On-Demand Traffic Control

Generated by Doxygen 1.9.4

Chapter 1

System static architecture

1.1 system description

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

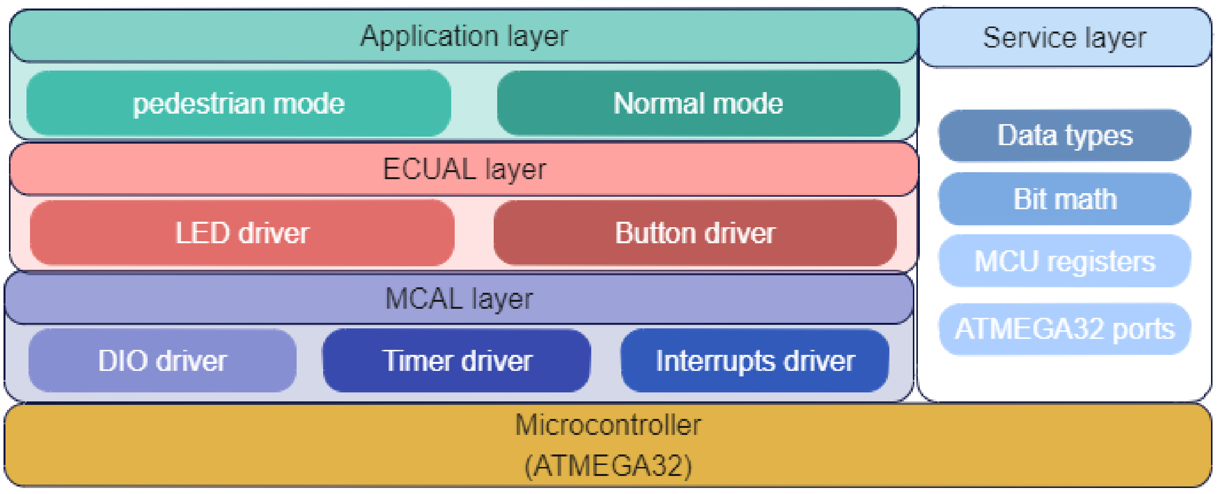
1.1.1 Hardware

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

1.1.2 Software & IDEs

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

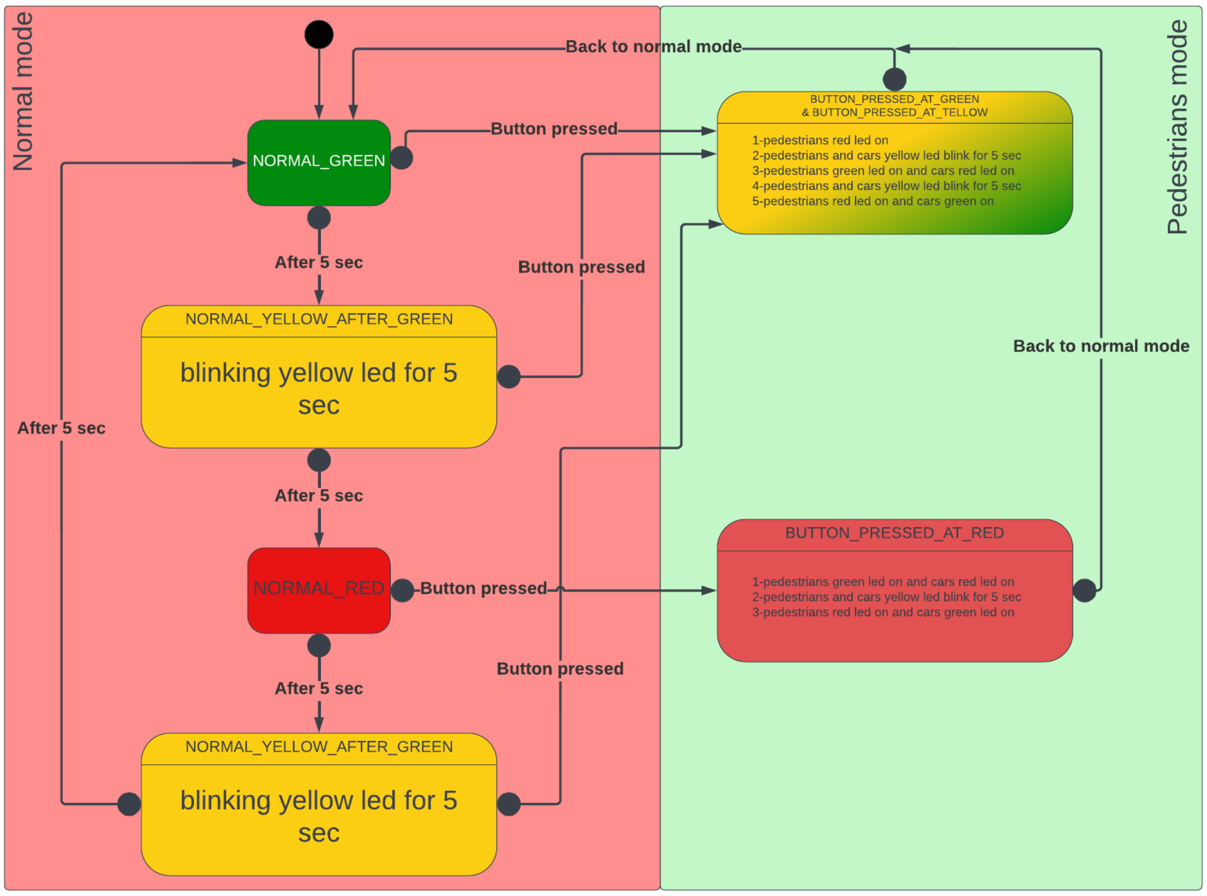
1.2 system design



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

1.3 State machine

- This state machine describes the flow of the system
- It consist of 7 main states
  NORMAL_GREEN
  NORMAL_YELLOW_AFTER_GREEN
  NORMAL_RED
  NORMAL_YELLOW_AFTER_RED
  BUTTON_PRESSED_AT_GREEN
  BUTTON_PRESSED_AT_YELLOW
  BUTTON_PRESSED_AT_RED


Normal mode
Pedestrians mode
Back to normal mode
NORMAL_GREEN
Button pressed
BUTTON_PRESSED_AT_GREEN
& BUTTON_PRESSED_AT_TELLOW
1-pedestrians red led on
2-pedestrians and cars yellow led blink for 5 sec
3-pedestrians green led on and cars red led on
4-pedestrians and cars yellow led blink for 5 sec
5-pedestrians red led on and cars green on
After 5 sec
Button pressed
NORMAL_YELLOW_AFTER_GREEN
blinking yellow led for 5 sec
Back to normal mode
After 5 sec
After 5 sec
NORMAL_RED
Button pressed
BUTTON_PRESSED_AT_RED
1-pedestrians green led on and cars red led on
2-pedestrians and cars yellow led blink for 5 sec
3-pedestrians red led on and cars green led on
After 5 sec
Button pressed
NORMAL_YELLOW_AFTER_GREEN
blinking yellow led for 5 sec

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Module Documentation

4.1 Application layer

Enumerations

- enum EN_TrafficStates_t {
  NORMAL_GREEN , NORMAL_YELLOW_AFTER_GREEN , NORMAL_RED , NORMAL_YELLOW_AFTER_RED
  ,
  BUTTON_PRESSED_AT_GREEN , BUTTON_PRESSED_AT_RED , BUTTON_PRESSED_AT_YELLOW }

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

Functions

- void app_start (void)

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

