written logic one when the pin is configured as an input pin, the pull-up resistor is activated. To switch the pull-up resistor off, PORTxn has to be written logic zero or the pin has to be configured as an output pin. The port pins are tri-stated when a reset condition becomes active, even if no clocks are running. \arglf PORTxn is written logic one when the pin is configured as an output pin, the port pin is driven high (one). If PORTxn is written logic zero when the pin is configured as an out put pin, the port pin is driven low (zero).

## 4.18 Port A registers

### Macros

- `#define PORTA (*((volatile uint8_t*)0x3B))`
- `#define DDRA (*((volatile uint8_t*)0x3A))`
- `#define PINA (*((volatile uint8_t*)0x39))`

### 4.18.1 Detailed Description

### 4.18.2 Macro Definition Documentation

#### 4.18.2.1 DDRA

```
#define DDRA (*((volatile uint8_t*)0x3A))
```

#### Direction register for port A

- This register controls the direction of the pin.
- Setting the bit in this register will make the pin output.
- Clearing the bit in this register will make the pin input

Definition at line 68 of file [RegisterFile.h](#).

#### 4.18.2.2 PINA

```
#define PINA (*((volatile uint8_t*)0x39))
```

#### Input register for port A

- This register stores the input values of port A.
- If the value is 1 then the applied voltage on this pin is high.
- If the value is 0 then the applied voltage on this pin is low.

Definition at line 76 of file [RegisterFile.h](#).



### 4.18.2.3 PORTA

```
#define PORTA (*((volatile uint8_t*)0x3B))
```

#### Output register for port A

- This register controls the output of the pin.
- Setting the bit in this register will make the pin high.
- Clearing the bit in this register will make the pin low
- If the pin is configured as output through DDRx and we write high to PORTx register this will activate internal pull up resistor (x may be A,B,C or D).

Definition at line 60 of file [RegisterFile.h](#).

## 4.19 Port B registers

### Macros

- #define [PORTB](#) (\*((volatile uint8\_t\*)0x38))
- #define [DDRB](#) (\*((volatile uint8\_t\*)0x37))
- #define [PINB](#) (\*((volatile uint8\_t\*)0x36))

### 4.19.1 Detailed Description

### 4.19.2 Macro Definition Documentation

#### 4.19.2.1 DDRB

```
#define DDRB (*((volatile uint8_t*)0x37))
```

#### Direction register for port B

- This register controls the direction of the pin.
- Setting the bit in this register will make the pin output.
- Clearing the bit in this register will make the pin input

Definition at line 102 of file [RegisterFile.h](#).

#### 4.19.2.2 PINB

```
#define PINB (*((volatile uint8_t*)0x36))
```

##### Input register for port A

- This register stores the input values of port B.
- If the value is 1 then the applied voltage on this pin is high.
- If the value is 0 then the applied voltage on this pin is low.

Definition at line 110 of file [RegisterFile.h](#).

#### 4.19.2.3 PORTB

```
#define PORTB (*((volatile uint8_t*)0x38))
```

##### Output register for port B

- This register controls the output of the pin.
- Setting the bit in this register will make the pin high.
- Clearing the bit in this register will make the pin low
- If the pin is configured as output through DDRx and we write high to PORTx register this will activate internal pull up resistor (x may be A,B,C or D).

Definition at line 94 of file [RegisterFile.h](#).

## 4.20 Port C registers

### Macros

- #define [PORTC](#) (\*((volatile [uint8\\_t](#)\*)0x35))
- #define [DDRC](#) (\*((volatile [uint8\\_t](#)\*)0x34))
- #define [PINC](#) (\*((volatile [uint8\\_t](#)\*)0x33))

#### 4.20.1 Detailed Description

#### 4.20.2 Macro Definition Documentation

#### 4.20.2.1 DDRC

```
#define DDRC (*((volatile uint8_t*)0x34))
```

##### Direction register for port C

- This register controls the direction of the pin.
- Setting the bit in this register will make the pin output.
- Clearing the bit in this register will make the pin input

Definition at line 133 of file [RegisterFile.h](#).

#### 4.20.2.2 PINC

```
#define PINC (*((volatile uint8_t*)0x33))
```

##### Input register for port C

- This register stores the input values of port C.
- If the value is 1 then the applied voltage on this pin is high.
- If the value is 0 then the applied voltage on this pin is low.

Definition at line 141 of file [RegisterFile.h](#).

#### 4.20.2.3 PORTC

```
#define PORTC (*((volatile uint8_t*)0x35))
```

##### Output register for port C

- This register controls the direction of the pin.
- Setting the bit in this register will make the pin output.
- Clearing the bit in this register will make the pin input

Definition at line 125 of file [RegisterFile.h](#).

## 4.21 Port D registers

### Macros

- `#define PORTD (*((volatile uint8_t*)0x32))`
- `#define DDRD (*((volatile uint8_t*)0x31))`
- `#define PIND (*((volatile uint8_t*)0x30))`

### 4.21.1 Detailed Description

### 4.21.2 Macro Definition Documentation

#### 4.21.2.1 DDRD

```
#define DDRD (*((volatile uint8_t*)0x31))
```

#### Direction register for port D

- This register controls the direction of the pin.
- Setting the bit in this register will make the pin output.
- Clearing the bit in this register will make the pin input

Definition at line 164 of file [RegisterFile.h](#).

#### 4.21.2.2 PIND

```
#define PIND (*((volatile uint8_t*)0x30))
```

#### Input register for port D

- This register stores the input values of port D.
- If the value is 1 then the applied voltage on this pin is high.
- If the value is 0 then the applied voltage on this pin is low.

Definition at line 172 of file [RegisterFile.h](#).

### 4.21.2.3 PORTD

```
#define PORTD (*((volatile uint8_t*)0x32))
```

#### Output register for port D

- This register controls the direction of the pin.
- Setting the bit in this register will make the pin output.
- Clearing the bit in this register will make the pin input

Definition at line 156 of file [RegisterFile.h](#).

## 4.22 Interrupt registers

### Macros

- #define [GICR](#) (\*((volatile uint8\_t\*)0x5B))
- #define [GIFR](#) (\*((volatile uint8\_t\*)0x5A))
- #define [MCUCR](#) (\*((volatile uint8\_t\*)0x55))
- #define [MCUCSR](#) (\*((volatile uint8\_t\*)0x54))

### 4.22.1 Detailed Description

### 4.22.2 Macro Definition Documentation

#### 4.22.2.1 GICR

```
#define GICR (*((volatile uint8_t*)0x5B))
```

General Interrupt Control Register.

| Bit           | 7           | 6           | 5           | 4 | 3 | 2 | 1            | 0           |             |
|---------------|-------------|-------------|-------------|---|---|---|--------------|-------------|-------------|
|               | <b>INT1</b> | <b>INT0</b> | <b>INT2</b> | – | – | – | <b>IVSEL</b> | <b>IVCE</b> | <b>GICR</b> |
| Read/Write    | R/W         | R/W         | R/W         | R | R | R | R/W          | R/W         |             |
| Initial Value | 0           | 0           | 0           | 0 | 0 | 0 | 0            | 0           |             |

- Bit 7 - INT1: External Interrupt Request 1 Enable
- Bit 6 - INT0: External Interrupt Request 0 Enable
- Bit 5 - INT2: External Interrupt Request 2 Enable

Definition at line 189 of file [RegisterFile.h](#).

#### 4.22.2.2 GIFR

```
#define GIFR (*(volatile uint8_t*)0x5A)
```

**General Interrupt Flag Register.**

| Bit           | 7            | 6            | 5            | 4 | 3 | 2 | 1 | 0 |             |
|---------------|--------------|--------------|--------------|---|---|---|---|---|-------------|
|               | <b>INTF1</b> | <b>INTF0</b> | <b>INTF2</b> | – | – | – | – | – | <b>GIFR</b> |
| Read/Write    | R/W          | R/W          | R/W          | R | R | R | R | R |             |
| Initial Value | 0            | 0            | 0            | 0 | 0 | 0 | 0 | 0 |             |

- Bit 7 - INTF1: External Interrupt Flag 1
- Bit 6 - INTF0: External Interrupt Flag 0
- Bit 5 - INTF2: External Interrupt Flag 2

Definition at line 200 of file [RegisterFile.h](#).

#### 4.22.2.3 MCUCR

```
#define MCUCR (*(volatile uint8_t*)0x55)
```

**MCU Control Register.**

| Bit           | 7         | 6          | 5          | 4          | 3            | 2            | 1            | 0            |              |
|---------------|-----------|------------|------------|------------|--------------|--------------|--------------|--------------|--------------|
|               | <b>SE</b> | <b>SM2</b> | <b>SM1</b> | <b>SM0</b> | <b>ISC11</b> | <b>ISC10</b> | <b>ISC01</b> | <b>ISC00</b> | <b>MCUCR</b> |
| Read/Write    | R/W       | R/W        | R/W        | R/W        | R/W          | R/W          | R/W          | R/W          |              |
| Initial Value | 0         | 0          | 0          | 0          | 0            | 0            | 0            | 0            |              |

- Bit 3, 2 - ISC11, ISC10: Interrupt Sense Control 1 Bit 1 and Bit 0.

| ISCx1                                            | ISCx0 | Description                                                |
|--------------------------------------------------|-------|------------------------------------------------------------|
| 0                                                | 0     | The low level of INTx generates an interrupt request.      |
| 0                                                | 1     | Any logical change on INTx generates an interrupt request. |
| 1                                                | 0     | The falling edge of INTx generates an interrupt request.   |
| 1                                                | 1     | The rising edge of INTx generates an interrupt request.    |
| <b>Interrupt 0 and interrupt 1 Sense Control</b> |       |                                                            |

Note

x may be 0 or 1.

Definition at line 218 of file [RegisterFile.h](#).

#### 4.22.2.4 MCUCSR

```
#define MCUCSR (*(volatile uint8_t*)0x54)
```

**MCU Control and Status Register.**

| Bit           | 7   | 6    | 5 | 4    | 3    | 2    | 1     | 0    |        |
|---------------|-----|------|---|------|------|------|-------|------|--------|
|               | JTD | ISC2 | – | JTRF | WDRF | BORF | EXTRF | PORF | MCUCSR |
| Read/Write    | R/W | R/W  | R | R/W  | R/W  | R/W  | R/W   | R/W  |        |
| Initial Value | 0   | 0    | 0 |      |      |      |       |      |        |

- Bit 6 - ISC2: Interrupt Sense Control 2

| ISC2 | Description                                               |
|------|-----------------------------------------------------------|
| 0    | The falling edge on INT2 activates the interrupt request. |
| 1    | The rising edge on INT2 activates the interrupt request.  |

Definition at line 231 of file [RegisterFile.h](#).

## 4.23 Timers Registers

### Modules

- [Timer0 Registers](#)
- [General Timers registers](#)

#### 4.23.1 Detailed Description

## 4.24 Timer0 Registers

### Macros

- #define [TCCR0](#) (\*(volatile uint8\_t\*)0x53)
- #define [TCNT0](#) (\*(volatile uint8\_t\*)0x52)
- #define [OCR0](#) (\*(volatile uint8\_t\*)0x5C)

#### 4.24.1 Detailed Description

- This contains all the registers to control Timer0.

## 4.24.2 Macro Definition Documentation

### 4.24.2.1 OCR0

```
#define OCR0 (*(volatile uint8_t*)0x5C)
```

#### Output Compare Register

| Bit           | 7         | 6   | 5   | 4   | 3   | 2   | 1   | 0   |      |
|---------------|-----------|-----|-----|-----|-----|-----|-----|-----|------|
|               | OCR0[7:0] |     |     |     |     |     |     |     | OCR0 |
| Read/Write    | R/W       | R/W | R/W | R/W | R/W | R/W | R/W | R/W |      |
| Initial Value | 0         | 0   | 0   | 0   | 0   | 0   | 0   | 0   |      |

- The Output Compare Register contains an 8-bit value that is continuously compared with the counter value (TCNT0).
- A match can be used to generate an output compare interrupt, or to generate a waveform output on the OC0 pin.

Definition at line 344 of file [RegisterFile.h](#).

### 4.24.2.2 TCCR0

```
#define TCCR0 (*(volatile uint8_t*)0x53)
```

#### Timer/Counter Control Register.

| Bit           | 7    | 6     | 5     | 4     | 3     | 2    | 1    | 0    |       |
|---------------|------|-------|-------|-------|-------|------|------|------|-------|
|               | FOC0 | WGM00 | COM01 | COM00 | WGM01 | CS02 | CS01 | CS00 | TCCR0 |
| Read/Write    | W    | R/W   | R/W   | R/W   | R/W   | R/W  | R/W  | R/W  |       |
| Initial Value | 0    | 0     | 0     | 0     | 0     | 0    | 0    | 0    |       |

#### Bit 7 - FOC0: Force Output Compare

- When writing a logical one to the FOC0 bit, an immediate compare match is forced on the Waveform Generation unit.
- These bits control the counting sequence of the counter, the source for the maximum (TOP) counter value, and what type of Waveform Generation to be used.



| Mode                                            | WGM01 (CTC0) | WGM00 (PWM0) | Timer/Counter Mode of Operation | TOP  | Update of OCR0 | TOV0 Flag Set-on |
|-------------------------------------------------|--------------|--------------|---------------------------------|------|----------------|------------------|
| 0                                               | 0            | 0            | Normal                          | 0xFF | Immediate      | MAX              |
| 1                                               | 0            | 1            | PWM, Phase Correct              | 0xFF | TOP            | BOTTOM           |
| 2                                               | 1            | 0            | CTC                             | OCR0 | Immediate      | MAX              |
| 3                                               | 1            | 1            | Fast PWM                        | 0xFF | BOTTOM         | MAX              |
| <b>Waveform Generation Mode Bit Description</b> |              |              |                                 |      |                |                  |

**Bit 5:4 - COM01:0: Compare Match Output Mode**

- These bits control the Output Compare pin (OC0) behavior.
- If one or both of the COM01:0 bits are set, the OC0 output overrides the normal port functionality of the I/O pin it is connected to.

