

## On-Demand Traffic Control

Generated by Doxygen 1.9.4



<b>1 Module Index</b>	<b>1</b>
1.1 Modules	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Module Documentation</b>	<b>5</b>
3.1 ECUAL layer	5
3.1.1 Detailed Description	5
3.2 Button driver	5
3.2.1 Detailed Description	5
3.2.2 Function Documentation	5
3.2.2.1 buttonInit()	5
3.2.2.2 buttonRead()	6
3.3 LED driver	6
3.3.1 Detailed Description	7
3.3.2 Function Documentation	7
3.3.2.1 ledInit()	7
3.3.2.2 ledOff()	7
3.3.2.3 ledOn()	8
3.3.2.4 ledToggle()	8
3.4 MCAL layer	9
3.4.1 Detailed Description	9
3.5 DIO driver	9
3.5.1 Detailed Description	10
3.5.2 Function Documentation	10
3.5.2.1 DIO_pinInit()	10
3.5.2.2 DIO_pinRead()	10
3.5.2.3 DIO_pinToggle()	11
3.5.2.4 DIO_pinWrite()	11
3.6 ATMEGA32 external interrupts driver	12
3.6.1 Detailed Description	13
3.6.2 Macro Definition Documentation	13
3.6.2.1 INT0_PIN	14
3.6.2.2 INT1_PIN	14
3.6.2.3 INT2_PIN	14
3.6.2.4 ISC00	14
3.6.2.5 ISC01	14
3.6.2.6 ISC10	15
3.6.2.7 ISC11	15
3.6.2.8 ISC2	15
3.6.3 Enumeration Type Documentation	15
3.6.3.1 EN_interruptError_t	15

3.6.3.2 EN_interruptNum_t	16
3.6.3.3 EN_interruptSenseControl_t	16
3.6.4 Function Documentation	17
3.6.4.1 Ext_interruptInit()	17
3.7 Interrupts driver	17
3.7.1 Detailed Description	17
3.8 ATMEGA32 interrupts definitions	17
3.8.1 Detailed Description	18
3.8.2 Macro Definition Documentation	18
3.8.2.1 ADC	18
3.8.2.2 ANA_COMP	19
3.8.2.3 cli	19
3.8.2.4 EE_RDY	19
3.8.2.5 EXT_INT0	19
3.8.2.6 EXT_INT1	20
3.8.2.7 EXT_INT2	20
3.8.2.8 ISR	20
3.8.2.9 sei	20
3.8.2.10 SPI_STC	21
3.8.2.11 SPM_RDY	21
3.8.2.12 TIM0_COMP	21
3.8.2.13 TIM0_OVF	21
3.8.2.14 TIM1_CAPT	21
3.8.2.15 TIM1_COMPA	22
3.8.2.16 TIM1_COMPB	22
3.8.2.17 TIM1_OVF	22
3.8.2.18 TIM2_COMP	22
3.8.2.19 TIM2_OVF	22
3.8.2.20 TWI	23
3.8.2.21 USART_RXC	23
3.8.2.22 USART_TXC	23
3.8.2.23 USART_UDRE	23
3.9 Timers driver	23
3.9.1 Detailed Description	24
3.10 Timer0 driver	24
3.10.1 Detailed Description	26
3.10.2 Macro Definition Documentation	26
3.10.2.1 CLR_TIMER0_CLK_SRC	26
3.10.2.2 CLR_TIMER0_MODE	26
3.10.2.3 CS00	26
3.10.2.4 CS01	27
3.10.2.5 CS02	27

3.10.2.6 OCF0	27
3.10.2.7 OCIE0	27
3.10.2.8 SYSTEM_CLK	27
3.10.2.9 TIMER0_NUM_OF_TICKS	28
3.10.2.10 TOIE0	28
3.10.2.11 TOV0	28
3.10.2.12 WGM00	28
3.10.2.13 WGM01	28
3.10.3 Enumeration Type Documentation	29
3.10.3.1 EN_Timer0_clkSource_t	29
3.10.3.2 En_Timer0_Error_t	29
3.10.3.3 EN_Timer0_Mode_t	30
3.10.3.4 TIMER0_interrupt_t	31
3.10.4 Function Documentation	31
3.10.4.1 Timer0_delay_ms()	31
3.10.4.2 Timer0_init()	32
3.10.4.3 Timer0_interruptDisable()	32
3.10.4.4 Timer0_interruptEnable()	33
3.10.4.5 Timer0_reset()	33
3.10.4.6 Timer0_start()	34
3.10.4.7 Timer0_stop()	34
3.11 MCU ports	35
3.11.1 Detailed Description	35
3.11.2 Macro Definition Documentation	35
3.11.2.1 PORTA_OFFSET	36
3.11.2.2 PORTB_OFFSET	36
3.11.2.3 PORTC_OFFSET	36
3.11.2.4 PORTD_OFFSET	36
3.11.3 Enumeration Type Documentation	36
3.11.3.1 EN_pinDirection_t	36
3.11.3.2 EN_pinError_t	37
3.11.3.3 EN_pinNum_t	37
3.11.3.4 EN_pinState_t	38
3.12 Bit math	38
3.12.1 Detailed Description	39
3.12.2 Macro Definition Documentation	39
3.12.2.1 clrBit	39
3.12.2.2 getBit	39
3.12.2.3 setBit	40
3.12.2.4 toggleBit	40
3.13 Definition of data types	41
3.13.1 Detailed Description	41

3.13.2 Typedef Documentation	41
3.13.2.1 float128_t	41
3.13.2.2 float32_t	41
3.13.2.3 float64_t	42
3.13.2.4 sint16_t	42
3.13.2.5 sint32_t	42
3.13.2.6 sint8_t	42
3.13.2.7 uint16_t	42
3.13.2.8 uint32_t	43
3.13.2.9 uint8_t	43
3.14 Service layer	43
3.14.1 Detailed Description	43
3.15 MCU Registers	43
3.15.1 Detailed Description	43
3.16 I/O registers	44
3.16.1 Detailed Description	44
3.17 Port A registers	44
3.17.1 Detailed Description	44
3.17.2 Macro Definition Documentation	44
3.17.2.1 DDRA	44
3.17.2.2 PINA	45
3.17.2.3 PORTA	45
3.18 Port B registers	45
3.18.1 Detailed Description	46
3.18.2 Macro Definition Documentation	46
3.18.2.1 DDRB	46
3.18.2.2 PINB	46
3.18.2.3 PORTB	46
3.19 Port C registers	47
3.19.1 Detailed Description	47
3.19.2 Macro Definition Documentation	47
3.19.2.1 DDRC	47
3.19.2.2 PINC	47
3.19.2.3 PORTC	48
3.20 Port D registers	48
3.20.1 Detailed Description	48
3.20.2 Macro Definition Documentation	48
3.20.2.1 DDRD	48
3.20.2.2 PIND	49
3.20.2.3 PORTD	49
3.21 Interrupt registers	49
3.21.1 Detailed Description	50

3.21.2 Macro Definition Documentation	50
3.21.2.1 GICR	50
3.21.2.2 GIFR	50
3.21.2.3 MCUCR	51
3.21.2.4 MCUCSR	51
3.22 Timers Registers	52
3.22.1 Detailed Description	52
3.23 Timer0 Registers	52
3.23.1 Detailed Description	52
3.23.2 Macro Definition Documentation	52
3.23.2.1 OCR0	52
3.23.2.2 TCCR0	53
3.23.2.3 TCNT0	55
3.24 General Timers registers	55
3.24.1 Detailed Description	55
3.24.2 Macro Definition Documentation	55
3.24.2.1 TIFR	56
3.24.2.2 TIMSK	56
<b>4 File Documentation</b>	<b>59</b>
4.1 App/app.c File Reference	59
4.2 app.c	59
4.3 App/app.h File Reference	59
4.4 app.h	59
4.5 Debug/App/app.d File Reference	60
4.6 app.d	60
4.7 Debug/ECUAL/Button driver/Button.d File Reference	60
4.8 Button.d	60
4.9 Debug/ECUAL/LED driver/LED.d File Reference	60
4.10 LED.d	60
4.11 Debug/main.d File Reference	61
4.12 main.d	61
4.13 Debug/MCAL/Dio driver/DIO.d File Reference	61
4.14 DIO.d	61
4.15 Debug/MCAL/Ext interrupt driver/Ext interrupt.d File Reference	62
4.16 Ext interrupt.d	62
4.17 Debug/MCAL/Timer driver/Timer_0.d File Reference	62
4.18 Timer_0.d	62
4.19 ECUAL/Button driver/Button.c File Reference	63
4.20 Button.c	63
4.21 ECUAL/Button driver/Button.h File Reference	63
4.22 Button.h	64