Cars led control function

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

Parameters

`in`	*None*	this function doesn't take any input parameters.
`out`	*None*	this function doesn't take any output parameters.

Return values

None	This function doesn't return anything.

- void carsGreenState (void)

  carsGreenState

- void carsYellowState (void)

    carsYellowState
- void carsRedState (void)

    carsRedState

Pedestrians led control function

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

Parameters

`in`	*None*	this function doesn't take any input parameters.
`out`	*None*	this function doesn't take any output parameters.

Return values

None	This function doesn't return anything.

- void pedestriansGreenState (void)

    pedestriansGreenState
- void pedestriansYellowState (void)

    pedestriansYellowState
- void pedestriansRedState (void)

    pedestriansRedState

Cars LEDS Macros

These macros represent the pins which cars leds connected to.

- #define CARS_GREEN_LED PA0
- #define CARS_YELOW_LED PA1
- #define CARS_RED_LED PA2

Pedestrians LEDS Macros

These macros represent the pins which pedestrians leds connected to.

- #define PEDESTRIANS_GREEN_LED PB0
- #define PEDESTRIANS_YELOW_LED PB1
- #define PEDESTRIANS_RED_LED PB2

4.1.1 Detailed Description

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

4.1.2 Macro Definition Documentation

4.1.2.1 CARS_GREEN_LED

```
#define CARS_GREEN_LED PA0
```

This is a cars green led pin

Definition at line 23 of file app.h.

4.1.2.2 CARS_RED_LED

```
#define CARS_RED_LED PA2
```

This is a cars red led pin

Definition at line 25 of file app.h.

4.1.2.3 CARS_YELOW_LED

```
#define CARS_YELOW_LED PA1
```

This is a cars yellow led pin

Definition at line 24 of file app.h.

4.1.2.4 PEDESTRIANS_GREEN_LED

```
#define PEDESTRIANS_GREEN_LED PB0
```

This is a pedestrians green led pin

Definition at line 34 of file app.h.