**Note**

The Data Direction Register (DDR) bit corresponding to the OC0 pin must be set in order to enable the output driver.

- When OC0 is connected to the pin, the function of the COM01:0 bits depends on the WGM01:0 bit setting.

| COM01                                    | COM00 | Description                              |
|------------------------------------------|-------|------------------------------------------|
| 0                                        | 0     | Normal port operation, OC0 disconnected. |
| 0                                        | 1     | Toggle OC0 on compare match.             |
| 1                                        | 0     | Clear OC0 on compare match.              |
| 1                                        | 1     | Set OC0 on compare match.                |
| <b>Compare Output Mode, non-PWM Mode</b> |       |                                          |

| COM01                                     | COM00 | Description                                                         |
|-------------------------------------------|-------|---------------------------------------------------------------------|
| 0                                         | 0     | Normal port operation, OC0 disconnected.                            |
| 0                                         | 1     | Reserved.                                                           |
| 1                                         | 0     | Clear OC0 on compare match, set OC0 at BOTTOM,(non-inverting mode). |
| 1                                         | 1     | Set OC0 on compare match, clear OC0 at BOTTOM,(inverting mode)      |
| <b>Compare Output Mode, Fast PWM Mode</b> |       |                                                                     |

**Note**

A special case occurs when OCR0 equals TOP and COM01 is set. In this case, the compare match is ignored, but the set or clear is done at BOTTOM.

| COM01                                     | COM00 | Description                                                                                 |
|-------------------------------------------|-------|---------------------------------------------------------------------------------------------|
| 0                                         | 0     | Normal port operation, OC0 disconnected.                                                    |
| 0                                         | 1     | Reserved.                                                                                   |
| 1                                         | 0     | Clear OC0 on compare match when up-counting.<br>Set OC0 on compare match when downcounting. |
| 1                                         | 1     | Set OC0 on compare match when up-counting.<br>Clear OC0 on compare match when downcounting. |
| <b>Compare Output Mode, Fast PWM Mode</b> |       |                                                                                             |

**Note**

A special case occurs when OCR0 equals TOP and COM01 is set. In this case, the compare match is ignored, but the set or clear is done at TOP.

**Bit 2:0 - CS02:0: Clock Select**

- The three Clock Select bits select the clock source to be used by the Timer/Counter.

| CS02                                | CS01 | CS00 | Description                                                |
|-------------------------------------|------|------|------------------------------------------------------------|
| 0                                   | 0    | 0    | No clock source (Timer/Counter stopped).                   |
| 0                                   | 0    | 1    | clkI/O/(No prescaling).                                    |
| 0                                   | 1    | 0    | clkI/O/8 (From prescaler).                                 |
| 0                                   | 1    | 1    | clkI/O/64 (From prescaler).                                |
| 1                                   | 0    | 0    | clkI/O/256 (From prescaler).                               |
| 1                                   | 0    | 1    | clkI/O/1024 (From prescaler).                              |
| 1                                   | 1    | 0    | External clock source on T0 pin.<br>Clock on falling edge. |
| 1                                   | 1    | 1    | External clock source on T0 pin.<br>Clock on rising edge.  |
| <b>Clock Select Bit Description</b> |      |      |                                                            |

**Note**

If external pin modes are used for the Timer/Counter0, transitions on the T0 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

Definition at line 321 of file [RegisterFile.h](#).

**4.24.2.3 TCNT0**

```
#define TCNT0 (*((volatile uint8_t*)0x52))
```

**Timer/Counter Register**

| Bit           | 7          | 6   | 5   | 4   | 3   | 2   | 1   | 0   |       |
|---------------|------------|-----|-----|-----|-----|-----|-----|-----|-------|
|               | TCNT0[7:0] |     |     |     |     |     |     |     | TCNT0 |
| Read/Write    | R/W        | R/W | R/W | R/W | R/W | R/W | R/W | R/W |       |
| Initial Value | 0          | 0   | 0   | 0   | 0   | 0   | 0   | 0   |       |

- The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter.
- Writing to the TCNT0 Register blocks (removes) the compare match on the following timer clock.
- Modifying the counter (TCNT0) while the counter is running, introduces a risk of missing a compare match between TCNT0 and the OCR0 Register.

Definition at line 333 of file [RegisterFile.h](#).

**4.25 General Timers registers****Macros**

- `#define TIFR (*(volatile uint8_t*)0x58)`
- `#define TIMSK (*(volatile uint8_t*)0x59)`

**4.25.1 Detailed Description****4.25.2 Macro Definition Documentation****4.25.2.1 TIFR**

```
#define TIFR (*(volatile uint8_t*)0x58)
```

**Timer/Counter Interrupt Flag Register**

| Bit           | 7    | 6    | 5    | 4     | 3     | 2    | 1    | 0    |      |
|---------------|------|------|------|-------|-------|------|------|------|------|
|               | OCF2 | TOV2 | ICF1 | OCF1A | OCF1B | TOV1 | OCF0 | TOV0 | TIFR |
| Read/Write    | R/W  | R/W  | R/W  | R/W   | R/W   | R/W  | R/W  | R/W  |      |
| Initial Value | 0    | 0    | 0    | 0     | 0     | 0    | 0    | 0    |      |

**Bit 1 - OCF0: Output Compare Flag 0**

- The OCF0 bit is set (one) when a compare match occurs between the Timer/Counter0 and the data in OCR0 - Output Compare Register0.
  - OCF0 is cleared by hardware when executing the corresponding interrupt handling vector.
  - Alternatively, OCF0 is cleared by writing a logic one to the flag.
  - When the I-bit in SREG, OCIE0 (Timer/Counter0 Compare Match Interrupt Enable), and OCF0 are set (one), the Timer/Counter0 Compare Match Interrupt is executed.
- 
- The bit TOV0 is set (one) when an overflow occurs in Timer/Counter0.
  - TOV0 is cleared by hardware when executing the corresponding interrupt handling vector.
  - Alternatively, TOV0 is cleared by writing a logic one to the flag. When the SREG I-bit, TOIE0 (Timer/Counter0 Overflow Interrupt Enable), and TOV0 are set (one), the Timer/Counter0 Overflow interrupt is executed.
  - In phase correct PWM mode, this bit is set when Timer/Counter0 changes counting direction at \$00.

Definition at line 374 of file [RegisterFile.h](#).

**4.25.2.2 TIMSK**

```
#define TIMSK (*((volatile uint8_t*)0x59))
```

**Timer/Counter Interrupt Mask Register**

| Bit           | 7     | 6     | 5      | 4      | 3      | 2     | 1     | 0     |       |
|---------------|-------|-------|--------|--------|--------|-------|-------|-------|-------|
|               | OCIE2 | TOIE2 | TICIE1 | OCIE1A | OCIE1B | TOIE1 | OCIE0 | TOIE0 | TIMSK |
| Read/Write    | R/W   | R/W   | R/W    | R/W    | R/W    | R/W   | R/W   | R/W   |       |
| Initial Value | 0     | 0     | 0      | 0      | 0      | 0     | 0     | 0     |       |

**Bit 1 - OCIE0: Timer/Counter0 Output Compare Match Interrupt Enable**

- When the OCIE0 bit is written to one, and the I-bit in the Status Register is set (one), the Timer/Counter0 Compare Match interrupt is enabled.
  - The corresponding interrupt is executed if a compare match in Timer/Counter0 occurs, i.e., when the OCF0 bit is set in the Timer/Counter Interrupt Flag Register - TIFR.
- 
- When the TOIE0 bit is written to one, and the I-bit in the Status Register is set (one), the Timer/Counter0 Overflow interrupt is enabled.
  - The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter Interrupt Flag Register - TIFR.

Definition at line 391 of file [RegisterFile.h](#).

## Chapter 5

# File Documentation

### 5.1 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/App/app.c File Reference

```
#include "../ECUAL/LED driver/LED.h"
#include "../ECUAL/Button driver/Button.h"
#include "../MCAL/Timer driver/Timer_0.h"
#include "../MCAL/Ext interrupt driver/Ext interrupt.h"
#include "app.h"
```

#### Macros

- #define `SYSTEM_CLK` 1000000

#### Functions

- void `app_start` (void)  
*app\_start this function used to initialize the MCU for the application*
- void `carsGreenState` (void)  
*carsGreenState*
- void `carsYellowState` (void)  
*carsYellowState*
- void `carsRedState` (void)  
*carsRedState*
- void `trafficLightOperation` (void)
- void `pedestriansGreenState` (void)  
*pedestriansGreenState*
- void `pedestriansYellowState` (void)  
*pedestriansYellowState*
- void `pedestriansRedState` (void)  
*pedestriansRedState*
- `ISR (EXT_INT0)`

## Variables

- static `EN_TrafficStates_t system_CurrentState = NORMAL_GREEN`  
*system current state*

### 5.1.1 Macro Definition Documentation

#### 5.1.1.1 SYSTEM\_CLK

```
#define SYSTEM_CLK 1000000
```

Definition at line 7 of file [app.c](#).

### 5.1.2 Function Documentation

#### 5.1.2.1 ISR()

```
ISR (
 EXT_INT0)
```

Definition at line 180 of file [app.c](#).

### 5.1.3 Variable Documentation

#### 5.1.3.1 system\_CurrentState

```
EN_TrafficStates_t system_CurrentState = NORMAL_GREEN [static]
```

*system current state*

- This variable store the current state of the system.

Definition at line 19 of file [app.c](#).

## 5.2 app.c

[Go to the documentation of this file.](#)

```

00001 /*****
00002 /* Author : Ehab Omara
00003 /* Date : 8/10/2022 12:03:35 PM
00004 /* File name: app.c
00005 /*
00006 /*****
00007 #define SYSTEM_CLK 1000000
00008 #include "../ECUAL/LED driver/LED.h"
00009 #include "../ECUAL/Buton driver/Button.h"
00010 #include "../MCAL/Timer driver/Timer_0.h"
00011 #include "../MCAL/Ext interrupt driver/Ext interrupt.h"
00012 #include "app.h"
00013
00019 EN_TrafficStates_t static system_CurrentState = NORMAL_GREEN;
00020
00021 void app_start(void)
00022 {
00023 //initialize cars leds as output
00024 ledInit(CARS_GREEN_LED);
00025 ledInit(CARS_YELLOW_LED);
00026 ledInit(CARS_RED_LED);
00027
00028 //initialize pedestrians leds as output
00029 ledInit(PEDESTRIANS_GREEN_LED);
00030 ledInit(PEDESTRIANS_YELLOW_LED);
00031 ledInit(PEDESTRIANS_RED_LED);
00032
00033 //enable #INT0 to trigger at falling edge
00034 Ext_interruptInit(INT0,FALLING_EDGE);
00035
00036 //configure Timer 0 at normal mode and prescaler 1024
00037 Timer0_init(NORMAL,clkI_DIVISION_BY_1024);
00038
00039 }
00040
00041 void carsGreenState(void)
00042 {
00043 ledOff(CARS_RED_LED); //turn cars red led off
00044 ledOn(CARS_GREEN_LED); //turns card green led on
00045 Timer0_delay_ms(5000); //wait for 5 sec
00046 }
00047
00048 void carsYellowState(void)
00049 {
00050 ledOff(CARS_YELLOW_LED);
00051 //toggle cars yellow led for 5 sec
00052 for(uint8_t i = 0; i < 10; i++)
00053 {
00054 ledToggle(CARS_YELLOW_LED);
00055 Timer0_delay_ms(500);
00056 //check if the system state changed due to button pressed then break from the loop
00057 if (system_CurrentState != NORMAL_YELLOW_AFTER_GREEN && system_CurrentState !=
NORMAL_YELLOW_AFTER_RED)
00058 {
00059 break;
00060 }
00061 }
00062 ledOff(CARS_YELLOW_LED);
00063 }
00064 void carsRedState(void)
00065 {
00066 ledOff(CARS_GREEN_LED); //turn cars green led off
00067 ledOn(CARS_RED_LED); //turns card red led on
00068 Timer0_delay_ms(5000); //wait for 5 sec
00069 }
00070
00071 void trafficLightOperation(void)
00072 {
00073 /*****<Normal mode
states>*****/
00074 ledOn(PEDESTRIANS_RED_LED); //turn pedestrians red led on
00075 /*check if the current state is NORMAL_GREEN*/
00076 if (system_CurrentState == NORMAL_GREEN)
00077 {
00078 //execute cars green state

```

```

00079 carsGreenState();
00080 //after the state finished execution
00081 //check if i still in the same state
00082 //as the state might be changed due to a button pressed
00083 // if i still in the same state then change the current state to the next one which is in this
example NORMAL_YELLOW_AFTER_GREEN
00084 //if the current state changed then do nothing just go to the state which assigned by the
button
00085 if (system_CurrentState == NORMAL_GREEN)
00086 {
00087 system_CurrentState = NORMAL_YELLOW_AFTER_GREEN;
00088 }
00089 }
00090 /*normal yellow after green state*/
00091 if (system_CurrentState == NORMAL_YELLOW_AFTER_GREEN)
00092 {
00093 carsYellowState();
00094 if (system_CurrentState == NORMAL_YELLOW_AFTER_GREEN)
00095 {
00096 system_CurrentState = NORMAL_RED;
00097 }
00098 }
00099 /*normal red state */
00100 if (system_CurrentState == NORMAL_RED)
00101 {
00102 carsRedState();
00103 if (system_CurrentState == NORMAL_RED)
00104 {
00105 system_CurrentState = NORMAL_YELLOW_AFTER_RED;
00106 }
00107 }
00108 /*normal yellow after red state*/
00109 if (system_CurrentState == NORMAL_YELLOW_AFTER_RED)
00110 {
00111 carsYellowState();
00112 if (system_CurrentState == NORMAL_YELLOW_AFTER_RED)
00113 {
00114 system_CurrentState = NORMAL_GREEN;
00115 }
00116 }
00117 /*****<Pedestrians mode
states>*****/
00118 //if the button pressed at green state or one of yellow state
00119 if (system_CurrentState == BUTTON_PRESSED_AT_GREEN || system_CurrentState ==
BUTTON_PRESSED_AT_YELLOW)
00120 {
00121 ledOn (PEDESTRIANS_RED_LED);
00122
00123 pedestriansYellowState();
00124
00125 pedestriansGreenState();
00126
00127 pedestriansYellowState();
00128
00129 pedestriansRedState();
00130 system_CurrentState = NORMAL_GREEN;
00131 }
00132 //if the button pressed at red state
00133 if (system_CurrentState == BUTTON_PRESSED_AT_RED)
00134 {
00135 pedestriansGreenState();
00136
00137 pedestriansYellowState();
00138
00139 pedestriansRedState();
00140 system_CurrentState = NORMAL_GREEN;
00141 }
00142 }
00143 }
00144
00145
00146
00147
/*****
00148 void pedestriansGreenState(void)
00149 {
00150 ledOn (PEDESTRIANS_GREEN_LED); //turn pedestrians green led on
00151 ledOn (CARS_RED_LED); //turn cars red led on
00152 ledOff (CARS_GREEN_LED); //turn cars green led off
00153 ledOff (PEDESTRIANS_RED_LED); //turn pedestrians red led off
00154 Timer0_delay_ms (5000); //wait for 5 sec
00155 }
00156
00157 void pedestriansYellowState(void)
00158 {
00159 ledOff (CARS_YELLOW_LED);
00160 ledOff (PEDESTRIANS_YELLOW_LED);