4.23 ECUAL/LED driver/LED.c File Reference . . . . .	64
4.24 LED.c . . . . .	64
4.25 ECUAL/LED driver/LED.h File Reference . . . . .	65
4.26 LED.h . . . . .	65
4.27 main.c File Reference . . . . .	66
4.27.1 Function Documentation . . . . .	66
4.27.1.1 main() . . . . .	66
4.28 main.c . . . . .	66
4.29 MCAL/Dio driver/DIO.c File Reference . . . . .	67
4.29.1 Function Documentation . . . . .	67
4.29.1.1 ISR() . . . . .	67
4.30 DIO.c . . . . .	67
4.31 MCAL/Dio driver/DIO.h File Reference . . . . .	70
4.31.1 Detailed Description . . . . .	71
4.32 DIO.h . . . . .	71
4.33 MCAL/Ext interrupt driver/Ext interrupt.c File Reference . . . . .	71
4.34 Ext interrupt.c . . . . .	72
4.35 MCAL/Ext interrupt driver/Ext interrupt.h File Reference . . . . .	73
4.36 Ext interrupt.h . . . . .	74
4.37 MCAL/Interrupt/Interrupt.h File Reference . . . . .	75
4.38 Interrupt.h . . . . .	76
4.39 MCAL/Timer driver/Timer_0.c File Reference . . . . .	76
4.39.1 Function Documentation . . . . .	77
4.39.1.1 ISR() . . . . .	77
4.39.2 Variable Documentation . . . . .	77
4.39.2.1 Timer0_globalClkSource . . . . .	77
4.39.2.2 Timer0_globalNumOfOverFlows . . . . .	77
4.39.2.3 Timer0_globalOverFlowTime . . . . .	78
4.39.2.4 Timer0_globalTickTime . . . . .	78
4.40 Timer_0.c . . . . .	78
4.41 MCAL/Timer driver/Timer_0.h File Reference . . . . .	80
4.42 Timer_0.h . . . . .	82
4.43 Service/ATmega32Port.h File Reference . . . . .	83
4.44 ATmega32Port.h . . . . .	83
4.45 Service/BitMath.h File Reference . . . . .	84
4.46 BitMath.h . . . . .	85
4.47 Service/dataTypes.h File Reference . . . . .	85
4.48 dataType.h . . . . .	85
4.49 Service/RegisterFile.h File Reference . . . . .	86
4.50 RegisterFile.h . . . . .	86



# Chapter 1

## Module Index

### 1.1 Modules

Here is a list of all modules:

ECUAL layer . . . . .	5
Button driver . . . . .	5
LED driver . . . . .	6
MCAL layer . . . . .	9
DIO driver . . . . .	9
Interrupts driver . . . . .	17
ATMEGA32 external interrupts driver . . . . .	12
ATMEGA32 interrupts definitions . . . . .	17
Timers driver . . . . .	23
Timer0 driver . . . . .	24
Service layer . . . . .	43
MCU ports . . . . .	35
Bit math . . . . .	38
Definition of data types . . . . .	41
MCU Registers . . . . .	43
I/O registers . . . . .	44
Port A registers . . . . .	44
Port B registers . . . . .	45
Port C registers . . . . .	47
Port D registers . . . . .	48
Interrupt registers . . . . .	49
Timers Registers . . . . .	52
Timer0 Registers . . . . .	52
General Timers registers . . . . .	55



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">main.c</a> . . . . .	66
App/ <a href="#">app.c</a> . . . . .	59
App/ <a href="#">app.h</a> . . . . .	59
Debug/ <a href="#">main.d</a> . . . . .	61
Debug/App/ <a href="#">app.d</a> . . . . .	60
Debug/ECUAL/Button driver/ <a href="#">Button.d</a> . . . . .	60
Debug/ECUAL/LED driver/ <a href="#">LED.d</a> . . . . .	60
Debug/MCAL/Dio driver/ <a href="#">DIO.d</a> . . . . .	61
Debug/MCAL/Ext interrupt driver/ <a href="#">Ext interrupt.d</a> . . . . .	62
Debug/MCAL/Timer driver/ <a href="#">Timer_0.d</a> . . . . .	62
ECUAL/Button driver/ <a href="#">Button.c</a> . . . . .	63
ECUAL/Button driver/ <a href="#">Button.h</a> . . . . .	63
ECUAL/LED driver/ <a href="#">LED.c</a> . . . . .	64
ECUAL/LED driver/ <a href="#">LED.h</a> . . . . .	65
MCAL/Dio driver/ <a href="#">DIO.c</a> . . . . .	67
MCAL/Dio driver/ <a href="#">DIO.h</a> . . . . .	70
MCAL/Ext interrupt driver/ <a href="#">Ext interrupt.c</a> . . . . .	71
MCAL/Ext interrupt driver/ <a href="#">Ext interrupt.h</a> . . . . .	73
MCAL/Interrupt/ <a href="#">Interrupt.h</a> . . . . .	75
MCAL/Timer driver/ <a href="#">Timer_0.c</a> . . . . .	76
MCAL/Timer driver/ <a href="#">Timer_0.h</a> . . . . .	80
Service/ <a href="#">ATmega32Port.h</a> . . . . .	83
Service/ <a href="#">BitMath.h</a> . . . . .	84
Service/ <a href="#">dataTypes.h</a> . . . . .	85
Service/ <a href="#">RegisterFile.h</a> . . . . .	86



## Chapter 3

# Module Documentation

### 3.1 ECUAL layer

#### Modules

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

#### 3.1.1 Detailed Description

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

### 3.2 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.*

#### 3.2.1 Detailed Description

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

#### 3.2.2 Function Documentation

##### 3.2.2.1 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](#).

**3.2.2.2 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](#).

**3.3 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.*

### 3.3.1 Detailed Description

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

### 3.3.2 Function Documentation

#### 3.3.2.1 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](#).

#### 3.3.2.2 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](#).

### 3.3.2.3 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](#).

### 3.3.2.4 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](#).

## 3.4 MCAL layer

### Modules

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

#### 3.4.1 Detailed Description

This layer contains all the driver related to the MCU.

## 3.5 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.*

### 3.5.1 Detailed Description

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

### 3.5.2 Function Documentation

#### 3.5.2.1 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](#).

#### 3.5.2.2 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](#).

### 3.5.2.3 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](#).

### 3.5.2.4 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](#).

## 3.6 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`

#### 3.6.1 Detailed Description

External interrupts driver.

#### 3.6.2 Macro Definition Documentation

### 3.6.2.1 INT0\_PIN

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

This Pin connected to INT0 interrupt

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

### 3.6.2.2 INT1\_PIN

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

This Pin connected to INT1 interrupt

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

### 3.6.2.3 INT2\_PIN

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

This Pin connected to INT2 interrupt

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

### 3.6.2.4 ISC00

```
#define ISC00 0
```

Interrupt Sense Control 0 Bit 0

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

### 3.6.2.5 ISC01

```
#define ISC01 1
```

Interrupt Sense Control 0 Bit 1

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

### 3.6.2.6 ISC10

```
#define ISC10 2
```

Interrupt Sense Control 1 Bit 0

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

### 3.6.2.7 ISC11

```
#define ISC11 3
```

Interrupt Sense Control 1 Bit 1

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

### 3.6.2.8 ISC2

```
#define ISC2 6
```

Interrupt Sense Control 2 Bit 6

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

## 3.6.3 Enumeration Type Documentation

### 3.6.3.1 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](#).

### 3.6.3.2 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](#).

### 3.6.3.3 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](#).

### 3.6.4 Function Documentation

#### 3.6.4.1 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](#).

## 3.7 Interrupts driver

### Modules

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

#### 3.7.1 Detailed Description

## 3.8 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.*

### 3.8.1 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.

### 3.8.2 Macro Definition Documentation

#### 3.8.2.1 ADC

```
#define ADC __vector_16
```

This Macro defines ADC Conversion Complete Handler

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

### 3.8.2.2 ANA\_COMP

```
#define ANA_COMP __vector_18
```

This Macro defines Analog Comparator Handler

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

### 3.8.2.3 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](#).

### 3.8.2.4 EE\_RDY

```
#define EE_RDY __vector_17
```

This Macro defines EEPROM Ready Handler

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

### 3.8.2.5 EXT\_INT0

```
#define EXT_INT0 __vector_1
```

This Macro defines IRQ0 Handler

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

### 3.8.2.6 EXT\_INT1

```
#define EXT_INT1 __vector_2
```

This Macro defines IRQ1 Handler

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

### 3.8.2.7 EXT\_INT2

```
#define EXT_INT2 __vector_3
```

This Macro defines IRQ2 Handler

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

### 3.8.2.8 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](#).

### 3.8.2.9 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](#).

#### 3.8.2.10 SPI\_STC

```
#define SPI_STC __vector_12
```

This Macro defines SPI Transfer Complete Handler

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

#### 3.8.2.11 SPM\_RDY

```
#define SPM_RDY __vector_20
```

This Macro defines Store Program Memory Ready Handler

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

#### 3.8.2.12 TIM0\_COMP

```
#define TIM0_COMP __vector_10
```

This Macro defines Timer0 Compare Handler

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

#### 3.8.2.13 TIM0\_OVF

```
#define TIM0_OVF __vector_11
```

This Macro defines Timer0 Overflow Handler

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

#### 3.8.2.14 TIM1\_CAPT

```
#define TIM1_CAPT __vector_6
```

This Macro defines Timer1 Capture Handler

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

### 3.8.2.15 TIM1\_COMPA

```
#define TIM1_COMPA __vector_7
```

This Macro defines Timer1 CompareA Handler

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

### 3.8.2.16 TIM1\_COMPB

```
#define TIM1_COMPB __vector_8
```

This Macro defines Timer1 CompareB Handler

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

### 3.8.2.17 TIM1\_OVF

```
#define TIM1_OVF __vector_9
```

This Macro defines Timer1 Overflow Handler

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

### 3.8.2.18 TIM2\_COMP

```
#define TIM2_COMP __vector_4
```

This Macro defines Timer2 Compare Handler

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

### 3.8.2.19 TIM2\_OVF

```
#define TIM2_OVF __vector_5
```

This Macro defines Timer2 Overflow Handler

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

#### 3.8.2.20 TWI

```
#define TWI __vector_19
```

This Macro defines Two-wire Serial Interface Handler

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

#### 3.8.2.21 USART\_RXC

```
#define USART_RXC __vector_13
```

This Macro defines USART RX Complete Handler

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

#### 3.8.2.22 USART\_TXC

```
#define USART_TXC __vector_15
```

This Macro defines USART TX Complete Handler

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

#### 3.8.2.23 USART\_UDRE

```
#define USART_UDRE __vector_14
```

This Macro defines UDR Empty Handler

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

## 3.9 Timers driver

### Modules

- [Timer0 driver](#)

### 3.9.1 Detailed Description

This contains the drivers for Atmega32 Timers

## 3.10 Timer0 driver

### Macros

- #define `SYSTEM_CLK` 1000000UL  
*System clock Macro.*
- #define `TIMER0_NUM_OF_TICKS` 256  
*Number of Ticks.*
- #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.
<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

### 3.10.1 Detailed Description

### 3.10.2 Macro Definition Documentation

#### 3.10.2.1 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](#).

#### 3.10.2.2 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](#).

#### 3.10.2.3 CS00

```
#define CS00 0
```

Definition at line [68](#) of file [Timer\\_0.h](#).