4.1.2.5 PEDESTRIANS_RED_LED

```
#define PEDESTRIANS_RED_LED PB2
```

This is a pedestrians red led pin

Definition at line 36 of file app.h.

4.1.2.6 PEDESTRIANS_YELOW_LED

```
#define PEDESTRIANS_YELOW_LED PB1
```

This is a pedestrians yellow led pin

Definition at line 35 of file app.h.

4.1.3 Enumeration Type Documentation

4.1.3.1 EN_TrafficStates_t

```
enum EN_TrafficStates_t
```

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

Enumerator

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

Definition at line 42 of file app.h.

4.1.4 Function Documentation

4.1.4.1 app_start()

```
void app_start (
            void  )
```

app_start this function used to initialize the MCU for the application

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

Parameters

`in`	*None*	this function doesn't take any input parameters.
`out`	*None*	this function doesn't take any output parameters.

Return values

None	This function doesn't return anything.

Definition at line 21 of file app.c.

4.1.4.2 carsGreenState()

```
void carsGreenState (
             void  )
```

carsGreenState

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

Definition at line 41 of file app.c.

4.1.4.3 carsRedState()

```
void carsRedState (
             void  )
```

carsRedState

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

Parameters

`in`	*None*	this function doesn't take any input parameters.
`out`	*None*	this function doesn't take any output parameters.

Return values

None	This function doesn't return anything.

Definition at line 64 of file app.c.

4.1.4.4 carsYellowState()

```
void carsYellowState (
            void  )
```

carsYellowState

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

Parameters

in	*None*	this function doesn't take any input parameters.
out	*None*	this function doesn't take any output parameters.

Return values

None	This function doesn't return anything.

Definition at line 48 of file app.c.

4.1.4.5 pedestriansGreenState()

```
void pedestriansGreenState (
            void  )
```

pedestriansGreenState

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

Parameters

in	*None*	this function doesn't take any input parameters.
out	*None*	this function doesn't take any output parameters.

Return values

None	This function doesn't return anything.

Definition at line 148 of file app.c.

4.1.4.6 pedestriansRedState()

```
void pedestriansRedState (
            void  )
```

pedestriansRedState

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

Parameters

`in`	*None*	this function doesn't take any input parameters.
`out`	*None*	this function doesn't take any output parameters.

Return values

None	This function doesn't return anything.

Definition at line 172 of file app.c.

4.1.4.7 pedestriansYellowState()

```
void pedestriansYellowState (
             void  )
```

pedestriansYellowState

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

Parameters

`in`	*None*	this function doesn't take any input parameters.
`out`	*None*	this function doesn't take any output parameters.

Return values

None	This function doesn't return anything.

Definition at line 157 of file app.c.

4.1.4.8 trafficLightOperation()

```
void trafficLightOperation (
             void  )
```

@brief trafficLightOperation

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

Parameters

`in`	*None*	this function doesn't take any input parameters.
`out`	*None*	this function doesn't take any output parameters.

Return values

None	This function doesn't return anything.

Definition at line 71 of file app.c.

4.2 ECUAL layer

Modules

- Button driver
- LED driver

4.2.1 Detailed Description

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

4.3 Button driver

Functions

- EN_pinErro_t buttonInit (EN_pinNum_t buttonPin)

  initialize the button pin.
- EN_pinErro_t buttonRead (EN_pinNum_t buttonPin, EN_pinState_t ∗pinState)

  reads the value of the button.

4.3.1 Detailed Description

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

4.3.2 Function Documentation

4.3.2.1 buttonInit()

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

initialize the button pin.

buttonInit function:

- This function makes the button pin as Input.

Parameters

`in`	*buttonPin*	it is the pin which the button is connected to,it may be (PA0 to PD7).
`out`	*none*	no output arguments

Return values

WRONG_PIN_NUM	if the pinNum is wrong.
OK	if the pinNum is correct.

Definition at line 11 of file Button.c.

4.3.2.2 buttonRead()

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

reads the value of the button.

buttonRead function:

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

Parameters

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

Return values

WRONG_PIN_NUM	if the pinNum is wrong.
OK	if the pinNum is correct.

Definition at line 16 of file Button.c.

4.4 LED driver

Functions

- EN_pinErro_t ledInit (EN_pinNum_t ledPin) *initialize the led pin.*

- EN_pinErro_t ledOn (EN_pinNum_t ledPin)

  turn the led on.
- EN_pinErro_t ledOff (EN_pinNum_t ledPin)

  turn the led off.
- EN_pinNum_t ledToggle (EN_pinNum_t ledPin)

  toggle the led state.

4.4.1 Detailed Description

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

4.4.2 Function Documentation

4.4.2.1 ledInit()

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

initialize the led pin.

ledInit function:

- This function initialize the led pin as output.

Parameters

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

Return values

WRONG_PIN_NUM	if the pinNum is wrong.
OK	if the pinNum is correct.