```



```

00161 //blink pedestrians and cars yellow led for 5 sec
00162 for(uint8_t i = 0; i < 10; i++)
00163 {
00164 ledToggle(CARS_YELLOW_LED);
00165 ledToggle(PEDESTRIANS_YELLOW_LED);
00166 Timer0_delay_ms(500);
00167 }
00168 ledOff(CARS_YELLOW_LED);
00169 ledOff(PEDESTRIANS_YELLOW_LED);
00170 }
00171
00172 void pedestriansRedState(void)
00173 {
00174 ledOn(CARS_GREEN_LED); //turn cars green led on
00175 ledOn(PEDESTRIANS_RED_LED); //turn pedestrians red led on
00176 ledOff(PEDESTRIANS_GREEN_LED); //turn pedestrians green led off
00177 ledOff(CARS_RED_LED); //turn cars red led off
00178 }
00179
00180 /*****
00180 ISR(EXT_INT0)
00181 {
00182 //if the button pressed at normal green state then switch to #BUTTON_PRESSED_AT_GREEN state
00183 if (system_CurrentState == NORMAL_GREEN)
00184 {
00185 system_CurrentState = BUTTON_PRESSED_AT_GREEN;
00186 }
00187 //if the button pressed at one of yellow states then switch to #BUTTON_PRESSED_AT_YELLOW state
00188 else if (system_CurrentState == NORMAL_YELLOW_AFTER_GREEN || system_CurrentState ==
NORMAL_YELLOW_AFTER_RED)
00189 {
00190 system_CurrentState = BUTTON_PRESSED_AT_YELLOW;
00191 }
00192 //if the button pressed at red state the switch to #BUTTON_PRESSED_AT_RED state
00193 else if (system_CurrentState == NORMAL_RED)
00194 {
00195 system_CurrentState = BUTTON_PRESSED_AT_RED;
00196 }
00197 else
00198 {
00199 //do nothing as at this state the button pressed at pedestrians mode.
00200 //this will neglect two press and long press.
00201 }
00202 }

```

## 5.3 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/App/app.h File Reference

```

#include "../Service/ATmega32Port.h"
#include "../Service/dataTypes.h"

```

### Macros

#### Cars LEDS Macros

*These macros represent the pins which cars leds connected to.*

- #define CARS\_GREEN\_LED PA0
- #define CARS\_YELLOW\_LED PA1
- #define CARS\_RED\_LED PA2

#### Pedestrians LEDS Macros

*These macros represent the pins which pedestrians leds connected to.*

- #define PEDESTRIANS\_GREEN\_LED PB0
- #define PEDESTRIANS\_YELLOW\_LED PB1
- #define PEDESTRIANS\_RED\_LED PB2

## Enumerations

- enum [EN\\_TrafficStates\\_t](#) {  
[NORMAL\\_GREEN](#) , [NORMAL\\_YELLOW\\_AFTER\\_GREEN](#) , [NORMAL\\_RED](#) , [NORMAL\\_YELLOW\\_AFTER\\_RED](#)  
 ,  
[BUTTON\\_PRESSED\\_AT\\_GREEN](#) , [BUTTON\\_PRESSED\\_AT\\_RED](#) , [BUTTON\\_PRESSED\\_AT\\_YELLOW](#) }

*This enum contains the values for each state of the system.*

## Functions

- void [app\\_start](#) (void)  
*app\_start this function used to initialize the MCU for the application*
- void [trafficLightOperation](#) (void)

### Cars led control function

*These function controls the leds of the cars. \It is used in normal mode states.*

#### Parameters

|            |      |                                                          |
|------------|------|----------------------------------------------------------|
| <i>in</i>  | None | <i>this function doesn't take any input parameters.</i>  |
| <i>out</i> | None | <i>this function doesn't take any output parameters.</i> |

#### Return values

|      |                                               |
|------|-----------------------------------------------|
| None | <i>This function doesn't return anything.</i> |
|------|-----------------------------------------------|

- void [carsGreenState](#) (void)  
*carsGreenState*
- void [carsYellowState](#) (void)  
*carsYellowState*
- void [carsRedState](#) (void)  
*carsRedState*

### Pedestrians led control function

*These function controls the leds of the pedestrians. \It is used in pedestrians mode states.*

#### Parameters

|            |      |                                                          |
|------------|------|----------------------------------------------------------|
| <i>in</i>  | None | <i>this function doesn't take any input parameters.</i>  |
| <i>out</i> | None | <i>this function doesn't take any output parameters.</i> |

#### Return values

|      |                                               |
|------|-----------------------------------------------|
| None | <i>This function doesn't return anything.</i> |
|------|-----------------------------------------------|

- void [pedestriansGreenState](#) (void)  
*pedestriansGreenState*
- void [pedestriansYellowState](#) (void)  
*pedestriansYellowState*

- void `pedestriansRedState` (void)  
*pedestriansRedState*

## 5.4 app.h

[Go to the documentation of this file.](#)

```

00001
00002 /***** Author : Ehab Omara *****/
00003 /* Date : 8/10/2022 12:03:55 PM */
00004 /* File name: app.h */
00005 /*****
00006
00007 #ifndef APP_H_
00008 #define APP_H_
00009 #include "../Service/ATmega32Port.h"
00010 #include "../Service/dataTypes.h"
00023 #define CARS_GREEN_LED PA0
00024 #define CARS_YELLOW_LED PA1
00025 #define CARS_RED_LED PA2
00027
00034 #define PEDESTRIANS_GREEN_LED PB0
00035 #define PEDESTRIANS_YELLOW_LED PB1
00036 #define PEDESTRIANS_RED_LED PB2
00038
00042 typedef enum
00043 {
00044 NORMAL_GREEN,
00045 NORMAL_YELLOW_AFTER_GREEN,
00046 NORMAL_RED,
00047 NORMAL_YELLOW_AFTER_RED,
00048 BUTTON_PRESSED_AT_GREEN,
00049 BUTTON_PRESSED_AT_RED,
00050 BUTTON_PRESSED_AT_YELLOW,
00051 }EN_TrafficStates_t;
00052
00063 void app_start(void);
00064
00074 void trafficLightOperation(void);
00075
00076
00086
00090 void carsGreenState(void);
00098 void carsYellowState(void);
00106 void carsRedState(void);
00108
00118
00127 void pedestriansGreenState(void);
00136 void pedestriansYellowState(void);
00146 void pedestriansRedState(void);
00148
00149 #endif /* APP_H_ */

```

## 5.5 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/Button driver/Button.c File Reference

```
#include "Button.h"
```

### Functions

- `EN_pinErro_t buttonInit` (EN\_pinNum\_t buttonPin)  
*initialize the button pin.*
- `EN_pinErro_t buttonRead` (EN\_pinNum\_t buttonPin, EN\_pinState\_t \*pinState)  
*reads the value of the button.*

## 5.6 Button.c

[Go to the documentation of this file.](#)

```

00001
00002
00003 /***** Author : Ehab Omara
00004 */
00005 /***** Date : 8/11/2022 8:25:13 PM
00006 */
00007 /***** File name: Button.c
00008 *****/
00009 #include "Button.h"
00010
00011 EN_pinErro_t buttonInit (EN_pinNum_t buttonPin)
00012 {
00013 return DIO_pinInit(buttonPin, Input);
00014 }
00015 /*****/
00016 EN_pinErro_t buttonRead (EN_pinNum_t buttonPin, EN_pinState_t *pinState)
00017 {
00018 return DIO_pinRead(buttonPin, pinState);
00019 }

```

## 5.7 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/Button driver/Button.h File Reference

```

#include "../Service/ATmega32Port.h"
#include "../MCAL/Dio driver/DIO.h"

```

### Functions

- [EN\\_pinErro\\_t buttonInit \(EN\\_pinNum\\_t buttonPin\)](#)  
*initialize the button pin.*
- [EN\\_pinErro\\_t buttonRead \(EN\\_pinNum\\_t buttonPin, EN\\_pinState\\_t \\*pinState\)](#)  
*reads the value of the button.*

## 5.8 Button.h

[Go to the documentation of this file.](#)

```

00001
00002 /***** Author : Ehab Omara
00003 */
00004 /***** Date : 8/11/2022 8:24:25 PM
00005 */
00006 /***** File name: Button.h
00007 *****/
00008 #ifndef BUTTON_H_
00009 #define BUTTON_H_
00010
00011 #include "../Service/ATmega32Port.h"
00012 #include "../MCAL/Dio driver/DIO.h"
00013
00014 EN_pinErro_t buttonInit (EN_pinNum_t buttonPin);
00015
00016 /*****/
00017 EN_pinErro_t buttonRead (EN_pinNum_t buttonPin, EN_pinState_t *pinState);
00018
00019 #endif /* BUTTON_H_ */

```

## 5.9 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/LED driver/LED.c File Reference

```
#include "LED.h"
```

### Functions

- [EN\\_pinErro\\_t ledInit \(EN\\_pinNum\\_t ledPin\)](#)  
*initialize the led pin.*
- [EN\\_pinErro\\_t ledOn \(EN\\_pinNum\\_t ledPin\)](#)  
*turn the led on.*
- [EN\\_pinErro\\_t ledOff \(EN\\_pinNum\\_t ledPin\)](#)  
*turn the led off.*
- [EN\\_pinNum\\_t ledToggle \(EN\\_pinNum\\_t ledPin\)](#)  
*toggle the led state.*

## 5.10 LED.c

[Go to the documentation of this file.](#)

```
00001
00002 /***** Author : Ehab Omara *****/
00003 /* Date : 8/12/2022 9:42:19 PM */
00004 /* File name: LED.c */
00005
00006 #include "LED.h"
00007
00008
00009
00010 EN_pinErro_t ledInit (EN_pinNum_t ledPin)
00011 {
00012 return DIO_pinInit (ledPin, Output);
00013 }
00014 /*****/
00015 EN_pinErro_t ledOn (EN_pinNum_t ledPin)
00016 {
00017 return DIO_pinWrite (ledPin, High);
00018 }
00019 /*****/
00020 EN_pinErro_t ledOff (EN_pinNum_t ledPin)
00021 {
00022 return DIO_pinWrite (ledPin, Low);
00023 }
00024 /*****/
00025 EN_pinNum_t ledToggle (EN_pinNum_t ledPin)
00026 {
00027 return DIO_pinToggle (ledPin);
00028 }
```

## 5.11 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/LED driver/LED.h File Reference

```
#include "../Service/ATmega32Port.h"
#include "../MCAL/Dio driver/DIO.h"
```

## Functions

- [EN\\_pinErro\\_t ledInit \(EN\\_pinNum\\_t ledPin\)](#)  
*initialize the led pin.*
- [EN\\_pinErro\\_t ledOn \(EN\\_pinNum\\_t ledPin\)](#)  
*turn the led on.*
- [EN\\_pinErro\\_t ledOff \(EN\\_pinNum\\_t ledPin\)](#)  
*turn the led off.*
- [EN\\_pinNum\\_t ledToggle \(EN\\_pinNum\\_t ledPin\)](#)  
*toggle the led state.*

## 5.12 LED.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002 /* Author : Ehab Omara
00003 /* Date : 8/12/2022 9:42:50 PM
00004 /* File name: LED.h
00005 /*
00006 /*****
00007 #ifndef LED_H_
00008 #define LED_H_
00009
00010 #include "../Service/ATmega32Port.h"
00011 #include "../MCAL/Dio driver/DIO.h"
00012
00032 EN_pinErro_t ledInit (EN_pinNum_t ledPin);
00033 /*****/
00046 EN_pinErro_t ledOn (EN_pinNum_t ledPin);
00047 /*****/
00060 EN_pinErro_t ledOff (EN_pinNum_t ledPin);
00061 /*****/
00076 EN_pinNum_t ledToggle (EN_pinNum_t ledPin);
00081 #endif /* LED_H_ */

```

## 5.13 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/IncFile1.h File Reference

## 5.14 IncFile1.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002 /* Author : Ehab Omara
00003 /* Date : 8/17/2022 8:27:24 PM
00004 /* File name: IncFile1.h
00005 /*
00006 /*****
00007 #ifndef INCFILE1_H_
00008 #define INCFILE1_H_
00043 #endif /* INCFILE1_H_ */

```

## 5.15 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/main.c File Reference

```
#include "App/app.h"
```

### Functions

- int [main](#) (void)

### 5.15.1 Function Documentation

#### 5.15.1.1 main()

```
int main (
 void)
```

Definition at line 9 of file [main.c](#).