#### 3.10.2.4 CS01

```
#define CS01 1
```

Definition at line 69 of file [Timer\\_0.h](#).

#### 3.10.2.5 CS02

```
#define CS02 2
```

Definition at line 70 of file [Timer\\_0.h](#).

#### 3.10.2.6 OCF0

```
#define OCF0 1
```

Bit 1 - OCF0: Output Compare Flag

Definition at line 136 of file [Timer\\_0.h](#).

#### 3.10.2.7 OCIE0

```
#define OCIE0 1
```

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

Definition at line 116 of file [Timer\\_0.h](#).

#### 3.10.2.8 SYSTEM\_CLK

```
#define SYSTEM_CLK 1000000UL
```

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.

Definition at line 36 of file [Timer\\_0.h](#).

### 3.10.2.9 TIMER0\_NUM\_OF\_TICKS

```
#define TIMER0_NUM_OF_TICKS 256
```

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](#).

### 3.10.2.10 TOIE0

```
#define TOIE0 0
```

Bit 0 - TOIE0: Timer/Counter0 Overflow Interrupt Enable

Definition at line 115 of file [Timer\\_0.h](#).

### 3.10.2.11 TOV0

```
#define TOV0 0
```

Bit 0 - TOV0: Timer/Counter0 Overflow Flag

Definition at line 135 of file [Timer\\_0.h](#).

### 3.10.2.12 WGM00

```
#define WGM00 3
```

Definition at line 96 of file [Timer\\_0.h](#).

### 3.10.2.13 WGM01

```
#define WGM01 6
```

Definition at line 97 of file [Timer\\_0.h](#).

### 3.10.3 Enumeration Type Documentation

#### 3.10.3.1 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 source as the system clock (No prescaling)
```

##### Enumerator

NO_CLOCK_SOURCE	No clock source (Timer/Counter stopped).
clkI_No_DIVISON	clkI/O/(No prescaling).
clkI_DIVISION_BY_8	clkI/O/8 (From prescaler).
clkI_DIVISION_BY_64	clkI/O/64 (From prescaler).
clkI_DIVISION_BY_256	clkI/O/256 (From prescaler).
clkI_DIVISION_BY_1024	clkI/O/1024 (From prescaler).
EXTERNAL_CLOCK_FALLING_EDGE	External clock source on T0 pin. Clock on falling edge.
EXTERNAL_CLOCK_RISING_EDGE	External clock source on T0 pin. Clock on rising edge.

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

#### 3.10.3.2 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](#).

### 3.10.3.3 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](#).

#### 3.10.3.4 TIMER0\_interrupt\_t

```
enum TIMER0\_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](#).

### 3.10.4 Function Documentation

#### 3.10.4.1 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 126 of file [Timer\\_0.c](#).

## 3.10.4.2 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](#).

## 3.10.4.3 Timer0\_interruptDiabale()

```
En_Timer0_Error_t Timer0_interruptDiabale (
    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](#).

**3.10.4.4 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](#).

**3.10.4.5 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](#).

#### 3.10.4.6 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](#).

#### 3.10.4.7 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](#).

## 3.11 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` }

#### 3.11.1 Detailed Description

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

#### 3.11.2 Macro Definition Documentation

#### 3.11.2.1 PORTA\_OFFSET

```
#define PORTA_OFFSET 0
```

This macro defines the start of the PORTA pins

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

#### 3.11.2.2 PORTB\_OFFSET

```
#define PORTB_OFFSET 8
```

This macro defines the start of the PORTB pins

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

#### 3.11.2.3 PORTC\_OFFSET

```
#define PORTC_OFFSET 16
```

This macro defines the start of the PORTC pins

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

#### 3.11.2.4 PORTD\_OFFSET

```
#define PORTD_OFFSET 24
```

This macro defines the start of the PORTD pins

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

### 3.11.3 Enumeration Type Documentation

#### 3.11.3.1 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](#).

### 3.11.3.2 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](#).

### 3.11.3.3 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

**Enumerator**

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](#).

**3.11.3.4 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](#).

**3.12 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.*

### 3.12.1 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.

### 3.12.2 Macro Definition Documentation

#### 3.12.2.1 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](#).

#### 3.12.2.2 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](#).

**3.12.2.3 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](#).

**3.12.2.4 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](#).

## 3.13 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](#)

#### 3.13.1 Detailed Description

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

#### 3.13.2 Typedef Documentation

##### 3.13.2.1 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](#).

##### 3.13.2.2 float32\_t

```
typedef float float32\_t
```

This is define a memory size of 4 byte float

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

### 3.13.2.3 float64\_t

```
typedef double float64_t
```

This is define a memory size of 8 byte float

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

### 3.13.2.4 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](#).

### 3.13.2.5 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](#).

### 3.13.2.6 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](#).

### 3.13.2.7 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](#).

### 3.13.2.8 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](#).

### 3.13.2.9 uint8\_t

```
typedef unsigned char uint8_t
```

This is define a memory size of 1 byte

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

## 3.14 Service layer

### Modules

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

### 3.14.1 Detailed Description

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

## 3.15 MCU Registers

### Modules

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

### 3.15.1 Detailed Description

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

## 3.16 I/O registers

### Modules

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

### 3.16.1 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. \argIf 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).

## 3.17 Port A registers

### Macros

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

### 3.17.1 Detailed Description

### 3.17.2 Macro Definition Documentation

#### 3.17.2.1 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](#).

#### 3.17.2.2 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](#).

#### 3.17.2.3 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](#).

## 3.18 Port B registers

### Macros

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

### 3.18.1 Detailed Description

### 3.18.2 Macro Definition Documentation

#### 3.18.2.1 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](#).

#### 3.18.2.2 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](#).

#### 3.18.2.3 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](#).

## 3.19 Port C registers

### Macros

- `#define PORTC (*((volatile uint8_t*)0x35))`
- `#define DDRC (*((volatile uint8_t*)0x34))`
- `#define PINC (*((volatile uint8_t*)0x33))`

### 3.19.1 Detailed Description

### 3.19.2 Macro Definition Documentation

#### 3.19.2.1 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](#).

#### 3.19.2.2 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](#).

### 3.19.2.3 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](#).

## 3.20 Port D registers

### Macros

- #define [PORTD](#) (\*((volatile [uint8\\_t](#)\*)0x32))
- #define [DDRD](#) (\*((volatile [uint8\\_t](#)\*)0x31))
- #define [PIND](#) (\*((volatile [uint8\\_t](#)\*)0x30))

### 3.20.1 Detailed Description

### 3.20.2 Macro Definition Documentation

#### 3.20.2.1 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](#).

### 3.20.2.2 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](#).

### 3.20.2.3 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](#).

## 3.21 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))

### 3.21.1 Detailed Description

### 3.21.2 Macro Definition Documentation

#### 3.21.2.1 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](#).

#### 3.21.2.2 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](#).

## 3.21.2.3 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](#).

## 3.21.2.4 MCUCSR

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

## MCU Control and Status Register.

Bit	7	6	5	4	3	2	1	0	
	<b>JTD</b>	<b>ISC2</b>	<b>—</b>	<b>JTRF</b>	<b>WDRF</b>	<b>BORF</b>	<b>EXTRF</b>	<b>PORF</b>	<b>MCUCSR</b>
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](#).

## 3.22 Timers Registers

### Modules

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

### 3.22.1 Detailed Description

## 3.23 Timer0 Registers

### Macros

- `#define TCCR0` (`((volatile uint8_t*)0x53)`)
- `#define TCNT0` (`((volatile uint8_t*)0x52)`)
- `#define OCR0` (`((volatile uint8_t*)0x5C)`)

### 3.23.1 Detailed Description

- This contains all the registers to control Timer0.

### 3.23.2 Macro Definition Documentation

#### 3.23.2.1 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](#).

## 3.23.2.2 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
Waveform Generation Mode Bit Description						

## 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.

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. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. 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. 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.

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

**3.23.2.3 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](#).

**3.24 General Timers registers****Macros**

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

**3.24.1 Detailed Description****3.24.2 Macro Definition Documentation**

### 3.24.2.1 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](#).

### 3.24.2.2 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](#).



## File Documentation

## 4.2 app.c

```
00001 /*****
00002 */
00003 */
00004 */
00005 */
```

#### 4.4 app.h

```
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
00010
00011
00012
00013
00014  #endif /* APP_H_ */
```

## 4.5 Debug/App/app.d File Reference

### 4.6 app.d

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

```
00001 App/app.d App/app.o:  ../App/app.c
```

## 4.7 Debug/ECUAL/Button driver/Button.d File Reference

### 4.8 Button.d

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