## 5.16 main.c

[Go to the documentation of this file.](#)

```
00001 /*****
00002 /* Author : Ehab Omara
00003 /* Date : 8/10/2022 12:00:19 PM
00004 /* File name: main.c
00005 /*
00006 /*****
00007
00008 #include "App/app.h"
00009 int main(void)
00010 {
00011 app_start();
00012 while (1)
00013 {
00014 trafficLightOperation();
00015 }
00016 return 0;
00017 }
00018
00019
```

## 5.17 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Dio driver/DIO.c File Reference

```
#include "DIO.h"
#include "../Interrupt/Interrupt.h"
```

## Functions

- [EN\\_pinErro\\_t DIO\\_pinInit \(EN\\_pinNum\\_t pinNum, EN\\_pinDirection\\_t pinDirection\)](#)  
*Set the direction of the pin.*
- [EN\\_pinErro\\_t DIO\\_pinWrite \(EN\\_pinNum\\_t pinNum, EN\\_pinState\\_t pinState\)](#)  
*This function writes High or Low on the pin.*
- [EN\\_pinErro\\_t DIO\\_pinRead \(EN\\_pinNum\\_t pinNum, EN\\_pinState\\_t \\*pinState\)](#)  
*This function reads the state of the pin.*
- [EN\\_pinErro\\_t DIO\\_pinToggle \(EN\\_pinNum\\_t pinNum\)](#)  
*This function toggles the state of the pin.*

## 5.18 DIO.c

[Go to the documentation of this file.](#)

```

00001 /*****
00002 /* Author : Ehab Omara
00003 /* Date : 8/10/2022 3:39:46 PM
00004 /* File name: DIO.c
00005 /*
00006 /*****
00007 #include "DIO.h"
00008 #include "../Interrupt/Interrupt.h"
00009
00010
00011
00012 EN_pinErro_t DIO_pinInit(EN_pinNum_t pinNum,EN_pinDirection_t pinDirection)
00013 {
00014 EN_pinErro_t error = OK;
00015 //check if the pin is located in port A
00016 if (pinNum <= PA7)
00017 {
00018 if (pinDirection == Output)
00019 {
00020 setBit(DDRA,pinNum);
00021 }
00022 else if (pinDirection == Input)
00023 {
00024 clrBit(DDRA,pinNum);
00025 }
00026 else
00027 {
00028 error = WRONG_PIN_DIR;
00029 }
00030 }
00031 //check if the pin is located in port B
00032 else if (pinNum <= PB7)
00033 {
00034 pinNum-=PORTB_OFFSET;
00035 if (pinDirection == Output)
00036 {
00037 setBit(DDRB,pinNum);
00038 }
00039 else if (pinDirection == Input)
00040 {
00041 clrBit(DDRB,pinNum);
00042 }
00043 else
00044 {
00045 error = WRONG_PIN_DIR;
00046 }
00047 }
00048 //check if the pin is located in port C
00049 else if (pinNum <= PC7)
00050 {
00051 pinNum-=PORTC_OFFSET;
00052 if (pinDirection == Output)
00053 {
00054 setBit(DDRC,pinNum);
00055 }
00056 else if (pinDirection == Input)

```



```

00057 {
00058 clrBit(DDRC, pinNum);
00059 }
00060 else
00061 {
00062 error = WRONG_PIN_DIR;
00063 }
00064 }
00065 //check if the pin is located in port D
00066 else if (pinNum <= PD7)
00067 {
00068 pinNum-=PORTD_OFFSET;
00069 if (pinDirection == Output)
00070 {
00071 setBit(DDRD, pinNum);
00072 }
00073 else if (pinDirection == Input)
00074 {
00075 clrBit(DDRD, pinNum);
00076 }
00077 else
00078 {
00079 error = WRONG_PIN_DIR;
00080 }
00081 }
00082 //if the pinNum is wrong
00083 else
00084 {
00085 error = WRONG_PIN_NUM;
00086 }
00087 return error;
00088 }
00089
/*****
00090 EN_pinErro_t DIO_pinWrite(EN_pinNum_t pinNum, EN_pinState_t pinState)
00091 {
00092 EN_pinErro_t error = OK;
00093 //check if the pin is located in port A
00094 if (pinNum <= PA7)
00095 {
00096 if (pinState == High)
00097 {
00098 setBit(PORTA, pinNum);
00099 }
00100 else if (pinState == Low)
00101 {
00102 clrBit(PORTA, pinNum);
00103 }
00104 else
00105 {
00106 error = WRONG_PIN_STATE;
00107 }
00108 }
00109 //check if the pin is located in port B
00110 else if (pinNum <= PB7)
00111 {
00112 pinNum-=PORTB_OFFSET;
00113 if (pinState == High)
00114 {
00115 setBit(PORTB, pinNum);
00116 }
00117 else if (pinState == Low)
00118 {
00119 clrBit(PORTB, pinNum);
00120 }
00121 else
00122 {
00123 error = WRONG_PIN_STATE;
00124 }
00125 }
00126 //check if the pin is located in port C
00127 else if (pinNum <= PC7)
00128 {
00129 if (pinState == High)
00130 {
00131 setBit(PORTC, pinNum);
00132 }
00133 else if (pinState == Low)
00134 {
00135 clrBit(PORTC, pinNum);
00136 }
00137 else
00138 {
00139 error = WRONG_PIN_STATE;
00140 }
00141 }
00142 //check if the pin is located in port D

```

```

00143 else if (pinNum <= PD7)
00144 {
00145 if (pinState == High)
00146 {
00147 setBit(PORTD,pinNum);
00148 }
00149 else if (pinState == Low)
00150 {
00151 clrBit(PORTD,pinNum);
00152 }
00153 else
00154 {
00155 error = WRONG_PIN_STATE;
00156 }
00157 }
00158 //if the pinNum is wrong
00159 else
00160 {
00161 error = WRONG_PIN_NUM;
00162 }
00163 return error;
00164 }
00165
00166 EN_pinErro_t DIO_pinRead(EN_pinNum_t pinNum,EN_pinState_t *pinState)
00167 {
00168 EN_pinErro_t error = OK;
00169 //check if the pin is located in port A
00170 if (pinNum <= PA7)
00171 {
00172 *pinState = getBit(PINA,pinNum);
00173 }
00174 //check if the pin is located in port B
00175 else if (pinNum <= PB7)
00176 {
00177 pinNum-=PORTB_OFFSET;
00178 *pinState = getBit(PINB,pinNum);
00179 }
00180 //check if the pin is located in port C
00181 else if (pinNum <= PC7)
00182 {
00183 *pinState = getBit(PINC,pinNum);
00184 }
00185 //check if the pin is located in port D
00186 else if (pinNum <= PD7)
00187 {
00188 *pinState = getBit(PIND,pinNum);
00189 }
00190 //if the pinNum is wrong
00191 else
00192 {
00193 error = WRONG_PIN_NUM;
00194 }
00195 return error;
00196 }
00197
00198 EN_pinErro_t DIO_pinToggle(EN_pinNum_t pinNum)
00199 {
00200 EN_pinErro_t error = OK;
00201 //check if the pin is located in port A
00202 if (pinNum <= PA7)
00203 {
00204 toggleBit(PORTA,pinNum);
00205 }
00206 //check if the pin is located in port B
00207 else if (pinNum <= PB7)
00208 {
00209 pinNum-=PORTB_OFFSET;
00210 toggleBit(PORTB,pinNum);
00211 }
00212 //check if the pin is located in port C
00213 else if (pinNum <= PC7)
00214 {
00215 toggleBit(PORTC,pinNum);
00216 }
00217 //check if the pin is located in port D
00218 else if (pinNum <= PD7)
00219 {
00220 toggleBit(PORTD,pinNum);
00221 }
00222 //if the pinNum is wrong
00223 else
00224 {
00225 error = WRONG_PIN_NUM;
00226 }
00227 return error;

```

```
00228 }
00229
```

```
/* ****
```

## 5.19 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Dio driver/DIO.h File Reference

```
#include "../Service/ATmega32Port.h"
#include "../Service/BitMath.h"
#include "../Service/dataTypes.h"
#include "../Service/RegisterFile.h"
```

### Functions

- [EN\\_pinErro\\_t DIO\\_pinInit \(EN\\_pinNum\\_t pinNum, EN\\_pinDirection\\_t pinDirection\)](#)  
*Set the direction of the pin.*
- [EN\\_pinErro\\_t DIO\\_pinWrite \(EN\\_pinNum\\_t pinNum, EN\\_pinState\\_t pinState\)](#)  
*This function writes High or Low on the pin.*
- [EN\\_pinErro\\_t DIO\\_pinToggle \(EN\\_pinNum\\_t pinNum\)](#)  
*This function toggles the state of the pin.*
- [EN\\_pinErro\\_t DIO\\_pinRead \(EN\\_pinNum\\_t pinNum, EN\\_pinState\\_t \\*pinState\)](#)  
*This function reads the state of the pin.*

### 5.19.1 Detailed Description

#### Author

: Ehab Omara

#### Date

: 8/10/2022 3:39:36 PM

Definition in file [DIO.h](#).

## 5.20 DIO.h

[Go to the documentation of this file.](#)

```
00001
```

```
/* ****
```

```
00007 #ifndef DIO_H_
00008 #define DIO_H_
00009
00010 #include "../Service/ATmega32Port.h"
00011 #include "../Service/BitMath.h"
00012 #include "../Service/dataTypes.h"
00013 #include "../Service/RegisterFile.h"
00040 EN_pinErro_t DIO_pinInit (EN_pinNum_t pinNum, EN_pinDirection_t pinDirection);
00058 EN_pinErro_t DIO_pinWrite (EN_pinNum_t pinNum, EN_pinState_t pinState);
00072 EN_pinErro_t DIO_pinToggle (EN_pinNum_t pinNum);
00086 EN_pinErro_t DIO_pinRead (EN_pinNum_t pinNum, EN_pinState_t *pinState);
00090 #endif /* DIO_H_ */
```

## 5.21 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Ext interrupt driver/Ext interrupt.c File Reference

```
#include "Ext interrupt.h"
```

### Functions

- [EN\\_interruptError\\_t Ext\\_interruptInit \(EN\\_interruptNum\\_t interruptNum, EN\\_interruptSenseControl\\_t interruptSenseControl\)](#)

*External interrupt init.*

## 5.22 Ext interrupt.c

[Go to the documentation of this file.](#)

```
00001
00002 /***** Author : Ehab Omara *****/
00003 /***** Date : 8/13/2022 4:40:08 AM *****/
00004 /***** File name: Ext interrupt.c *****/
00005 /*****/
00006
00007
00008 #include "Ext interrupt.h"
00009 EN_interruptError_t Ext_interruptInit (EN_interruptNum_t interruptNum, EN_interruptSenseControl_t
00010 interruptSenseControl)
00011 {
00012 EN_interruptError_t interruptError = INT_OK;
00013 if (interruptNum == INT0)
00014 {
00015 //check if the value of the interruptSenseControl is correct
00016 if (interruptSenseControl >= LOW_LEVEL && interruptSenseControl <= RISING_EDGE)
00017 {
00018 //enable INT0
00019 setBit(GICR, INT0);
00020 //clearing interruptSenseControl old value
00021 MCUCR &= (~ (ISC00 << 0x03));
00022 //setting interruptSenseControl new value
00023 MCUCR |= interruptSenseControl << ISC00;
00024 //set INT0 pin as input
00025 clrBit(DDRD, INT0_PIN);
00026 }
00027 else
00028 {
00029 interruptError = WRONG_SENSE_CONTROL;
00030 }
00031 }
00032 else if (interruptNum == INT1)
00033 {
00034 //check if the value of the interruptSenseControl is correct
00035 if (interruptSenseControl >= LOW_LEVEL && interruptSenseControl <= RISING_EDGE)
00036 {
00037 //enable INT1
00038 setBit(GICR, INT1);
00039 //clearing interruptSenseControl old value
00040 MCUCR &= (~ (0x03 << ISC10));
00041 //setting interruptSenseControl new value
00042 MCUCR |= interruptSenseControl << ISC10;
00043 //set INT1 pin as input
00044 clrBit(DDRD, INT1_PIN);
00045 }
00046 else
00047 {
00048 interruptError = WRONG_SENSE_CONTROL;
00049 }
00050 }
00051 }
```

```

00049 }
00050 }
00051 else if (interruptNum == INT2)
00052 {
00053 //check if the value of the interruptSenseControl is correct
00054 if (interruptSenseControl == FALLING_EDGE)
00055 {
00056 //enable INT1
00057 setBit(GICR,INT2);
00058 clrBit(MCUCSR,ISC2);
00059 //set INT2 pin as input
00060 clrBit(DDRB,INT2_PIN);
00061 }
00062 else if(interruptSenseControl == RISING_EDGE)
00063 {
00064 //enable INT1
00065 setBit(GICR,INT2);
00066 setBit(MCUCSR,ISC2);
00067 //set INT2 pin as input
00068 clrBit(DDRB,INT2_PIN);
00069 }
00070 }
00071 else
00072 {
00073 interruptError = WRONG_SENSE_CONTROL;
00074 }
00075 }
00076 else
00077 {
00078 interruptError = WRONG_INT_NUM;
00079 }
00080 if (interruptError == INT_OK)
00081 {
00082 //enable global interrupt
00083 sei();
00084 }
00085 return interruptError;
00086 }
00087
00088

```

## 5.23 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Ext interrupt driver/Ext interrupt.h File Reference

```

#include "../Service/ATmega32Port.h"
#include "../Service/RegisterFile.h"
#include "../Interrupt/Interrupt.h"
#include "../Service/BitMath.h"

```

### Macros

#### External interrupts pins

- These are the pins which connected to each interrupt.
- It should be configured as *Input*.
- #define INT0\_PIN (PD2 - PORTD\_OFFSET)
- #define INT1\_PIN (PD3 - PORTD\_OFFSET)
- #define INT2\_PIN (PB2 - PORTB\_OFFSET)

#### INT0 sense control

- These two bits *ISC00* and *ISC01* which located in *MCUCR* register control the *INT0* sense control.

| ISC01 | ISC00 | Description                                                |
|-------|-------|------------------------------------------------------------|
| 0     | 0     | The low level of INT0 generates an interrupt request.      |
| 0     | 1     | Any logical change on INT0 generates an interrupt request. |
| 1     | 0     | The falling edge of INT0 generates an interrupt request.   |
| 1     | 1     | The rising edge of INT0 generates an interrupt request.    |

- #define ISC00 0
- #define ISC01 1

### INT1 sense control

- These two bits ISC10 and ISC11 which located in MCUCR register control the INT1 sense control.

| ISC11 | ISC10 | Description                                                |
|-------|-------|------------------------------------------------------------|
| 0     | 0     | The low level of INT1 generates an interrupt request.      |
| 0     | 1     | Any logical change on INT1 generates an interrupt request. |
| 1     | 0     | The falling edge of INT1 generates an interrupt request.   |
| 1     | 1     | The rising edge of INT1 generates an interrupt request.    |

- #define ISC10 2
- #define ISC11 3

### INT2 sense control

- This bit ISC2 which located in MCUCSR register control the INT2 sense control.

| ISC2 | Description                                               |
|------|-----------------------------------------------------------|
| 0    | The falling edge on INT2 activates the interrupt request. |
| 1    | The rising edge on INT2 activates the interrupt request.  |

- #define ISC2 6

## Enumerations

- enum EN\_interruptNum\_t { INT2 = 5, INT0, INT1 }  
External interrupt number.
- enum EN\_interruptSenseControl\_t { LOW\_LEVEL, ANY\_LOGICAL\_CHANGE, FALLING\_EDGE, RISING\_EDGE }  
External interrupt sense control.
- enum EN\_interruptError\_t { INT\_OK, WRONG\_INT\_NUM, WRONG\_SENSE\_CONTROL }  
External interrupt errors.

## Functions

- EN\_interruptError\_t Ext\_interruptInit (EN\_interruptNum\_t interruptNum, EN\_interruptSenseControl\_t interruptSenseControl)  
External interrupt init.

## 5.24 Ext interrupt.h

[Go to the documentation of this file.](#)

```

00001
00002 /*****
00003 */
00004 */
00005
00006 /*****
00007 #ifndef EXT_INTERRUPT_H_
00008 #define EXT_INTERRUPT_H_
00009
00010 #include "../Service/ATmega32Port.h"
00011 #include "../Service/RegisterFile.h"
00012 #include "../Interrupt/Interrupt.h"
00013 #include "../Service/BitMath.h"
00014
00029 #define INT0_PIN (PD2 - PORTD_OFFSET)
00030 #define INT1_PIN (PD3 - PORTD_OFFSET)
00031 #define INT2_PIN (PB2 - PORTB_OFFSET)
00033
00046 #define ISC00 0
00047 #define ISC01 1
00049
00050 /*****
00063 #define ISC10 2
00064 #define ISC11 3
00066
00067 /*****
00068
00078 #define ISC2 6
00080
00081 /*****
00089 typedef enum
00090 {
00091 INT2 = 5,
00092 INT0,
00093 INT1
00094 }EN_interruptNum_t;
00095
00104 typedef enum
00105 {
00106 LOW_LEVEL,
00107 ANY_LOGICAL_CHANGE,
00108 FALLING_EDGE,
00109 RISING_EDGE
00110 }EN_interruptSenseControl_t;
00111
00117 typedef enum
00118 {
00119 INT_OK,
00120 WRONG_INT_NUM,
00121 WRONG_SENSE_CONTROL
00122 }EN_interruptError_t;
00123
00136 EN_interruptError_t Ext_interruptInit (EN_interruptNum_t interruptNum,EN_interruptSenseControl_t
interruptSenseControl);
00138 #endif /* EXT_INTERRUPT_H_ */

```

## 5.25 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Interrupt/Interrupt.h File Reference

### Macros

- #define sei() \_\_asm\_\_ \_\_volatile\_\_ ("sei" ::: "memory")

- #define cli() \_\_asm\_\_ \_\_volatile\_\_ ("cli" ::: "memory")
- #define EXT\_INT0 \_\_vector\_1
- #define EXT\_INT1 \_\_vector\_2
- #define EXT\_INT2 \_\_vector\_3
- #define TIM2\_COMP \_\_vector\_4
- #define TIM2\_OVF \_\_vector\_5
- #define TIM1\_CAPT \_\_vector\_6
- #define TIM1\_COMPA \_\_vector\_7
- #define TIM1\_COMPB \_\_vector\_8
- #define TIM1\_OVF \_\_vector\_9
- #define TIM0\_COMP \_\_vector\_10
- #define TIM0\_OVF \_\_vector\_11
- #define SPI\_STC \_\_vector\_12
- #define USART\_RXC \_\_vector\_13
- #define USART\_UDRE \_\_vector\_14
- #define USART\_TXC \_\_vector\_15
- #define ADC \_\_vector\_16
- #define EE\_RDY \_\_vector\_17
- #define ANA\_COMP \_\_vector\_18
- #define TWI \_\_vector\_19
- #define SPM\_RDY \_\_vector\_20
- #define ISR(INT\_VECT)

*interrupt service routine Macro.*

## 5.26 Interrupt.h

[Go to the documentation of this file.](#)

```

00001
00002 /*****
00003 */
00004 */
00005 /*****
00006
00007 #ifndef INTERRUPT_H_
00008 #define INTERRUPT_H_
00035 # define sei() __asm__ __volatile__ ("sei" ::: "memory")
00036
00046 # define cli() __asm__ __volatile__ ("cli" ::: "memory")
00047
00048 #define EXT_INT0 __vector_1
00049 #define EXT_INT1 __vector_2
00050 #define EXT_INT2 __vector_3
00051 #define TIM2_COMP __vector_4
00052 #define TIM2_OVF __vector_5
00053 #define TIM1_CAPT __vector_6
00054 #define TIM1_COMPA __vector_7
00055 #define TIM1_COMPB __vector_8
00056 #define TIM1_OVF __vector_9
00057 #define TIM0_COMP __vector_10
00058 #define TIM0_OVF __vector_11
00059 #define SPI_STC __vector_12
00060 #define USART_RXC __vector_13
00061 #define USART_UDRE __vector_14
00062 #define USART_TXC __vector_15
00063 #define ADC __vector_16
00064 #define EE_RDY __vector_17
00065 #define ANA_COMP __vector_18
00066 #define TWI __vector_19
00067 #define SPM_RDY __vector_20
00078 #define ISR(INT_VECT) void INT_VECT(void) __attribute__((signal,used));\
00079 void INT_VECT(void)
00082 #endif /* INTERRUPT_H_ */

```



## 5.27 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Timer driver/Timer\_0.c File Reference

```
#include "Timer_0.h"
#include <math.h>
#include "../Interrupt/Interrupt.h"
```

### Functions

- [En\\_Timer0\\_Error\\_t](#) [Timer0\\_init](#) ([EN\\_Timer0\\_Mode\\_t](#) [Timer0\\_mode](#), [EN\\_Timer0\\_clkSource\\_t](#) [Timer0\\_clkSource](#))
- void [Timer0\\_start](#) (void)
- void [Timer0\\_stop](#) (void)
- void [Timer0\\_reset](#) (void)
- [En\\_Timer0\\_Error\\_t](#) [Timer0\\_interruptDisable](#) ([TIMER0\\_interrupt\\_t](#) [Timer0\\_interrupt](#))
- [En\\_Timer0\\_Error\\_t](#) [Timer0\\_interruptEnable](#) ([TIMER0\\_interrupt\\_t](#) [Timer0\\_interrupt](#))
- void [Timer0\\_delay\\_ms](#) ([uint32\\_t](#) [delay\\_ms](#))
- [ISR](#) ([TIM0\\_OVF](#))

### Variables

- static [EN\\_Timer0\\_clkSource\\_t](#) [Timer0\\_globalClkSource](#) = [clk\\_No\\_DIVISON](#)  
*Global static variable for Timer 0 clock source.*
- static [uint32\\_t](#) volatile [Timer0\\_globalNumOfOverFlows](#) = 0  
*Global static variable for Timer 0 over flows number.*
- static [float64\\_t](#) volatile [Timer0\\_globalOverflowTime](#) = 0  
*Global static variable for Timer 0 over flow time.*
- static [float64\\_t](#) volatile [Timer0\\_globalTickTime](#) = 0  
*Global static variable for Timer 0 tick time.*

### 5.27.1 Function Documentation

#### 5.27.1.1 [ISR\(\)](#)

```
ISR (
 TIM0_OVF)
```

Definition at line 147 of file [Timer\\_0.c](#).

### 5.27.2 Variable Documentation

#### 5.27.2.1 Timer0\_globalClkSource

```
EN_Timer0_clkSource_t Timer0_globalClkSource = clkI_No_DIVISON [static]
```

Global static variable for Timer 0 clock source.

- This variable stores the value of the clock source for Timer 0.

Definition at line 15 of file [Timer\\_0.c](#).

#### 5.27.2.2 Timer0\_globalNumOfOverFlows

```
uint32_t Timer0_globalNumOfOverFlows = 0 [static]
```

Global static variable for Timer 0 over flows number.

- This variable stores the number of over flows of the clock source for Timer 0.
- This variable declared as volatile to prevent the compiler from deleting it as it will be used by [ISR](#).

Definition at line 25 of file [Timer\\_0.c](#).

#### 5.27.2.3 Timer0\_globalOverflowTime

```
float64_t volatile Timer0_globalOverflowTime = 0 [static]
```

Global static variable for Timer 0 over flow time.

- This variable stores the time for one over flow.

Definition at line 33 of file [Timer\\_0.c](#).

#### 5.27.2.4 Timer0\_globalTickTime

```
float64_t volatile Timer0_globalTickTime = 0 [static]
```

Global static variable for Timer 0 tick time.

- This variable stores the time for one tick.

Definition at line 40 of file [Timer\\_0.c](#).

## 5.28 Timer\_0.c

[Go to the documentation of this file.](#)

```

00001
00002 /* Author : Ehab Omara
00003 */
00004 /* Date : 8/14/2022 12:55:03 PM
00005 */
00006 /* File name: Timer_0.c
00007 */
00008
00009 /******
00010 #include "Timer_0.h"
00011 #include <math.h>
00012 #include "../Interrupt/Interrupt.h"
00013 static EN_Timer0_clkSource_t Timer0_globalClkSource = clkI_No_DIVISON;
00014
00015 /******
00016 static uint32_t volatile Timer0_globalNumOfOverFlows = 0;
00017
00018 /******
00019 static float64_t volatile Timer0_globalOverflowTime = 0;
00020 static float64_t volatile Timer0_globalTickTime = 0;
00021
00022 /******
00023 En_Timer0_Error_t Timer0_init(EN_Timer0_Mode_t Timer0_mode, EN_Timer0_clkSource_t Timer0_clkSource)
00024 {
00025 En_Timer0_Error_t Timer0_error = TIMER0_OK;
00026 //selecting Timer 0 mode
00027 if (Timer0_mode == NORMAL || Timer0_mode == PWM_PHASE_CORRECR || Timer0_mode == CTC || Timer0_mode
00028 == FAST_PWM)
00029 {
00030 //clear the old mode value
00031 TCCR0 &= CLR_TIMER0_MODE;
00032 //set the new mode value
00033 TCCR0 |= Timer0_mode;
00034 }
00035 else
00036 {
00037 Timer0_error = TIMER0_WRONG_MODE;
00038 }
00039 //selecting Timer 0 clock source
00040 if (Timer0_clkSource >= NO_CLOCK_SOURCE && Timer0_clkSource <= EXTERNAL_CLOCK_RISING_EDGE)
00041 {
00042 Timer0_globalClkSource = Timer0_clkSource;
00043
00044 uint16_t Timer0_localClkPrescaler[] = {0,1,8,64,256,1024};
00045 //calculate Timer 0 tick time
00046 Timer0_globalTickTime = (float64_t)Timer0_localClkPrescaler[Timer0_clkSource]/SYSTEM_CLK;
00047 //calculate Timer 0 over flow time
00048 Timer0_globalOverflowTime = Timer0_globalTickTime * TIMER0_NUM_OF_TICKS;
00049 }
00050 else
00051 {
00052 Timer0_error = TIMER0_WRONG_CLK_SOURCE;
00053 }
00054 return Timer0_error;
00055 }
00056
00057 /******
00058 void Timer0_start(void)
00059 {
00060 //clear the old clock source value
00061 TCCR0 &= CLR_TIMER0_CLK_SRC;
00062 //set the new clock source value
00063 TCCR0 |= Timer0_globalClkSource << CS00;
00064 }
00065
00066 /******
00067 void Timer0_stop(void)
00068 {
00069 //clear the value of Timer 0 clock source
00070 //this is done by clearing the three bits #CS00, #CS01 and #CS02
00071 TCCR0 &= CLR_TIMER0_CLK_SRC;
00072 }
00073
00074 /******
00075 void Timer0_reset(void)
00076 {
00077 TCNT0 = 0x00;
00078 Timer0_globalNumOfOverFlows = 0;
00079 }

```

```

00097
00098 /*****
00099 En_Timer0_Error_t Timer0_interruptDiable(TIMER0_interrupt_t Timer0_interrupt)
00100 {
00101 En_Timer0_Error_t Timer0_error = TIMER0_OK;
00102 if (Timer0_interrupt == TIMER0_OVER_FLOW_INT || Timer0_interrupt == TIMER0_OUT_CMP_MATCH_INT)
00103 {
00104 clrBit(TIMSK,Timer0_interrupt);
00105 }
00106 else
00107 {
00108 Timer0_error = TIMER0_WRONG_INT;
00109 }
00110 return Timer0_error;
00111 }
00112
00112 /*****
00113 En_Timer0_Error_t Timer0_interruptEnable(TIMER0_interrupt_t Timer0_interrupt)
00114 {
00115 En_Timer0_Error_t Timer0_error = TIMER0_OK;
00116 if (Timer0_interrupt == TIMER0_OVER_FLOW_INT || Timer0_interrupt == TIMER0_OUT_CMP_MATCH_INT)
00117 {
00118 sei();
00119 setBit(TIMSK,Timer0_interrupt);
00120 }
00121 else
00122 {
00123 Timer0_error = TIMER0_WRONG_INT;
00124 }
00125 return Timer0_error;
00126 }
00127
00127 /*****
00128 void Timer0_delay_ms(uint32_t delay_ms)
00129 {
00130 //reset Timer 0
00131 Timer0_reset();
00132 //convert delay time from mile seconds to seconds
00133 float64_t neededTimeInsecond = (float64_t)delay_ms/1000;
00134 //calculate number of over flows needed to reach the desired time
00135 uint32_t numberOfoverFlows = ceil(neededTimeInsecond/Timer0_globalOverFlowTime);
00136 //calculate the initial value for #TCNT0 register
00137 TCNT0 = TIMER0_NUM_OF_TICKS - (neededTimeInsecond/Timer0_globalTickTime)/numberOfoverFlows;
00138 //enable Timer 0 over flow interrupt
00139 Timer0_interruptEnable(TIMER0_OVER_FLOW_INT);
00140 //start Timer 0 to count
00141 Timer0_start();
00142 //wait until reaching needed number over flows
00143 while(Timer0_globalNumOfOverFlows < numberOfoverFlows);
00144 //stop Timer 0 after reaching the desired time.
00145 Timer0_stop();
00146 }
00147
00147 ISR(TIM0_OVF)
00148 {
00149 Timer0_globalNumOfOverFlows++;
00150 }

```

## 5.29 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Timer driver/Timer\_0.h File Reference

```

#include "../Service/BitMath.h"
#include "../Service/ATmega32Port.h"
#include "../Service/dataTypes.h"
#include "../Service/RegisterFile.h"

```

### Macros

- `#define TIMER0_NUM_OF_TICKS 256`  
System clock Macro.