```
00001 ECUAL/Button driver/Button.d ECUAL/Button driver/Button.o:  \
00002  ../ECUAL/Button\ driver/Button.c  ../ECUAL/Button\ driver/Button.h \
00003  ../ECUAL/Button\ driver/../../Service/ATmega32Port.h \
00004  ../ECUAL/Button\ driver/../../MCAL/Dio\ driver/DIO.h \
00005  ../ECUAL/Button\ driver/../../MCAL/Dio\ driver/../../Service/ATmega32Port.h \
00006  ../ECUAL/Button\ driver/../../MCAL/Dio\ driver/../../Service/BitMath.h \
00007  ../ECUAL/Button\ driver/../../MCAL/Dio\ driver/../../Service/dataTypes.h \
00008  ../ECUAL/Button\ driver/../../MCAL/Dio\ driver/../../Service/RegisterFile.h
00009
00010  ../ECUAL/Button\ driver/Button.h:
00011
00012  ../ECUAL/Button\ driver/../../Service/ATmega32Port.h:
00013
00014  ../ECUAL/Button\ driver/../../MCAL/Dio\ driver/DIO.h:
00015
00016  ../ECUAL/Button\ driver/../../MCAL/Dio\ driver/../../Service/ATmega32Port.h:
00017
00018  ../ECUAL/Button\ driver/../../MCAL/Dio\ driver/../../Service/BitMath.h:
00019
00020  ../ECUAL/Button\ driver/../../MCAL/Dio\ driver/../../Service/dataTypes.h:
00021
00022  ../ECUAL/Button\ driver/../../MCAL/Dio\ driver/../../Service/RegisterFile.h:
```

## 4.9 Debug/ECUAL/LED driver/LED.d File Reference

### 4.10 LED.d

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

```
00001 ECUAL/LED driver/LED.d ECUAL/LED driver/LED.o:  ../ECUAL/LED\ driver/LED.c \
00002  ../ECUAL/LED\ driver/LED.h \
00003  ../ECUAL/LED\ driver/../../Service/ATmega32Port.h \
00004  ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/DIO.h \
00005  ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/ATmega32Port.h \
00006  ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/BitMath.h \
00007  ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/dataTypes.h \
00008  ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/RegisterFile.h
00009
00010  ../ECUAL/LED\ driver/LED.h:
00011
00012  ../ECUAL/LED\ driver/../../Service/ATmega32Port.h:
00013
00014  ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/DIO.h:
00015
00016  ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/ATmega32Port.h:
00017
00018  ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/BitMath.h:
00019
00020  ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/dataTypes.h:
00021
00022  ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/RegisterFile.h:
```

## 4.11 Debug/main.d File Reference

### 4.12 main.d

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

```
00001 main.d main.o: ../main.c ../ECUAL/LED\ driver/LED.h \
00002 ../ECUAL/LED\ driver/../../Service/ATmega32Port.h \
00003 ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/DIO.h \
00004 ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/ATmega32Port.h \
00005 ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/BitMath.h \
00006 ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/dataTypes.h \
00007 ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/RegisterFile.h \
00008 ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/dataTypes.h \
00009 ../MCAL/Ext\ interrupt\ driver/Ext\ interrupt.h \
00010 ../MCAL/Ext\ interrupt\ driver/../../Service/ATmega32Port.h \
00011 ../MCAL/Ext\ interrupt\ driver/../../Service/RegisterFile.h \
00012 ../MCAL/Ext\ interrupt\ driver/../../Interrupt/Interrupt.h \
00013 ../MCAL/Ext\ interrupt\ driver/../../Service/BitMath.h \
00014 ../MCAL/Timer\ driver/Timer_0.h \
00015 ../MCAL/Timer\ driver/../../Service/BitMath.h \
00016 ../MCAL/Timer\ driver/../../Service/ATmega32Port.h \
00017 ../MCAL/Timer\ driver/../../Service/dataTypes.h \
00018 ../MCAL/Timer\ driver/../../Service/RegisterFile.h
00019
00020 ../ECUAL/LED\ driver/LED.h:
00021
00022 ../ECUAL/LED\ driver/../../Service/ATmega32Port.h:
00023
00024 ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/DIO.h:
00025
00026 ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/ATmega32Port.h:
00027
00028 ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/BitMath.h:
00029
00030 ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/dataTypes.h:
00031
00032 ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/RegisterFile.h:
00033
00034 ../ECUAL/LED\ driver/../../MCAL/Dio\ driver/../../Service/dataTypes.h:
00035
00036 ../MCAL/Ext\ interrupt\ driver/Ext\ interrupt.h:
00037
00038 ../MCAL/Ext\ interrupt\ driver/../../Service/ATmega32Port.h:
00039
00040 ../MCAL/Ext\ interrupt\ driver/../../Service/RegisterFile.h:
00041
00042 ../MCAL/Ext\ interrupt\ driver/../../Interrupt/Interrupt.h:
00043
00044 ../MCAL/Ext\ interrupt\ driver/../../Service/BitMath.h:
00045
00046 ../MCAL/Timer\ driver/Timer_0.h:
00047
00048 ../MCAL/Timer\ driver/../../Service/BitMath.h:
00049
00050 ../MCAL/Timer\ driver/../../Service/ATmega32Port.h:
00051
00052 ../MCAL/Timer\ driver/../../Service/dataTypes.h:
00053
00054 ../MCAL/Timer\ driver/../../Service/RegisterFile.h:
```

## 4.13 Debug/MCAL/Dio driver/DIO.d File Reference

### 4.14 DIO.d

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

```
00001 MCAL/Dio driver/DIO.d MCAL/Dio driver/DIO.o: ../MCAL/Dio\ driver/DIO.c \
00002 ../MCAL/Dio\ driver/DIO.h \
00003 ../MCAL/Dio\ driver/../../Service/ATmega32Port.h \
00004 ../MCAL/Dio\ driver/../../Service/BitMath.h \
00005 ../MCAL/Dio\ driver/../../Service/dataTypes.h \
00006 ../MCAL/Dio\ driver/../../Service/RegisterFile.h \
00007 ../MCAL/Dio\ driver/../../Service/dataTypes.h \
00008 ../MCAL/Dio\ driver/../../Interrupt/Interrupt.h
```

```

00009
00010 ../MCAL/Dio\ driver/DIO.h:
00011
00012 ../MCAL/Dio\ driver/../../Service/ATmega32Port.h:
00013
00014 ../MCAL/Dio\ driver/../../Service/BitMath.h:
00015
00016 ../MCAL/Dio\ driver/../../Service/dataTypes.h:
00017
00018 ../MCAL/Dio\ driver/../../Service/RegisterFile.h:
00019
00020 ../MCAL/Dio\ driver/../../Service/dataTypes.h:
00021
00022 ../MCAL/Dio\ driver/../../Interrupt/Interrupt.h:

```

## 4.15 Debug/MCAL/Ext interrupt driver/Ext interrupt.d File Reference

## 4.16 Ext interrupt.d

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

```

00001 MCAL/Ext interrupt driver/Ext interrupt.d \
00002 MCAL/Ext interrupt driver/Ext interrupt.o: \
00003 ../MCAL/Ext\ interrupt\ driver/Ext\ interrupt.c \
00004 ../MCAL/Ext\ interrupt\ driver/Ext\ interrupt.h \
00005 ../MCAL/Ext\ interrupt\ driver/../../Service/ATmega32Port.h \
00006 ../MCAL/Ext\ interrupt\ driver/../../Service/RegisterFile.h \
00007 ../MCAL/Ext\ interrupt\ driver/../../Service/dataTypes.h \
00008 ../MCAL/Ext\ interrupt\ driver/../../Interrupt/Interrupt.h \
00009 ../MCAL/Ext\ interrupt\ driver/../../Service/BitMath.h
00010
00011 ../MCAL/Ext\ interrupt\ driver/Ext\ interrupt.h:
00012
00013 ../MCAL/Ext\ interrupt\ driver/../../Service/ATmega32Port.h:
00014
00015 ../MCAL/Ext\ interrupt\ driver/../../Service/RegisterFile.h:
00016
00017 ../MCAL/Ext\ interrupt\ driver/../../Service/dataTypes.h:
00018
00019 ../MCAL/Ext\ interrupt\ driver/../../Interrupt/Interrupt.h:
00020
00021 ../MCAL/Ext\ interrupt\ driver/../../Service/BitMath.h:

```

## 4.17 Debug/MCAL/Timer driver/Timer\_0.d File Reference

## 4.18 Timer\_0.d

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

```

00001 MCAL/Timer driver/Timer_0.d MCAL/Timer driver/Timer_0.o: \
00002 ../MCAL/Timer\ driver/Timer_0.c ../MCAL/Timer\ driver/Timer_0.h \
00003 ../MCAL/Timer\ driver/../../Service/BitMath.h \
00004 ../MCAL/Timer\ driver/../../Service/ATmega32Port.h \
00005 ../MCAL/Timer\ driver/../../Service/dataTypes.h \
00006 ../MCAL/Timer\ driver/../../Service/RegisterFile.h \
00007 ../MCAL/Timer\ driver/../../Service/dataTypes.h \
00008 c:\program\ files\ (x86)\atmel\studio\7.0\toolchain\avr8\avr8-gnu-toolchain\avr\include\math.h \
00009 ../MCAL/Timer\ driver/../../Interrupt/Interrupt.h
00010
00011 ../MCAL/Timer\ driver/Timer_0.h:
00012
00013 ../MCAL/Timer\ driver/../../Service/BitMath.h:
00014
00015 ../MCAL/Timer\ driver/../../Service/ATmega32Port.h:
00016
00017 ../MCAL/Timer\ driver/../../Service/dataTypes.h:
00018
00019 ../MCAL/Timer\ driver/../../Service/RegisterFile.h:
00020
00021 ../MCAL/Timer\ driver/../../Service/dataTypes.h:
00022
00023 c:\program\ files\ (x86)\atmel\studio\7.0\toolchain\avr8\avr8-gnu-toolchain\avr\include\math.h:
00024
00025 ../MCAL/Timer\ driver/../../Interrupt/Interrupt.h:

```

## 4.19 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.*

## 4.20 Button.c

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