- #define [CLR\\_TIMER0\\_CLK\\_SRC](#) 0xF8
- #define [CLR\\_TIMER0\\_MODE](#) 0xB7

#### Bit 2:0 - CS02:0: Clock Select

- The three Clock Select bits select the clock source to be used by the Timer/Counter and located in [TCCR0](#).

| CS02                                | CS01 | CS00 | Description                                             |
|-------------------------------------|------|------|---------------------------------------------------------|
| 0                                   | 0    | 0    | No clock source (Timer/Counter stopped).                |
| 0                                   | 0    | 1    | clkI/O/(No prescaling).                                 |
| 0                                   | 1    | 0    | clkI/O/8 (From prescaler).                              |
| 0                                   | 1    | 1    | clkI/O/64 (From prescaler).                             |
| 1                                   | 0    | 0    | clkI/O/256 (From prescaler).                            |
| 1                                   | 0    | 1    | clkI/O/1024 (From prescaler).                           |
| 1                                   | 1    | 0    | External clock source on T0 pin. Clock on falling edge. |
| 1                                   | 1    | 1    | External clock source on T0 pin. Clock on rising edge.  |
| <b>Clock Select Bit Description</b> |      |      |                                                         |

#### Note

If external pin modes are used for the Timer/Counter0, transitions on the T0 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

- #define [CS00](#) 0
- #define [CS01](#) 1
- #define [CS02](#) 2

#### Bit 6, 3 - WGM01:0: Waveform Generation Mode

- These bits control the counting sequence of the counter and located in [TCCR0](#).
- the source for the maximum (TOP) counter value, and what type of Waveform Generation to be used.

| Mode                                            | WGM01 (CTC0) | WGM00 (PWM0) | Timer/Counter Mode of Operation | TOP                  | Update of OCR0 | TOV0 Flag Set-on |
|-------------------------------------------------|--------------|--------------|---------------------------------|----------------------|----------------|------------------|
| 0                                               | 0            | 0            | Normal                          | 0xFF                 | Immediate      | MAX              |
| 1                                               | 0            | 1            | PWM, Phase Correct              | 0xFF                 | TOP            | BOTTOM           |
| 2                                               | 1            | 0            | CTC                             | <a href="#">OCR0</a> | Immediate      | MAX              |
| 3                                               | 1            | 1            | Fast PWM                        | 0xFF                 | BOTTOM         | MAX              |
| <b>Waveform Generation Mode Bit Description</b> |              |              |                                 |                      |                |                  |

- #define [WGM00](#) 3
- #define [WGM01](#) 6

#### Timer/Counter0 Interrupts Flags

- These bits are flags for interrupts of the Timer 0 and located in [TIFR](#).
- #define [TOV0](#) 0
- #define [OCF0](#) 1

## Enumerations

- enum `En_Timer0_Mode_t` { `NORMAL` = 0 , `PWM_PHASE_CORRECR` =8 , `CTC` =64 , `FAST_PWM` =72 }
- enum `En_Timer0_clkSource_t` {  
`NO_CLOCK_SOURCE` , `clk_No_DIVISON` , `clk_DIVISION_BY_8` , `clk_DIVISION_BY_64` ,  
`clk_DIVISION_BY_256` , `clk_DIVISION_BY_1024` , `EXTERNAL_CLOCK_FALLING_EDGE` , `EXTERNAL_CLOCK_RISING_E`  
}
- enum `En_Timer0_Error_t` { `TIMER0_OK` , `TIMER0_WRONG_MODE` , `TIMER0_WRONG_CLK_SOURCE` ,  
`TIMER0_WRONG_INT` }

## Functions

- `En_Timer0_Error_t` `Timer0_interruptEnable` (`TIMER0_interrupt_t` `Timer0_interrupt`)
- `En_Timer0_Error_t` `Timer0_interruptDiable` (`TIMER0_interrupt_t` `Timer0_interrupt`)
- `En_Timer0_Error_t` `Timer0_init` (`En_Timer0_Mode_t` `Timer0_mode`, `En_Timer0_clkSource_t` `Timer0_clkSource`)
- void `Timer0_start` (void)
- void `Timer0_stop` (void)
- void `Timer0_reset` (void)
- void `Timer0_delay_ms` (`uint32_t` `delay_ms`)

## Timer/Counter0 Interrupts Enable

- These bits enable and disable the interrupts of the counter and located in `TIMSK`.
- `#define TOIE0` 0
- `#define OCIE0` 1
- enum `TIMER0_interrupt_t` { `TIMER0_OVER_FLOW_INT` , `TIMER0_OUT_CMP_MATCH_INT` }

## 5.30 Timer\_0.h

[Go to the documentation of this file.](#)

```

00001
00002 /*
00003 */
00004 /*
00005 */
00006
00007 #ifndef TIMER_0_H_
00008 #define TIMER_0_H_
00009 #include "../Service/BitMath.h"
00010 #include "../Service/ATmega32Port.h"
00011 #include "../Service/dataTypes.h"
00012 #include "../Service/RegisterFile.h"
00013
00014
00015
00016
00017
00018
00019
00020
00021
00022
00023
00024
00025
00026
00027
00028
00029
00030
00031
00032
00033 #ifndef SYSTEM_CLK
00034 /* prevent compiler error by supplying a default */
00035 # warning "SYSTEM_CLK not defined for Timer_0.h, default value is 1MHz"
00036 #define SYSTEM_CLK 1000000UL
00037 #endif
00038
00039
00040
00041
00042
00043
00044
00045
00046 #define TIMER0_NUM_OF_TICKS 256

```

```

00047
00068 /*****
00069 #define CS00 0
00070 #define CS01 1
00071 #define CS02 2
00072
00080 /*****
00081 #define CLR_TIMER0_CLK_SRC 0xF8
00082
00096 /*****
00097 #define WGM00 3
00098 #define WGM01 6
00099
00107 /*****
00108 #define CLR_TIMER0_MODE 0xB7
00109
00115 /*****
00116 #define TOIE0 0
00117 #define OCIE0 1
00122 typedef enum
00123 {
00124 TIMER0_OVER_FLOW_INT,
00125 TIMER0_OUT_CMP_MATCH_INT
00126 }TIMER0_interrupt_t;
00127
00135 /*****
00136 #define TOV0 0
00137 #define OCF0 1
00138
00170 /*****
00171 typedef enum
00172 {
00173 NORMAL = 0,
00174 PWM_PHASE_CORRECR=8,
00175 CTC=64,
00176 FAST_PWM=72
00177 }EN_Timer0_Mode_t;
00178
00187 /*****
00188 typedef enum
00189 {
00190 NO_CLOCK_SOURCE,
00191 clkI_No_DIVISON,
00192 clkI_DIVISION_BY_8,
00193 clkI_DIVISION_BY_64,
00194 clkI_DIVISION_BY_256,
00195 clkI_DIVISION_BY_1024,
00196 EXTERNAL_CLOCK_FALLING_EDGE,
00197 EXTERNAL_CLOCK_RISING_EDGE
00198 }EN_Timer0_clkSource_t;
00199
00204 /*****
00205 typedef enum
00206 {
00207 TIMER0_OK,
00208 TIMER0_WRONG_MODE,
00209 TIMER0_WRONG_CLK_SOURCE,
00210 TIMER0_WRONG_INT
00211 }En_Timer0_Error_t;
00212
00223 /*****
00224 En_Timer0_Error_t Timer0_interruptEnable(TIMER0_interrupt_t Timer0_interrupt);
00225
00236 /*****
00237 En_Timer0_Error_t Timer0_interruptDiablo(TIMER0_interrupt_t Timer0_interrupt);
00238
00253 /*****
00254 En_Timer0_Error_t Timer0_init(EN_Timer0_Mode_t Timer0_mode,EN_Timer0_clkSource_t Timer0_clkSource);
00255
00263 /*****
00264 void Timer0_start(void);
00265
00273 /*****
00274 void Timer0_stop(void);
00275
00283 /*****
00284 void Timer0_reset(void);
00285
00293 /*****
00294 void Timer0_delay_ms(uint32_t delay_ms);
00295 #endif /* TIMER_0_H_ */

```

## 5.31 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/ATmega32Port.h File Reference

### Macros

- #define [PORTA\\_OFFSET](#) 0
- #define [PORTB\\_OFFSET](#) 8
- #define [PORTC\\_OFFSET](#) 16
- #define [PORTD\\_OFFSET](#) 24

### Enumerations

- enum [EN\\_pinNum\\_t](#) {  
[PA0](#) , [PA1](#) , [PA2](#) , [PA3](#) ,  
[PA4](#) , [PA5](#) , [PA6](#) , [PA7](#) ,  
[PB0](#) , [PB1](#) , [PB2](#) , [PB3](#) ,  
[PB4](#) , [PB5](#) , [PB6](#) , [PB7](#) ,  
[PC0](#) , [PC1](#) , [PC2](#) , [PC3](#) ,  
[PC4](#) , [PC5](#) , [PC6](#) , [PC7](#) ,  
[PD0](#) , [PD1](#) , [PD2](#) , [PD3](#) ,  
[PD4](#) , [PD5](#) , [PD6](#) , [PD7](#) }
- enum [EN\\_pinState\\_t](#) { [Low](#) , [High](#) }
- enum [EN\\_pinDirection\\_t](#) { [Input](#) , [Output](#) }
- enum [EN\\_pinErro\\_t](#) { [OK](#) , [WRONG\\_PIN\\_NUM](#) , [WRONG\\_PIN\\_DIR](#) , [WRONG\\_PIN\\_STATE](#) }

## 5.32 ATmega32Port.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002 /* Author : Ehab Omara
00003 /* Date : 8/10/2022 3:49:55 PM
00004 /* File name: ATmega32Port.h
00005 /*
00006 /*****
00007 #ifndef ATMEGA32PORT_H_
00008 #define ATMEGA32PORT_H_
00009
00010
00022 typedef enum
00023 {
00024 /*PORTA pins*/
00025 PA0,
00026 PA1,
00027 PA2,
00028 PA3,
00029 PA4,
00030 PA5,
00031 PA6,
00032 PA7,
00033 /*PORTB pins*/
00034 PB0,
00035 PB1,
00036 PB2,
00037 PB3,
00038 PB4,
00039 PB5,
00040 PB6,
00041 PB7,

```



```
00042 /*PORTC pins*/
00043 PC0,
00044 PC1,
00045 PC2,
00046 PC3,
00047 PC4,
00048 PC5,
00049 PC6,
00050 PC7,
00051 /*PORTD pins*/
00052 PD0,
00053 PD1,
00054 PD2,
00055 PD3,
00056 PD4,
00057 PD5,
00058 PD6,
00059 PD7
00060 }EN_pinNum_t;
00061
00062 #define PORTA_OFFSET 0
00063 #define PORTB_OFFSET 8
00064 #define PORTC_OFFSET 16
00065 #define PORTD_OFFSET 24
00066 typedef enum
00067 {
00068 Low,
00069 High
00070 }EN_pinState_t;
00071 typedef enum
00072 {
00073 Input,
00074 Output
00075 }EN_pinDirection_t;
00076 typedef enum
00077 {
00078 OK,
00079 WRONG_PIN_NUM,
00080 WRONG_PIN_DIR,
00081 WRONG_PIN_STATE
00082 }EN_pinError_t;
00083 #endif /* ATMEGA32PORT_H_ */
```

## 5.33 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/BitMath.h File Reference

### Macros

- #define [setBit](#)(reg, bitNum) reg |= (1<<bitNum)  
*this Macro writes 1 to the bit.*
- #define [clrBit](#)(reg, bitNum) reg &= (~(1<<bitNum))  
*this Macro clear the bit.*
- #define [toggleBit](#)(reg, bitNum) reg ^= (1<<bitNum)  
*This Macro toggle the bit logic.*
- #define [getBit](#)(reg, bitNum) ((reg>>bitNum) & 0x01)  
*This Macro read this bit value.*

## 5.34 BitMath.h

[Go to the documentation of this file.](#)

```
00001
00002 /******
00003
00007 #ifndef BITMATH_H_
00008 #define BITMATH_H_
00009
00026 #define setBit(reg,bitNum) reg |= (1<<bitNum)
00037 #define clrBit(reg,bitNum) reg &= ~(1<<bitNum)
00050 #define toggleBit(reg,bitNum) reg ^= (1<<bitNum)
00062 #define getBit(reg,bitNum) ((reg>>bitNum) & 0x01)
00066 #endif /* BITMATH_H_ */
```

## 5.35 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/dataTypes.h File Reference

### Typedefs

- typedef unsigned char [uint8\\_t](#)
- typedef signed char [sint8\\_t](#)
- typedef unsigned short int [uint16\\_t](#)
- typedef signed short int [sint16\\_t](#)
- typedef unsigned long int [uint32\\_t](#)
- typedef signed long int [sint32\\_t](#)
- typedef float [float32\\_t](#)
- typedef double [float64\\_t](#)
- typedef long double [float128\\_t](#)

## 5.36 dataType.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002 */
00003 */
00004 */
00005 /*****
00006 #ifndef DATATYPES_H_
00007 #define DATATYPES_H_
00015 typedef unsigned char uint8_t;
00016 typedef signed char sint8_t;
00017 typedef unsigned short int uint16_t;
00018 typedef signed short int sint16_t;
00019 typedef unsigned long int uint32_t;
00020 typedef signed long int sint32_t;
00021 typedef float float32_t;
00022 typedef double float64_t;
00023 typedef long double float128_t;
00027 #endif /* DATATYPES_H_ */

```

## 5.37 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/RegisterFile.h File Reference

```
#include "dataTypes.h"
```

## Macros

- `#define PORTA (*((volatile uint8_t*)0x3B))`
- `#define DDRA (*((volatile uint8_t*)0x3A))`
- `#define PINA (*((volatile uint8_t*)0x39))`
- `#define PORTB (*((volatile uint8_t*)0x38))`
- `#define DDRB (*((volatile uint8_t*)0x37))`
- `#define PINB (*((volatile uint8_t*)0x36))`
- `#define PORTC (*((volatile uint8_t*)0x35))`
- `#define DDRC (*((volatile uint8_t*)0x34))`
- `#define PINC (*((volatile uint8_t*)0x33))`
- `#define PORTD (*((volatile uint8_t*)0x32))`
- `#define DDRD (*((volatile uint8_t*)0x31))`
- `#define PIND (*((volatile uint8_t*)0x30))`
- `#define GICR (*((volatile uint8_t*)0x5B))`
- `#define GIFR (*((volatile uint8_t*)0x5A))`
- `#define MCUCR (*((volatile uint8_t*)0x55))`
- `#define MCUCSR (*((volatile uint8_t*)0x54))`
- `#define TCCR0 (*((volatile uint8_t*)0x53))`
- `#define TCNT0 (*((volatile uint8_t*)0x52))`
- `#define OCR0 (*((volatile uint8_t*)0x5C))`
- `#define TIFR (*((volatile uint8_t*)0x58))`
- `#define TIMSK (*((volatile uint8_t*)0x59))`

## 5.38 RegisterFile.h

[Go to the documentation of this file.](#)

```
00001
00002 /*****
00003 */
00004 */
00005
00006
00007 #ifndef REGISTERFILE_H_
00008 #define REGISTERFILE_H_
00009
00010 #include "dataTypes.h"
00011 */
00012 * if the DDRx is set to be output and we write High to the PORTx
00013 * this will activate the internal Pull up resistor.
00014 */
00015
00016 /***** Port A registers
00017 *****/
00060 #define PORTA (*((volatile uint8_t*)0x3B)) //1->high output 0->low output
00068 #define DDRA (*((volatile uint8_t*)0x3A)) //1->to make it output 0->to make it input
00076 #define PINA (*((volatile uint8_t*)0x39)) //this register to read a value from a pin
00080 /***** Port B registers
00081 *****/
00094 #define PORTB (*((volatile uint8_t*)0x38))
00102 #define DDRB (*((volatile uint8_t*)0x37))
00110 #define PINB (*((volatile uint8_t*)0x36))
00112 /***** Port C registers
00113 *****/
00125 #define PORTC (*((volatile uint8_t*)0x35))
00133 #define DDRC (*((volatile uint8_t*)0x34))
00141 #define PINC (*((volatile uint8_t*)0x33))
00143 /***** Port D registers
00144 *****/
00156 #define PORTD (*((volatile uint8_t*)0x32))
00164 #define DDRD (*((volatile uint8_t*)0x31))
00172 #define PIND (*((volatile uint8_t*)0x30))
```

```
00174 /***** Interrupts registers
00175 *****/
00189 #define GICR (*((volatile uint8_t*)0x5B))
00190
00200 #define GIFR (*((volatile uint8_t*)0x5A))
00218 #define MCUCR (*((volatile uint8_t*)0x55))
00219
00231 #define MCUCSR (*((volatile uint8_t*)0x54))
00233 /***** Timers registers
00234 *****/
00321 #define TCCR0 (*((volatile uint8_t*)0x53))
00333 #define TCNT0 (*((volatile uint8_t*)0x52))
00344 #define OCR0 (*((volatile uint8_t*)0x5C))
00374 #define TIFR (*((volatile uint8_t*)0x58))
00391 #define TIMSK (*((volatile uint8_t*)0x59))
00396 #endif /* REGISTERFILE_H_ */
```

# Index

- ADC
  - ATMEGA32 interrupts definitions, [28](#)
- ANA\_COMP
  - ATMEGA32 interrupts definitions, [28](#)
- ANY\_LOGICAL\_CHANGE
  - ATMEGA32 external interrupts driver, [26](#)
- app.c
  - ISR, [68](#)
  - SYSTEM\_CLK, [68](#)
  - system\_CurrentState, [68](#)
- app\_start
  - Application layer, [10](#)
- Application layer, [7](#)
  - app\_start, [10](#)
  - BUTTON\_PRESSED\_AT\_GREEN, [10](#)
  - BUTTON\_PRESSED\_AT\_RED, [10](#)
  - BUTTON\_PRESSED\_AT\_YELLOW, [10](#)
  - CARS\_GREEN\_LED, [9](#)
  - CARS\_RED\_LED, [9](#)
  - CARS\_YELLOW\_LED, [9](#)
  - carsGreenState, [11](#)
  - carsRedState, [11](#)
  - carsYellowState, [11](#)
  - EN\_TrafficStates\_t, [10](#)
  - NORMAL\_GREEN, [10](#)
  - NORMAL\_RED, [10](#)
  - NORMAL\_YELLOW\_AFTER\_GREEN, [10](#)
  - NORMAL\_YELLOW\_AFTER\_RED, [10](#)
  - PEDESTRIANS\_GREEN\_LED, [9](#)
  - PEDESTRIANS\_RED\_LED, [9](#)
  - PEDESTRIANS\_YELLOW\_LED, [9](#)
  - pedestriansGreenState, [12](#)
  - pedestriansRedState, [12](#)
  - pedestriansYellowState, [13](#)
  - trafficLightOperation, [13](#)
- ATMEGA32 external interrupts driver, [22](#)
  - ANY\_LOGICAL\_CHANGE, [26](#)
  - EN\_interruptError\_t, [25](#)
  - EN\_interruptNum\_t, [26](#)
  - EN\_interruptSenseControl\_t, [26](#)
  - Ext\_interruptInit, [27](#)
  - FALLING\_EDGE, [26](#)
  - INT0, [26](#)
  - INT0\_PIN, [23](#)
  - INT1, [26](#)
  - INT1\_PIN, [24](#)
  - INT2, [26](#)
  - INT2\_PIN, [24](#)
  - INT\_OK, [25](#)
  - ISC00, [24](#)
  - ISC01, [24](#)
  - ISC10, [24](#)
  - ISC11, [25](#)
  - ISC2, [25](#)
  - LOW\_LEVEL, [26](#)
  - RISING\_EDGE, [26](#)
  - WRONG\_INT\_NUM, [25](#)
  - WRONG\_SENSE\_CONTROL, [25](#)
- ATMEGA32 interrupts definitions, [27](#)
  - ADC, [28](#)
  - ANA\_COMP, [28](#)
  - cli, [29](#)
  - EE\_RDY, [29](#)
  - EXT\_INT0, [29](#)
  - EXT\_INT1, [29](#)
  - EXT\_INT2, [30](#)
  - ISR, [30](#)
  - sei, [30](#)
  - SPI\_STC, [30](#)
  - SPM\_RDY, [31](#)
  - TIM0\_COMP, [31](#)
  - TIM0\_OVF, [31](#)
  - TIM1\_CAPT, [31](#)
  - TIM1\_COMPA, [31](#)
  - TIM1\_COMPB, [32](#)
  - TIM1\_OVF, [32](#)
  - TIM2\_COMP, [32](#)
  - TIM2\_OVF, [32](#)
  - TWI, [32](#)
  - USART\_RXC, [33](#)
  - USART\_TXC, [33](#)
  - USART\_UDRE, [33](#)
- Bit math, [48](#)
  - clrBit, [48](#)
  - getBit, [49](#)
  - setBit, [49](#)
  - toggleBit, [50](#)
- Button driver, [14](#)
  - buttonInit, [14](#)
  - buttonRead, [16](#)
- BUTTON\_PRESSED\_AT\_GREEN
  - Application layer, [10](#)
- BUTTON\_PRESSED\_AT\_RED
  - Application layer, [10](#)
- BUTTON\_PRESSED\_AT\_YELLOW
  - Application layer, [10](#)
- buttonInit
  - Button driver, [14](#)

- buttonRead
  - Button driver, [16](#)
- CARS\_GREEN\_LED
  - Application layer, [9](#)
- CARS\_RED\_LED
  - Application layer, [9](#)
- CARS\_YELLOW\_LED
  - Application layer, [9](#)
- carsGreenState
  - Application layer, [11](#)
- carsRedState
  - Application layer, [11](#)
- carsYellowState
  - Application layer, [11](#)
- cli
  - ATMEGA32 interrupts definitions, [29](#)
- clk\_DIVISION\_BY\_1024
  - Timer0 driver, [39](#)
- clk\_DIVISION\_BY\_256
  - Timer0 driver, [39](#)
- clk\_DIVISION\_BY\_64
  - Timer0 driver, [39](#)
- clk\_DIVISION\_BY\_8
  - Timer0 driver, [39](#)
- clk\_No\_DIVISION
  - Timer0 driver, [39](#)
- CLR\_TIMER0\_CLK\_SRC
  - Timer0 driver, [36](#)
- CLR\_TIMER0\_MODE
  - Timer0 driver, [36](#)
- clrBit
  - Bit math, [48](#)
- CS00
  - Timer0 driver, [36](#)
- CS01
  - Timer0 driver, [36](#)
- CS02
  - Timer0 driver, [37](#)
- CTC
  - Timer0 driver, [40](#)
- DDRA
  - Port A registers, [54](#)
- DDRB
  - Port B registers, [55](#)
- DDRC
  - Port C registers, [56](#)
- DDRD
  - Port D registers, [58](#)
- Definition of data types, [50](#)
  - float128\_t, [51](#)
  - float32\_t, [51](#)
  - float64\_t, [51](#)
  - sint16\_t, [51](#)
  - sint32\_t, [51](#)
  - sint8\_t, [51](#)
  - uint16\_t, [52](#)
  - uint32\_t, [52](#)
  - uint8\_t, [52](#)
- DIO driver, [19](#)
  - DIO\_pinInit, [20](#)
  - DIO\_pinRead, [20](#)
  - DIO\_pinToggle, [21](#)
  - DIO\_pinWrite, [21](#)
- DIO\_pinInit
  - DIO driver, [20](#)
- DIO\_pinRead
  - DIO driver, [20](#)
- DIO\_pinToggle
  - DIO driver, [21](#)
- DIO\_pinWrite
  - DIO driver, [21](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/App/app.c, [67](#), [69](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/App/app.h, [71](#), [73](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/Button driver/Button.c, [73](#), [74](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/Button driver/Button.h, [74](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/LED driver/LED.c, [75](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/ECUAL/LED driver/LED.h, [75](#), [76](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/IncFile1.h, [76](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/main.c, [77](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Dio driver/DIO.c, [77](#), [78](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Dio driver/DIO.h, [81](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Ext interrupt driver/Ext interrupt.c, [82](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Ext interrupt driver/Ext interrupt.h, [83](#), [85](#)

- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Interrupt/Interrupt.h, [85, 86](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Timer driver/Timer\_0.c,Timer0 driver, [39](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Timer driver/Timer\_0.h,Timer0 driver, [39](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Timer driver/Timer\_0.h,Timer0 driver, [90, 92](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/ATmega32Port.h, [94](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/BitMath.h, [95](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/dataTypes.h, [96](#)
- E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/Service/RegisterFile.h, [96, 97](#)
- ECUAL layer, [14](#)
- EE\_RDY
  - ATMEGA32 interrupts definitions, [29](#)
- EN\_interruptError\_t
  - ATMEGA32 external interrupts driver, [25](#)
- EN\_interruptNum\_t
  - ATMEGA32 external interrupts driver, [26](#)
- EN\_interruptSenseControl\_t
  - ATMEGA32 external interrupts driver, [26](#)
- EN\_pinDirection\_t
  - MCU ports, [46](#)
- EN\_pinErro\_t
  - MCU ports, [46](#)
- EN\_pinNum\_t
  - MCU ports, [46](#)
- EN\_pinState\_t
  - MCU ports, [47](#)
- EN\_Timer0\_clkSource\_t
  - Timer0 driver, [38](#)
- En\_Timer0\_Error\_t
  - Timer0 driver, [39](#)
- EN\_Timer0\_Mode\_t
  - Timer0 driver, [39](#)
- EN\_TrafficStates\_t
  - Application layer, [10](#)
- EXT\_INT0
  - ATMEGA32 interrupts definitions, [29](#)
- EXT\_INT1
  - ATMEGA32 interrupts definitions, [29](#)
- EXT\_INT2
  - ATMEGA32 interrupts definitions, [30](#)
- Ext\_interruptInit
  - ATMEGA32 external interrupts driver, [27](#)
- EXTERNAL\_CLOCK\_FALLING\_EDGE
  - Timer0 driver, [39](#)
- EXTERNAL\_CLOCK\_RISING\_EDGE
  - Timer0 driver, [39](#)
- FALLING\_EDGE
  - ATMEGA32 external interrupts driver, [26](#)
- FAST\_PWM
  - Timer0 driver, [40](#)
- float128\_t
  - Definition of data types, [51](#)
- float32\_t
  - Definition of data types, [51](#)
- float64\_t
  - Definition of data types, [51](#)
- General Timers registers, [65](#)
  - TIFR, [65](#)
  - TIMSK, [66](#)
- getBit
  - Bit math, [49](#)
- GICR
  - Interrupt registers, [59](#)
- GIFR
  - Interrupt registers, [59](#)
- High
  - MCU ports, [47](#)
- I/O registers, [53](#)
- Input
  - MCU ports, [46](#)
- INT0
  - ATMEGA32 external interrupts driver, [26](#)
- INT0\_PIN
  - ATMEGA32 external interrupts driver, [23](#)
- INT1
  - ATMEGA32 external interrupts driver, [26](#)
- INT1\_PIN
  - ATMEGA32 external interrupts driver, [24](#)
- INT2
  - ATMEGA32 external interrupts driver, [26](#)
- INT2\_PIN
  - ATMEGA32 external interrupts driver, [24](#)
- INT\_OK
  - ATMEGA32 external interrupts driver, [25](#)
- Interrupt registers, [59](#)
  - GICR, [59](#)
  - GIFR, [59](#)
  - MCUCR, [60](#)
  - MCUCSR, [61](#)
- Interrupts driver, [27](#)
- ISC00
  - ATMEGA32 external interrupts driver, [24](#)
- ISC01
  - ATMEGA32 external interrupts driver, [24](#)