```
00001
00002
00003 /******
00004  */
00005  */
00006  */
00007  */
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 }
```

## 4.21 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.*





```

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 }

```

## 4.25 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.*

## 4.26 LED.h

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

```

00001 /*****
00002  */
00003  */
00004  */
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_  */

```

## 4.27 main.c File Reference

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

### Functions

- int [main](#) (void)

#### 4.27.1 Function Documentation

##### 4.27.1.1 main()

```
int main (
    void )
```

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

## 4.28 main.c

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

```
00001
00002  /*****
00003  */
00004  */
00005  /*****
00006
00007  #include "../ECUAL/LED driver/LED.h"
00008  #include "MCAL/Ext interrupt driver/Ext interrupt.h"
00009  #include "MCAL/Timer driver/Timer_0.h"
00010  int main(void)
00011  {
00012      ledInit (PA0);
00013      ledInit (PB0);
00014      Ext_interruptInit (INT0, ANY_LOGICAL_CHANGE);
00015      Timer0_init (NORMAL, clkI_DIVISION_BY_1024);
00016      while (1)
00017      {
00018          ledOff (PB0);
00019          Timer0_delay_ms (1000);
00020          ledOn (PB0);
00021          Timer0_delay_ms (1000);
00022      }
00023      return 0;
00024  }
00025
00026
```

## 4.29 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.*
- [ISR \(EXT\\_INT0\)](#)

### 4.29.1 Function Documentation

#### 4.29.1.1 ISR()

```
ISR (
    EXT_INT0 )
```

Definition at line 230 of file [DIO.c](#).

## 4.30 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 /*****
00167 EN_pinErro_t DIO_pinRead(EN_pinNum_t pinNum,EN_pinState_t *pinState)
00168 {
00169     EN_pinErro_t error = OK;
00170     //check if the pin is located in port A
00171     if (pinNum <= PA7)
00172     {
00173         *pinState = getBit(PINA,pinNum);
00174     }
00175     //check if the pin is located in port B
00176     else if (pinNum <= PB7)
00177     {
00178         pinNum-=PORTB_OFFSET;
00179         *pinState = getBit(PINB,pinNum);
00180     }
00181     //check if the pin is located in port C
00182     else if (pinNum <= PC7)
00183     {
00184         *pinState = getBit(PINC,pinNum);
00185     }
00186     //check if the pin is located in port D
00187     else if (pinNum <= PD7)
00188     {
00189         *pinState = getBit(PIND,pinNum);
00190     }
00191     //if the pinNum is wrong
00192     else
00193     {

```

```

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
00230 ISR(EXT_INT0)
00231 {
00232     DIO_pinToggle(PA0);
00233 }

```

## 4.31 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.*

### 4.31.1 Detailed Description

Author

: Ehab Omara

Date

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

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

## 4.32 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_pinError_t DIO_pinInit(EN_pinNum_t pinNum, EN_pinDirection_t pinDirection);
00058  EN_pinError_t DIO_pinWrite(EN_pinNum_t pinNum, EN_pinState_t pinState);
00072  EN_pinError_t DIO_pinToggle(EN_pinNum_t pinNum);
00086  EN_pinError_t DIO_pinRead(EN_pinNum_t pinNum, EN_pinState_t *pinState);
00090  #endif /* DIO_H_ */
```

## 4.33 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.*

## 4.34 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
interruptSenseControl)
00010  {
00011      EN_interruptError_t interruptError = INT_OK;
00012      if (interruptNum == INT0)
00013      {
00014          //check if the value of the interruptSenseControl is correct
00015          if (interruptSenseControl >= LOW_LEVEL &&interruptSenseControl <= RISING_EDGE)
00016          {
00017              //enable INT0
00018              setBit(GICR,INT0);
00019              //clearing interruptSenseControl old value
00020              MCUCR&=~(ISC00«0x03));
00021              //setting interruptSenseControl new value
00022              MCUCR|=interruptSenseControl«ISC00;
00023              //set INT0 pin as input
00024              clrBit(DDRD,INT0_PIN);
00025          }
00026          else
00027          {
00028              interruptError = WRONG_SENSE_CONTROL;
00029          }
00030      }
00031      else if (interruptNum == INT1)
00032      {
00033          //check if the value of the interruptSenseControl is correct
00034          if (interruptSenseControl >= LOW_LEVEL &&interruptSenseControl <= RISING_EDGE)
00035          {
00036              //enable INT1
00037              setBit(GICR,INT1);
00038              //clearing interruptSenseControl old value
00039              MCUCR&=~(0x03«ISC10));
00040              //setting interruptSenseControl new value
00041              MCUCR|=interruptSenseControl«ISC10;
00042              //set INT1 pin as input
00043              clrBit(DDRD,INT1_PIN);
00044          }
00045          else
00046          {
00047              interruptError = WRONG_SENSE_CONTROL;
00048          }
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          else
00071          {
00072              interruptError = WRONG_SENSE_CONTROL;
00073          }
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

```

## 4.35 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.

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

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

## 4.36 Ext interrupt.h

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

```

00001
00002  /* ***** Author : Ehab Omara ***** */
00003  /* ***** Date : 8/13/2022 4:39:49 AM ***** */
00004  /* ***** File name: Ext interrupt.h ***** */
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 */

```

## 4.37 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.*

## 4.38 Interrupt.h

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

```

00001  /*****
00002  /*                                     Author   :   Ehab Omara
00003  /*                                     Date      :   8/13/2022 1:08:16 AM
00004  /*                                     File name:   Interrupt.h
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_ */

```

## 4.39 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\\_interruptDiable](#) ([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](#) = [clkI\\_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.*

## 4.39.1 Function Documentation

### 4.39.1.1 ISR()

```
ISR (
    TIM0\_OVF )
```

Definition at line 146 of file [Timer\\_0.c](#).

## 4.39.2 Variable Documentation

### 4.39.2.1 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](#).

### 4.39.2.2 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](#).

#### 4.39.2.3 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](#).

#### 4.39.2.4 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](#).

### 4.40 Timer\_0.c

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

```
00001
00002 /*****
00003 */
00004 */
00005
00006 #include "Timer_0.h"
00007 #include <math.h>
00008 #include "../Interrupt/Interrupt.h"
00015 static EN_Timer0_clkSource_t Timer0_globalClkSource = clkI_No_DIVISON;
00016
00025 static uint32_t volatile Timer0_globalNumOfOverFlows = 0;
00026
00033 static float64_t volatile Timer0_globalOverflowTime = 0;
00040 static float64_t volatile Timer0_globalTickTime = 0;
00041
00042 En_Timer0_Error_t Timer0_init(EN_Timer0_Mode_t Timer0_mode, EN_Timer0_clkSource_t Timer0_clkSource)
00043 {
00044     En_Timer0_Error_t Timer0_error = TIMER0_OK;
00045     //selecting Timer 0 mode
00046     if (Timer0_mode == NORMAL || Timer0_mode == PWM_PHASE_CORRECR || Timer0_mode == CTC || Timer0_mode
== FAST_PWM)
00047     {
00048
00049         //clear the old mode value
00050         TCCR0 &= CLR_TIMER0_MODE;
00051         //set the new mode value
00052         TCCR0 |= Timer0_mode;
00053     }
00054     else
00055     {
00056         Timer0_error = TIMER0_WRONG_MODE;
```

```

00057     }
00058     //selecting Timer 0 clock source
00059     if (Timer0_clkSource >= NO_CLOCK_SOURCE && Timer0_clkSource <= EXTERNAL_CLOCK_RISING_EDGE)
00060     {
00061         Timer0_globalClkSource = Timer0_clkSource;
00062
00063         uint16_t Timer0_localClkPrescaler[] = {0,1,8,64,256,1024};
00064         //calculate Timer 0 tick time
00065         Timer0_globalTickTime = (float64_t)Timer0_localClkPrescaler[Timer0_clkSource]/SYSTEM_CLK;
00066         //calculate Timer 0 over flow time
00067         Timer0_globalOverFlowTime = Timer0_globalTickTime * TIMER0_NUM_OF_TICKS;
00068     }
00069     else
00070     {
00071         Timer0_error = TIMER0_WRONG_CLK_SOURCE;
00072     }
00073     return Timer0_error;
00074 }
00075 }
00076
00077 void Timer0_start(void)
00078 {
00079     //clear the old clock source value
00080     TCCR0 &= CLR_TIMER0_CLK_SRC;
00081     //set the new clock source value
00082     TCCR0 |= Timer0_globalClkSource << CS00;
00083 }
00084
00085 void Timer0_stop(void)
00086 {
00087     //clear the value of Timer 0 clock source
00088     //this is done by clearing the three bits #CS00, #CS01 and #CS02
00089     TCCR0 &= CLR_TIMER0_CLK_SRC;
00090 }
00091
00092 void Timer0_reset(void)
00093 {
00094     TCNT0 = 0x00;
00095     Timer0_globalNumOfOverFlows = 0;
00096 }
00097
00098 En_Timer0_Error_t Timer0_interruptDiable(TIMER0_interrupt_t Timer0_interrupt)
00099 {
00100     En_Timer0_Error_t Timer0_error = TIMER0_OK;
00101     if (Timer0_interrupt == TIMER0_OVER_FLOW_INT || Timer0_interrupt == TIMER0_OUT_CMP_MATCH_INT)
00102     {
00103         clrBit(TIMSK,Timer0_interrupt);
00104     }
00105     else
00106     {
00107         Timer0_error = TIMER0_WRONG_INT;
00108     }
00109     return Timer0_error;
00110 }
00111
00112 En_Timer0_Error_t Timer0_interruptEnable(TIMER0_interrupt_t Timer0_interrupt)
00113 {
00114     En_Timer0_Error_t Timer0_error = TIMER0_OK;
00115     if (Timer0_interrupt == TIMER0_OVER_FLOW_INT || Timer0_interrupt == TIMER0_OUT_CMP_MATCH_INT)
00116     {
00117         setBit(TIMSK,Timer0_interrupt);
00118     }
00119     else
00120     {
00121         Timer0_error = TIMER0_WRONG_INT;
00122     }
00123     return Timer0_error;
00124 }
00125
00126 void Timer0_delay_ms(uint32_t delay_ms)
00127 {
00128     //reset Timer 0
00129     Timer0_reset();
00130     //convert delay time from mile seconds to seconds
00131     float64_t neededTimeInsecond = (float64_t)delay_ms/1000;
00132     //calculate number of over flows needed to reach the desired time
00133     uint32_t numberOfoverFlows = ceil(neededTimeInsecond/Timer0_globalOverFlowTime);
00134     //calculate the initial value for #TCNT0 register
00135     TCNT0 = TIMER0_NUM_OF_TICKS - (neededTimeInsecond/Timer0_globalTickTime)/numberOfoverFlows;
00136     //enable Timer 0 over flow interrupt
00137     Timer0_interruptEnable(TIMER0_OVER_FLOW_INT);