- ISC10
  - ATMEGA32 external interrupts driver, [24](#)
- ISC11
  - ATMEGA32 external interrupts driver, [25](#)
- ISC2
  - ATMEGA32 external interrupts driver, [25](#)
- ISR
  - app.c, [68](#)
  - ATMEGA32 interrupts definitions, [30](#)
  - Timer\_0.c, [87](#)
- LED driver, [16](#)
  - ledInit, [17](#)
  - ledOff, [17](#)
  - ledOn, [18](#)
  - ledToggle, [18](#)
- ledInit
  - LED driver, [17](#)
- ledOff
  - LED driver, [17](#)
- ledOn
  - LED driver, [18](#)
- ledToggle
  - LED driver, [18](#)
- Low
  - MCU ports, [47](#)
- LOW\_LEVEL
  - ATMEGA32 external interrupts driver, [26](#)
- main
  - main.c, [77](#)
- main.c
  - main, [77](#)
- MCAL layer, [19](#)
- MCU ports, [44](#)
  - EN\_pinDirection\_t, [46](#)
  - EN\_pinErro\_t, [46](#)
  - EN\_pinNum\_t, [46](#)
  - EN\_pinState\_t, [47](#)
  - High, [47](#)
  - Input, [46](#)
  - Low, [47](#)
  - OK, [46](#)
  - Output, [46](#)
  - PA0, [47](#)
  - PA1, [47](#)
  - PA2, [47](#)
  - PA3, [47](#)
  - PA4, [47](#)
  - PA5, [47](#)
  - PA6, [47](#)
  - PA7, [47](#)
  - PB0, [47](#)
  - PB1, [47](#)
  - PB2, [47](#)
  - PB3, [47](#)
  - PB4, [47](#)
  - PB5, [47](#)
  - PB6, [47](#)
  - PB7, [47](#)
  - PC0, [47](#)
  - PC1, [47](#)
  - PC2, [47](#)
  - PC3, [47](#)
  - PC4, [47](#)
  - PC5, [47](#)
  - PC6, [47](#)
  - PC7, [47](#)
  - PD0, [47](#)
  - PD1, [47](#)
  - PD2, [47](#)
  - PD3, [47](#)
  - PD4, [47](#)
  - PD5, [47](#)
  - PD6, [47](#)
  - PD7, [47](#)
  - PORTA\_OFFSET, [45](#)
  - PORTB\_OFFSET, [45](#)
  - PORTC\_OFFSET, [45](#)
  - PORTD\_OFFSET, [45](#)
  - WRONG\_PIN\_DIR, [46](#)
  - WRONG\_PIN\_NUM, [46](#)
  - WRONG\_PIN\_STATE, [46](#)
- MCU Registers, [53](#)
- MCUCR
  - Interrupt registers, [60](#)
- MCUCSR
  - Interrupt registers, [61](#)
- NO\_CLOCK\_SOURCE
  - Timer0 driver, [39](#)
- NORMAL
  - Timer0 driver, [40](#)
- NORMAL\_GREEN
  - Application layer, [10](#)
- NORMAL\_RED
  - Application layer, [10](#)
- NORMAL\_YELLOW\_AFTER\_GREEN
  - Application layer, [10](#)
- NORMAL\_YELLOW\_AFTER\_RED
  - Application layer, [10](#)
- OCF0
  - Timer0 driver, [37](#)
- OCIE0
  - Timer0 driver, [37](#)
- OCR0
  - Timer0 Registers, [62](#)
- OK
  - MCU ports, [46](#)
- Output
  - MCU ports, [46](#)
- PA0
  - MCU ports, [47](#)
- PA1
  - MCU ports, [47](#)
- PA2



- MCU ports, [47](#)
- PA3
  - MCU ports, [47](#)
- PA4
  - MCU ports, [47](#)
- PA5
  - MCU ports, [47](#)
- PA6
  - MCU ports, [47](#)
- PA7
  - MCU ports, [47](#)
- PB0
  - MCU ports, [47](#)
- PB1
  - MCU ports, [47](#)
- PB2
  - MCU ports, [47](#)
- PB3
  - MCU ports, [47](#)
- PB4
  - MCU ports, [47](#)
- PB5
  - MCU ports, [47](#)
- PB6
  - MCU ports, [47](#)
- PB7
  - MCU ports, [47](#)
- PC0
  - MCU ports, [47](#)
- PC1
  - MCU ports, [47](#)
- PC2
  - MCU ports, [47](#)
- PC3
  - MCU ports, [47](#)
- PC4
  - MCU ports, [47](#)
- PC5
  - MCU ports, [47](#)
- PC6
  - MCU ports, [47](#)
- PC7
  - MCU ports, [47](#)
- PD0
  - MCU ports, [47](#)
- PD1
  - MCU ports, [47](#)
- PD2
  - MCU ports, [47](#)
- PD3
  - MCU ports, [47](#)
- PD4
  - MCU ports, [47](#)
- PD5
  - MCU ports, [47](#)
- PD6
  - MCU ports, [47](#)
- PD7
  - MCU ports, [47](#)
- MCU ports, [47](#)
- PEDESTRIANS\_GREEN\_LED
  - Application layer, [9](#)
- PEDESTRIANS\_RED\_LED
  - Application layer, [9](#)
- PEDESTRIANS\_YELLOW\_LED
  - Application layer, [9](#)
- pedestriansGreenState
  - Application layer, [12](#)
- pedestriansRedState
  - Application layer, [12](#)
- pedestriansYellowState
  - Application layer, [13](#)
- PINA
  - Port A registers, [54](#)
- PINB
  - Port B registers, [55](#)
- PINC
  - Port C registers, [57](#)
- PIND
  - Port D registers, [58](#)
- Port A registers, [54](#)
  - DDRA, [54](#)
  - PINA, [54](#)
  - PORTA, [54](#)
- Port B registers, [55](#)
  - DDRB, [55](#)
  - PINB, [55](#)
  - PORTB, [56](#)
- Port C registers, [56](#)
  - DDRC, [56](#)
  - PINC, [57](#)
  - PORTC, [57](#)
- Port D registers, [58](#)
  - DDRD, [58](#)
  - PIND, [58](#)
  - PORTD, [58](#)
- PORTA
  - Port A registers, [54](#)
- PORTA\_OFFSET
  - MCU ports, [45](#)
- PORTB
  - Port B registers, [56](#)
- PORTB\_OFFSET
  - MCU ports, [45](#)
- PORTC
  - Port C registers, [57](#)
- PORTC\_OFFSET
  - MCU ports, [45](#)
- PORTD
  - Port D registers, [58](#)
- PORTD\_OFFSET
  - MCU ports, [45](#)
- PWM\_PHASE\_CORRECR
  - Timer0 driver, [40](#)
- RISING\_EDGE
  - ATMEGA32 external interrupts driver, [26](#)

- sei
  - ATMEGA32 interrupts definitions, 30
- Service layer, 52
- setBit
  - Bit math, 49
- sint16\_t
  - Definition of data types, 51
- sint32\_t
  - Definition of data types, 51
- sint8\_t
  - Definition of data types, 51
- SPI\_STC
  - ATMEGA32 interrupts definitions, 30
- SPM\_RDY
  - ATMEGA32 interrupts definitions, 31
- SYSTEM\_CLK
  - app.c, 68
- system\_CurrentState
  - app.c, 68
- TCCR0
  - Timer0 Registers, 62
- TCNT0
  - Timer0 Registers, 64
- TIFR
  - General Timers registers, 65
- TIM0\_COMP
  - ATMEGA32 interrupts definitions, 31
- TIM0\_OVF
  - ATMEGA32 interrupts definitions, 31
- TIM1\_CAPT
  - ATMEGA32 interrupts definitions, 31
- TIM1\_COMPA
  - ATMEGA32 interrupts definitions, 31
- TIM1\_COMPB
  - ATMEGA32 interrupts definitions, 32
- TIM1\_OVF
  - ATMEGA32 interrupts definitions, 32
- TIM2\_COMP
  - ATMEGA32 interrupts definitions, 32
- TIM2\_OVF
  - ATMEGA32 interrupts definitions, 32
- Timer0 driver, 34
  - clk\_DIVISION\_BY\_1024, 39
  - clk\_DIVISION\_BY\_256, 39
  - clk\_DIVISION\_BY\_64, 39
  - clk\_DIVISION\_BY\_8, 39
  - clk\_No\_DIVISION, 39
  - CLR\_TIMER0\_CLK\_SRC, 36
  - CLR\_TIMER0\_MODE, 36
  - CS00, 36
  - CS01, 36
  - CS02, 37
  - CTC, 40
  - EN\_Timer0\_clkSource\_t, 38
  - En\_Timer0\_Error\_t, 39
  - EN\_Timer0\_Mode\_t, 39
  - EXTERNAL\_CLOCK\_FALLING\_EDGE, 39
  - EXTERNAL\_CLOCK\_RISING\_EDGE, 39
  - FAST\_PWM, 40
  - NO\_CLOCK\_SOURCE, 39
  - NORMAL, 40
  - OCF0, 37
  - OCIE0, 37
  - PWM\_PHASE\_CORRECR, 40
  - Timer0\_delay\_ms, 41
  - Timer0\_init, 41
  - TIMER0\_interrupt\_t, 40
  - Timer0\_interruptDiable, 42
  - Timer0\_interruptEnable, 42
  - TIMER0\_NUM\_OF\_TICKS, 37
  - TIMER0\_OK, 39
  - TIMER0\_OUT\_CMP\_MATCH\_INT, 41
  - TIMER0\_OVER\_FLOW\_INT, 41
  - Timer0\_reset, 43
  - Timer0\_start, 43
  - Timer0\_stop, 44
  - TIMER0\_WRONG\_CLK\_SOURCE, 39
  - TIMER0\_WRONG\_INT, 39
  - TIMER0\_WRONG\_MODE, 39
  - TOIE0, 37
  - TOV0, 38
  - WGM00, 38
  - WGM01, 38
- Timer0 Registers, 61
  - OCR0, 62
  - TCCR0, 62
  - TCNT0, 64
- Timer0\_delay\_ms
  - Timer0 driver, 41
- Timer0\_globalClkSource
  - Timer\_0.c, 87
- Timer0\_globalNumOfOverFlows
  - Timer\_0.c, 88
- Timer0\_globalOverFlowTime
  - Timer\_0.c, 88
- Timer0\_globalTickTime
  - Timer\_0.c, 88
- Timer0\_init
  - Timer0 driver, 41
- TIMER0\_interrupt\_t
  - Timer0 driver, 40
- Timer0\_interruptDiable
  - Timer0 driver, 42
- Timer0\_interruptEnable
  - Timer0 driver, 42
- TIMER0\_NUM\_OF\_TICKS
  - Timer0 driver, 37
- TIMER0\_OK
  - Timer0 driver, 39
- TIMER0\_OUT\_CMP\_MATCH\_INT
  - Timer0 driver, 41
- TIMER0\_OVER\_FLOW\_INT
  - Timer0 driver, 41
- Timer0\_reset
  - Timer0 driver, 43
- Timer0\_start

- Timer0 driver, [43](#)
- Timer0\_stop
  - Timer0 driver, [44](#)
- TIMER0\_WRONG\_CLK\_SOURCE
  - Timer0 driver, [39](#)
- TIMER0\_WRONG\_INT
  - Timer0 driver, [39](#)
- TIMER0\_WRONG\_MODE
  - Timer0 driver, [39](#)
- Timer\_0.c
  - ISR, [87](#)
  - Timer0\_globalClkSource, [87](#)
  - Timer0\_globalNumOfOverFlows, [88](#)
  - Timer0\_globalOverflowTime, [88](#)
  - Timer0\_globalTickTime, [88](#)
- Timers driver, [33](#)
- Timers Registers, [61](#)
- TIMSK
  - General Timers registers, [66](#)
- toggleBit
  - Bit math, [50](#)
- TOIE0
  - Timer0 driver, [37](#)
- TOV0
  - Timer0 driver, [38](#)
- trafficLightOperation
  - Application layer, [13](#)
- TWI
  - ATMEGA32 interrupts definitions, [32](#)
- uint16\_t
  - Definition of data types, [52](#)
- uint32\_t
  - Definition of data types, [52](#)
- uint8\_t
  - Definition of data types, [52](#)
- USART\_RXC
  - ATMEGA32 interrupts definitions, [33](#)
- USART\_TXC
  - ATMEGA32 interrupts definitions, [33](#)
- USART\_UDRE
  - ATMEGA32 interrupts definitions, [33](#)
- WGM00
  - Timer0 driver, [38](#)
- WGM01
  - Timer0 driver, [38](#)
- WRONG\_INT\_NUM
  - ATMEGA32 external interrupts driver, [25](#)
- WRONG\_PIN\_DIR
  - MCU ports, [46](#)
- WRONG\_PIN\_NUM
  - MCU ports, [46](#)
- WRONG\_PIN\_STATE
  - MCU ports, [46](#)
- WRONG\_SENSE\_CONTROL
  - ATMEGA32 external interrupts driver, [25](#)