```

```

00138 //start Timer 0 to count
00139 Timer0_start();
00140 //wait until reaching needed number over flows
00141 while(Timer0_globalNumOfOverFlows < numberOfoverFlows);
00142 //stop Timer 0 after reaching the desired time.
00143 Timer0_stop();
00144 }
00145
00146 ISR(TIM0_OVF)
00147 {
00148     Timer0_globalNumOfOverFlows++;
00149 }

```

## 4.41 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 **SYSTEM\_CLK** 1000000UL  
*System clock Macro.*
- #define **TIMER0\_NUM\_OF\_TICKS** 256  
*Number of Ticks.*
- #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 prescaler).
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	<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\\_EDGE](#) }
- 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\\_interruptDisable](#) ([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](#) }

## 4.42 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 #ifndef SYSTEM_CLK
00018 /* prevent compiler error by supplying a default */
00019 # warning "SYSTEM_CLK not defined for Timer_0.h, default value is 1MHz"
00020 #define SYSTEM_CLK 1000000UL
00021 #endif
00022
00023 #define TIMER0_NUM_OF_TICKS 256
00024
00025
00026 #define CS00 0
00027 #define CS01 1
00028 #define CS02 2
00029
00030 /*****
00031 #define CLR_TIMER0_CLK_SRC 0xF8
00032
00033 /*****
00034 #define WGM00 3
00035 #define WGM01 6
00036
00037 /*****
00038 #define CLR_TIMER0_MODE 0xB7
00039
00040 /*****
00041 #define TOIE0 0
00042 #define OCIE0 1
00043 typedef enum
00044 {
00045     TIMER0_OVER_FLOW_INT,
00046     TIMER0_OUT_CMP_MATCH_INT
00047 }TIMER0_interrupt_t;
00048
00049 /*****
00050 #define TOV0 0
00051 #define OCF0 1
00052
00053 /*****
00054 typedef enum
00055 {
00056     NORMAL = 0,
00057     PWM_PHASE_CORRECR=8,
00058     CTC=64,
00059     FAST_PWM=72
00060 }EN_Timer0_Mode_t;
00061
00062 /*****
00063 typedef enum
00064 {
00065     NO_CLOCK_SOURCE,
00066     clkI_No_DIVISON,
00067     clkI_DIVISION_BY_8,
00068     clkI_DIVISION_BY_64,
00069     clkI_DIVISION_BY_256,
00070     clkI_DIVISION_BY_1024,
00071     EXTERNAL_CLOCK_FALLING_EDGE,
00072     EXTERNAL_CLOCK_RISING_EDGE
00073 }EN_Timer0_clkSource_t;
00074
00075 /*****
00076 typedef enum

```

```

00205 {
00206     TIMER0_OK,
00207     TIMER0_WRONG_MODE,
00208     TIMER0_WRONG_CLK_SOURCE,
00209     TIMER0_WRONG_INT
00210 }En_Timer0_Error_t;
00211
00212 /*****
00223 En_Timer0_Error_t Timer0_interruptEnable(TIMER0_interrupt_t Timer0_interrupt);
00224
00225 /*****
00236 En_Timer0_Error_t Timer0_interruptDiable(TIMER0_interrupt_t Timer0_interrupt);
00237
00238 /*****
00253 En_Timer0_Error_t Timer0_init(EN_Timer0_Mode_t Timer0_mode,EN_Timer0_clkSource_t Timer0_clkSource);
00254
00255 /*****
00263 void Timer0_start(void);
00264
00265 /*****
00273 void Timer0_stop(void);
00274
00275 /*****
00283 void Timer0_reset(void);
00284
00285 /*****
00293 void Timer0_delay_ms(uint32_t delay_ms);
00295 #endif /* TIMER_0_H_ */

```

#### 4.43 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` }

#### 4.44 ATmega32Port.h

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

[illegible]

```

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
00067  typedef enum
00068  {
00069      Low,
00070      High
00071  }EN_pinState_t;
00072  typedef enum
00073  {
00074      Input,
00075      Output
00076  }EN_pinDirection_t;
00077  typedef enum
00078  {
00079      OK,
00080      WRONG_PIN_NUM,
00081      WRONG_PIN_DIR,
00082      WRONG_PIN_STATE
00083  }EN_pinErro_t;
00087  #endif /* ATMEGA32PORT_H_ */

```

## 4.45 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.*

## 4.46 BitMath.h

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

```
00001
00002 /*****
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_ */
```

## 4.47 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`

## 4.48 dataType.h

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

```
00001
00002 /*****
00003  * Author      : Ehab Omara
00004  * Date       : 8/10/2022 12:06:28 PM
00005  * File name:  dataType.h
00006  */
00007 #ifndef DATATYPES_H_
00008 #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_ */
```

## 4.49 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))`

## 4.50 RegisterFile.h

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

```
00001
00002 /***** Author : Ehab Omara *****/
00003 /* Date : 8/10/2022 12:06:56 PM */
00004 /* File name: RegisterFile.h */
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
00046 /***** Port A registers *****/
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 *****/
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
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
00156 #define PORTD      (*(volatile uint8_t*)0x32))
00164 #define DDRD       (*(volatile uint8_t*)0x31))
00172 #define PIND       (*(volatile uint8_t*)0x30))
00174 /***** Interrupts registers
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
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, [18](#)
- ANA\_COMP
  - ATMEGA32 interrupts definitions, [18](#)
- ANY\_LOGICAL\_CHANGE
  - ATMEGA32 external interrupts driver, [16](#)
- App/app.c, [59](#)
- App/app.h, [59](#)
- ATMEGA32 external interrupts driver, [12](#)
  - ANY\_LOGICAL\_CHANGE, [16](#)
  - EN\_interruptError\_t, [15](#)
  - EN\_interruptNum\_t, [16](#)
  - EN\_interruptSenseControl\_t, [16](#)
  - Ext\_interruptInit, [17](#)
  - FALLING\_EDGE, [16](#)
  - INT0, [16](#)
  - INT0\_PIN, [13](#)
  - INT1, [16](#)
  - INT1\_PIN, [14](#)
  - INT2, [16](#)
  - INT2\_PIN, [14](#)
  - INT\_OK, [15](#)
  - ISC00, [14](#)
  - ISC01, [14](#)
  - ISC10, [14](#)
  - ISC11, [15](#)
  - ISC2, [15](#)
  - LOW\_LEVEL, [16](#)
  - RISING\_EDGE, [16](#)
  - WRONG\_INT\_NUM, [15](#)
  - WRONG\_SENSE\_CONTROL, [15](#)
- ATMEGA32 interrupts definitions, [17](#)
  - ADC, [18](#)
  - ANA\_COMP, [18](#)
  - cli, [19](#)
  - EE\_RDY, [19](#)
  - EXT\_INT0, [19](#)
  - EXT\_INT1, [19](#)
  - EXT\_INT2, [20](#)
  - ISR, [20](#)
  - sei, [20](#)
  - SPI\_STC, [20](#)
  - SPM\_RDY, [21](#)
  - TIM0\_COMP, [21](#)
  - TIM0\_OVF, [21](#)
  - TIM1\_CAPT, [21](#)
  - TIM1\_COMPA, [21](#)
  - TIM1\_COMPB, [22](#)
  - TIM1\_OVF, [22](#)
  - TIM2\_COMP, [22](#)
  - TIM2\_OVF, [22](#)
  - TWI, [22](#)
  - USART\_RXC, [23](#)
  - USART\_TXC, [23](#)
  - USART\_UDRE, [23](#)
- Bit math, [38](#)
  - clrBit, [39](#)
  - getBit, [39](#)
  - setBit, [40](#)
  - toggleBit, [40](#)
- Button driver, [5](#)
  - buttonInit, [5](#)
  - buttonRead, [6](#)
- buttonInit
  - Button driver, [5](#)
- buttonRead
  - Button driver, [6](#)
- cli
  - ATMEGA32 interrupts definitions, [19](#)
- clk\_DIVISION\_BY\_1024
  - Timer0 driver, [29](#)
- clk\_DIVISION\_BY\_256
  - Timer0 driver, [29](#)
- clk\_DIVISION\_BY\_64
  - Timer0 driver, [29](#)
- clk\_DIVISION\_BY\_8
  - Timer0 driver, [29](#)
- clk\_No\_DIVISION
  - Timer0 driver, [29](#)
- CLR\_TIMER0\_CLK\_SRC
  - Timer0 driver, [26](#)
- CLR\_TIMER0\_MODE
  - Timer0 driver, [26](#)
- clrBit
  - Bit math, [39](#)
- CS00
  - Timer0 driver, [26](#)
- CS01
  - Timer0 driver, [26](#)
- CS02
  - Timer0 driver, [27](#)
- CTC
  - Timer0 driver, [31](#)
- DDRA
  - Port A registers, [44](#)
- DDRB

- Port B registers, [46](#)
- DDRC
  - Port C registers, [47](#)
- DDRD
  - Port D registers, [48](#)
- Debug/App/app.d, [60](#)
- Debug/ECUAL/Button driver/Button.d, [60](#)
- Debug/ECUAL/LED driver/LED.d, [60](#)
- Debug/main.d, [61](#)
- Debug/MCAL/Dio driver/DIO.d, [61](#)
- Debug/MCAL/Ext interrupt driver/Ext interrupt.d, [62](#)
- Debug/MCAL/Timer driver/Timer\_0.d, [62](#)
- Definition of data types, [41](#)
  - float128\_t, [41](#)
  - float32\_t, [41](#)
  - float64\_t, [41](#)
  - sint16\_t, [42](#)
  - sint32\_t, [42](#)
  - sint8\_t, [42](#)
  - uint16\_t, [42](#)
  - uint32\_t, [42](#)
  - uint8\_t, [43](#)
- DIO driver, [9](#)
  - DIO\_pinInit, [10](#)
  - DIO\_pinRead, [10](#)
  - DIO\_pinToggle, [11](#)
  - DIO\_pinWrite, [11](#)
- DIO.c
  - ISR, [67](#)
- DIO\_pinInit
  - DIO driver, [10](#)
- DIO\_pinRead
  - DIO driver, [10](#)
- DIO\_pinToggle
  - DIO driver, [11](#)
- DIO\_pinWrite
  - DIO driver, [11](#)
- ECUAL layer, [5](#)
- ECUAL/Button driver/Button.c, [63](#)
- ECUAL/Button driver/Button.h, [63](#), [64](#)
- ECUAL/LED driver/LED.c, [64](#)
- ECUAL/LED driver/LED.h, [65](#)
- EE\_RDY
  - ATMEGA32 interrupts definitions, [19](#)
- EN\_interruptError\_t
  - ATMEGA32 external interrupts driver, [15](#)
- EN\_interruptNum\_t
  - ATMEGA32 external interrupts driver, [16](#)
- EN\_interruptSenseControl\_t
  - ATMEGA32 external interrupts driver, [16](#)
- EN\_pinDirection\_t
  - MCU ports, [36](#)
- EN\_pinErro\_t
  - MCU ports, [37](#)
- EN\_pinNum\_t
  - MCU ports, [37](#)
- EN\_pinState\_t
  - MCU ports, [38](#)
- EN\_Timer0\_clkSource\_t
  - Timer0 driver, [29](#)
- En\_Timer0\_Error\_t
  - Timer0 driver, [29](#)
- EN\_Timer0\_Mode\_t
  - Timer0 driver, [30](#)
- EXT\_INT0
  - ATMEGA32 interrupts definitions, [19](#)
- EXT\_INT1
  - ATMEGA32 interrupts definitions, [19](#)
- EXT\_INT2
  - ATMEGA32 interrupts definitions, [20](#)
- Ext\_interruptInit
  - ATMEGA32 external interrupts driver, [17](#)
- EXTERNAL\_CLOCK\_FALLING\_EDGE
  - Timer0 driver, [29](#)
- EXTERNAL\_CLOCK\_RISING\_EDGE
  - Timer0 driver, [29](#)
- FALLING\_EDGE
  - ATMEGA32 external interrupts driver, [16](#)
- FAST\_PWM
  - Timer0 driver, [31](#)
- float128\_t
  - Definition of data types, [41](#)
- float32\_t
  - Definition of data types, [41](#)
- float64\_t
  - Definition of data types, [41](#)
- General Timers registers, [55](#)
  - TIFR, [55](#)
  - TIMSK, [56](#)
- getBit
  - Bit math, [39](#)
- GICR
  - Interrupt registers, [50](#)
- GIFR
  - Interrupt registers, [50](#)
- High
  - MCU ports, [38](#)
- I/O registers, [44](#)
- Input
  - MCU ports, [37](#)
- INT0
  - ATMEGA32 external interrupts driver, [16](#)
- INT0\_PIN
  - ATMEGA32 external interrupts driver, [13](#)
- INT1
  - ATMEGA32 external interrupts driver, [16](#)
- INT1\_PIN
  - ATMEGA32 external interrupts driver, [14](#)
- INT2
  - ATMEGA32 external interrupts driver, [16](#)
- INT2\_PIN
  - ATMEGA32 external interrupts driver, [14](#)
- INT\_OK

- ATMEGA32 external interrupts driver, [15](#)
- Interrupt registers, [49](#)
  - GICR, [50](#)
  - GIFR, [50](#)
  - MCUCR, [50](#)
  - MCUCSR, [51](#)
- Interrupts driver, [17](#)
- ISC00
  - ATMEGA32 external interrupts driver, [14](#)
- ISC01
  - ATMEGA32 external interrupts driver, [14](#)
- ISC10
  - ATMEGA32 external interrupts driver, [14](#)
- ISC11
  - ATMEGA32 external interrupts driver, [15](#)
- ISC2
  - ATMEGA32 external interrupts driver, [15](#)
- ISR
  - ATMEGA32 interrupts definitions, [20](#)
  - DIO.c, [67](#)
  - Timer\_0.c, [77](#)
- LED driver, [6](#)
  - ledInit, [7](#)
  - ledOff, [7](#)
  - ledOn, [8](#)
  - ledToggle, [8](#)
- ledInit
  - LED driver, [7](#)
- ledOff
  - LED driver, [7](#)
- ledOn
  - LED driver, [8](#)
- ledToggle
  - LED driver, [8](#)
- Low
  - MCU ports, [38](#)
- LOW\_LEVEL
  - ATMEGA32 external interrupts driver, [16](#)
- main
  - main.c, [66](#)
- main.c, [66](#)
  - main, [66](#)
- MCAL layer, [9](#)
- MCAL/Dio driver/DIO.c, [67](#)
- MCAL/Dio driver/DIO.h, [70](#), [71](#)
- MCAL/Ext interrupt driver/Ext interrupt.c, [71](#), [72](#)
- MCAL/Ext interrupt driver/Ext interrupt.h, [73](#), [74](#)
- MCAL/Interrupt/Interrupt.h, [75](#), [76](#)
- MCAL/Timer driver/Timer\_0.c, [76](#), [78](#)
- MCAL/Timer driver/Timer\_0.h, [80](#), [82](#)
- MCU ports, [35](#)
  - EN\_pinDirection\_t, [36](#)
  - EN\_pinErro\_t, [37](#)
  - EN\_pinNum\_t, [37](#)
  - EN\_pinState\_t, [38](#)
  - High, [38](#)
  - Input, [37](#)
  - Low, [38](#)
  - OK, [37](#)
  - Output, [37](#)
  - PA0, [37](#)
  - PA1, [37](#)
  - PA2, [37](#)
  - PA3, [37](#)
  - PA4, [37](#)
  - PA5, [37](#)
  - PA6, [37](#)
  - PA7, [37](#)
  - PB0, [37](#)
  - PB1, [37](#)
  - PB2, [37](#)
  - PB3, [37](#)
  - PB4, [37](#)
  - PB5, [37](#)
  - PB6, [37](#)
  - PB7, [37](#)
  - PC0, [38](#)
  - PC1, [38](#)
  - PC2, [38](#)
  - PC3, [38](#)
  - PC4, [38](#)
  - PC5, [38](#)
  - PC6, [38](#)
  - PC7, [38](#)
  - PD0, [38](#)
  - PD1, [38](#)
  - PD2, [38](#)
  - PD3, [38](#)
  - PD4, [38](#)
  - PD5, [38](#)
  - PD6, [38](#)
  - PD7, [38](#)
  - PORTA\_OFFSET, [35](#)
  - PORTB\_OFFSET, [36](#)
  - PORTC\_OFFSET, [36](#)
  - PORTD\_OFFSET, [36](#)
  - WRONG\_PIN\_DIR, [37](#)
  - WRONG\_PIN\_NUM, [37](#)
  - WRONG\_PIN\_STATE, [37](#)
- MCU Registers, [43](#)
- MCUCR
  - Interrupt registers, [50](#)
- MCUCSR
  - Interrupt registers, [51](#)
- NO\_CLOCK\_SOURCE
  - Timer0 driver, [29](#)
- NORMAL
  - Timer0 driver, [31](#)
- OCF0
  - Timer0 driver, [27](#)
- OCIE0
  - Timer0 driver, [27](#)
- OCR0
  - Timer0 Registers, [52](#)

- OK
  - MCU ports, [37](#)
- Output
  - MCU ports, [37](#)
- PA0
  - MCU ports, [37](#)
- PA1
  - MCU ports, [37](#)
- PA2
  - MCU ports, [37](#)
- PA3
  - MCU ports, [37](#)
- PA4
  - MCU ports, [37](#)
- PA5
  - MCU ports, [37](#)
- PA6
  - MCU ports, [37](#)
- PA7
  - MCU ports, [37](#)
- PB0
  - MCU ports, [37](#)
- PB1
  - MCU ports, [37](#)
- PB2
  - MCU ports, [37](#)
- PB3
  - MCU ports, [37](#)
- PB4
  - MCU ports, [37](#)
- PB5
  - MCU ports, [37](#)
- PB6
  - MCU ports, [37](#)
- PB7
  - MCU ports, [37](#)
- PC0
  - MCU ports, [38](#)
- PC1
  - MCU ports, [38](#)
- PC2
  - MCU ports, [38](#)
- PC3
  - MCU ports, [38](#)
- PC4
  - MCU ports, [38](#)
- PC5
  - MCU ports, [38](#)
- PC6
  - MCU ports, [38](#)
- PC7
  - MCU ports, [38](#)
- PD0
  - MCU ports, [38](#)
- PD1
  - MCU ports, [38](#)
- PD2
  - MCU ports, [38](#)
- PD3
  - MCU ports, [38](#)
- PD4
  - MCU ports, [38](#)
- PD5
  - MCU ports, [38](#)
- PD6
  - MCU ports, [38](#)
- PD7
  - MCU ports, [38](#)
- PINA
  - Port A registers, [45](#)
- PINB
  - Port B registers, [46](#)
- PINC
  - Port C registers, [47](#)
- PIND
  - Port D registers, [49](#)
- Port A registers, [44](#)
  - DDRA, [44](#)
  - PINA, [45](#)
  - PORTA, [45](#)
- Port B registers, [45](#)
  - DDRB, [46](#)
  - PINB, [46](#)
  - PORTB, [46](#)
- Port C registers, [47](#)
  - DDRC, [47](#)
  - PINC, [47](#)
  - PORTC, [48](#)
- Port D registers, [48](#)
  - DDRD, [48](#)
  - PIND, [49](#)
  - PORTD, [49](#)
- PORTA
  - Port A registers, [45](#)
- PORTA\_OFFSET
  - MCU ports, [35](#)
- PORTB
  - Port B registers, [46](#)
- PORTB\_OFFSET
  - MCU ports, [36](#)
- PORTC
  - Port C registers, [48](#)
- PORTC\_OFFSET
  - MCU ports, [36](#)
- PORTD
  - Port D registers, [49](#)
- PORTD\_OFFSET
  - MCU ports, [36](#)
- PWM\_PHASE\_CORRECR
  - Timer0 driver, [31](#)
- RISING\_EDGE
  - ATMEGA32 external interrupts driver, [16](#)
- sei
  - ATMEGA32 interrupts definitions, [20](#)
- Service layer, [43](#)

Service/ATmega32Port.h, [83](#)  
 Service/BitMath.h, [84](#), [85](#)  
 Service/dataTypes.h, [85](#)  
 Service/RegisterFile.h, [86](#)  
 setBit  
     Bit math, [40](#)  
 sint16\_t  
     Definition of data types, [42](#)  
 sint32\_t  
     Definition of data types, [42](#)  
 sint8\_t  
     Definition of data types, [42](#)  
 SPI\_STC  
     ATMEGA32 interrupts definitions, [20](#)  
 SPM\_RDY  
     ATMEGA32 interrupts definitions, [21](#)  
 SYSTEM\_CLK  
     Timer0 driver, [27](#)  
  
 TCCR0  
     Timer0 Registers, [52](#)  
 TCNT0  
     Timer0 Registers, [55](#)  
 TIFR  
     General Timers registers, [55](#)  
 TIM0\_COMP  
     ATMEGA32 interrupts definitions, [21](#)  
 TIM0\_OVF  
     ATMEGA32 interrupts definitions, [21](#)  
 TIM1\_CAPT  
     ATMEGA32 interrupts definitions, [21](#)  
 TIM1\_COMPA  
     ATMEGA32 interrupts definitions, [21](#)  
 TIM1\_COMPB  
     ATMEGA32 interrupts definitions, [22](#)  
 TIM1\_OVF  
     ATMEGA32 interrupts definitions, [22](#)  
 TIM2\_COMP  
     ATMEGA32 interrupts definitions, [22](#)  
 TIM2\_OVF  
     ATMEGA32 interrupts definitions, [22](#)  
 Timer0 driver, [24](#)  
     clkI\_DIVISION\_BY\_1024, [29](#)  
     clkI\_DIVISION\_BY\_256, [29](#)  
     clkI\_DIVISION\_BY\_64, [29](#)  
     clkI\_DIVISION\_BY\_8, [29](#)  
     clkI\_No\_DIVISION, [29](#)  
     CLR\_TIMER0\_CLK\_SRC, [26](#)  
     CLR\_TIMER0\_MODE, [26](#)  
     CS00, [26](#)  
     CS01, [26](#)  
     CS02, [27](#)  
     CTC, [31](#)  
     EN\_Timer0\_clkSource\_t, [29](#)  
     En\_Timer0\_Error\_t, [29](#)  
     EN\_Timer0\_Mode\_t, [30](#)  
     EXTERNAL\_CLOCK\_FALLING\_EDGE, [29](#)  
     EXTERNAL\_CLOCK\_RISING\_EDGE, [29](#)  
     FAST\_PWM, [31](#)  
     NO\_CLOCK\_SOURCE, [29](#)  
     NORMAL, [31](#)  
     OCF0, [27](#)  
     OCIE0, [27](#)  
     PWM\_PHASE\_CORRECR, [31](#)  
     SYSTEM\_CLK, [27](#)  
     Timer0\_delay\_ms, [31](#)  
     Timer0\_init, [32](#)  
     TIMER0\_interrupt\_t, [31](#)  
     Timer0\_interruptDiable, [32](#)  
     Timer0\_interruptEnable, [33](#)  
     TIMER0\_NUM\_OF\_TICKS, [27](#)  
     TIMER0\_OK, [30](#)  
     TIMER0\_OUT\_CMP\_MATCH\_INT, [31](#)  
     TIMER0\_OVER\_FLOW\_INT, [31](#)  
     Timer0\_reset, [33](#)  
     Timer0\_start, [34](#)  
     Timer0\_stop, [34](#)  
     TIMER0\_WRONG\_CLK\_SOURCE, [30](#)  
     TIMER0\_WRONG\_INT, [30](#)  
     TIMER0\_WRONG\_MODE, [30](#)  
     TOIE0, [28](#)  
     TOV0, [28](#)  
     WGM00, [28](#)  
     WGM01, [28](#)  
 Timer0 Registers, [52](#)  
     OCR0, [52](#)  
     TCCR0, [52](#)  
     TCNT0, [55](#)  
 Timer0\_delay\_ms  
     Timer0 driver, [31](#)  
 Timer0\_globalClkSource  
     Timer\_0.c, [77](#)  
 Timer0\_globalNumOfOverFlows  
     Timer\_0.c, [77](#)  
 Timer0\_globalOverflowTime  
     Timer\_0.c, [77](#)  
 Timer0\_globalTickTime  
     Timer\_0.c, [78](#)  
 Timer0\_init  
     Timer0 driver, [32](#)  
 TIMER0\_interrupt\_t  
     Timer0 driver, [31](#)  
 Timer0\_interruptDiable  
     Timer0 driver, [32](#)  
 Timer0\_interruptEnable  
     Timer0 driver, [33](#)  
 TIMER0\_NUM\_OF\_TICKS  
     Timer0 driver, [27](#)  
 TIMER0\_OK  
     Timer0 driver, [30](#)  
 TIMER0\_OUT\_CMP\_MATCH\_INT  
     Timer0 driver, [31](#)  
 TIMER0\_OVER\_FLOW\_INT  
     Timer0 driver, [31](#)  
 Timer0\_reset  
     Timer0 driver, [33](#)  
 Timer0\_start

- Timer0 driver, [34](#)
- Timer0\_stop
  - Timer0 driver, [34](#)
- TIMER0\_WRONG\_CLK\_SOURCE
  - Timer0 driver, [30](#)
- TIMER0\_WRONG\_INT
  - Timer0 driver, [30](#)
- TIMER0\_WRONG\_MODE
  - Timer0 driver, [30](#)
- Timer\_0.c
  - ISR, [77](#)
  - Timer0\_globalClkSource, [77](#)
  - Timer0\_globalNumOfOverFlows, [77](#)
  - Timer0\_globalOverflowTime, [77](#)
  - Timer0\_globalTickTime, [78](#)
- Timers driver, [23](#)
- Timers Registers, [52](#)
- TIMSK
  - General Timers registers, [56](#)
- toggleBit
  - Bit math, [40](#)
- TOIE0
  - Timer0 driver, [28](#)
- TOV0
  - Timer0 driver, [28](#)
- TWI
  - ATMEGA32 interrupts definitions, [22](#)
- uint16\_t
  - Definition of data types, [42](#)
- uint32\_t
  - Definition of data types, [42](#)
- uint8\_t
  - Definition of data types, [43](#)
- USART\_RXC
  - ATMEGA32 interrupts definitions, [23](#)
- USART\_TXC
  - ATMEGA32 interrupts definitions, [23](#)
- USART\_UDRE
  - ATMEGA32 interrupts definitions, [23](#)
- WGM00
  - Timer0 driver, [28](#)
- WGM01
  - Timer0 driver, [28](#)
- WRONG\_INT\_NUM
  - ATMEGA32 external interrupts driver, [15](#)
- WRONG\_PIN\_DIR
  - MCU ports, [37](#)
- WRONG\_PIN\_NUM
  - MCU ports, [37](#)
- WRONG\_PIN\_STATE
  - MCU ports, [37](#)
- WRONG\_SENSE\_CONTROL
  - ATMEGA32 external interrupts driver, [15](#)