Definition at line 10 of file LED.c.

4.4.2.2 ledOff()

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

turn the led off.

ledOff function:

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

Parameters

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

Return values

WRONG_PIN_NUM	if the pinNum is wrong.
OK	if the pinNum is correct.

Definition at line 20 of file LED.c.

4.4.2.3 ledOn()

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

turn the led on.

ledOn function:

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

Parameters

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

Return values

WRONG_PIN_NUM	if the pinNum is wrong.
OK	if the pinNum is correct.

Definition at line 15 of file LED.c.

4.4.2.4 ledToggle()

```
EN_pinNum_t ledToggle (
            EN_pinNum_t ledPin )
```

toggle the led state.

ledToggle function:

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

Parameters

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

Return values

WRONG_PIN_NUM	if the pinNum is wrong.
OK	if the pinNum is correct.

Definition at line 25 of file LED.c.

4.5 MCAL layer

Modules

- DIO driver
- Interrupts driver
- Timers driver

4.5.1 Detailed Description

This layer contains all the driver related to the MCU.

4.6 DIO driver

Functions

- EN_pinErro_t DIO_pinInit (EN_pinNum_t pinNum, EN_pinDirection_t pinDirection)

  Set the direction of the pin.
- EN_pinErro_t DIO_pinWrite (EN_pinNum_t pinNum, EN_pinState_t pinState)

  This function writes High or Low on the pin.
- EN_pinErro_t DIO_pinToggle (EN_pinNum_t pinNum)

  This function toggles the state of the pin.
- EN_pinErro_t DIO_pinRead (EN_pinNum_t pinNum, EN_pinState_t ∗pinState)

  This function reads the state of the pin.

4.6.1 Detailed Description

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

4.6.2 Function Documentation

4.6.2.1 DIO_pinInit()

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

Set the direction of the pin.

DIO_pinInit

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

Parameters

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

Return values

WRONG_PIN_NUM	if the pinNum is wrong.
WRONG_PIN_DIR	if the pinDirection is wrong.
OK	if the pinNum and the pinDirection are correct.

Definition at line 12 of file DIO.c.

4.6.2.2 DIO_pinRead()

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

This function reads the state of the pin.

DIO_pinRead

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

Parameters

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

Return values

WRONG_PIN_NUM	if the pinNum is wrong.
OK	if the pinNum is correct.

Definition at line 166 of file DIO.c.

4.6.2.3 DIO_pinToggle()

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

This function toggles the state of the pin.

DIO_pinToggle

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

Parameters

in	*pinNum*	it represent the pin number (PA0 to PD7).
out	*none*	no output arguments

Return values

WRONG_PIN_NUM	if the pinNum is wrong.
OK	if the pinNum is correct.

Definition at line 198 of file DIO.c.

4.6.2.4 DIO_pinWrite()

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

This function writes High or Low on the pin.

DIO_pinWrite

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

Parameters

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

Return values

WRONG_PIN_NUM	if the pinNum is wrong.
WRONG_PIN_STATE	if the pinState is wrong.
OK	if the pinNum and the pinState are correct.

Definition at line 90 of file DIO.c.

4.7 ATMEGA32 external interrupts driver

External interrupts driver.

Enumerations

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

Functions

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

External interrupts pins

- These are the pins which connected to each interrupt.
- It should be configured as Input.

- #define INT0_PIN (PD2 - PORTD_OFFSET)
- #define INT1_PIN (PD3 - PORTD_OFFSET)
- #define INT2_PIN (PB2 - PORTB_OFFSET)

INT0 sense control

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

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

- #define ISC00 0
- #define ISC01 1

INT1 sense control

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

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

- #define ISC10 2
- #define ISC11 3

INT2 sense control

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

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

- #define ISC2 6

4.7.1 Detailed Description

External interrupts driver.

4.7.2 Macro Definition Documentation

4.7.2.1 INT0_PIN

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

This Pin connected to INT0 interrupt

Definition at line 29 of file Ext interrupt.h.

4.7.2.2 INT1_PIN

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

This Pin connected to INT1 interrupt

Definition at line 30 of file Ext interrupt.h.

4.7.2.3 INT2_PIN

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

This Pin connected to INT2 interrupt

Definition at line 31 of file Ext interrupt.h.

4.7.2.4 ISC00

```
#define ISC00 0
```

Interrupt Sense Control 0 Bit 0

Definition at line 46 of file Ext interrupt.h.

4.7.2.5 ISC01

```
#define ISC01 1
```

Interrupt Sense Control 0 Bit 1

Definition at line 47 of file Ext interrupt.h.

4.7.2.6 ISC10

```
#define ISC10 2
```

Interrupt Sense Control 1 Bit 0

Definition at line 63 of file Ext interrupt.h.

4.7.2.7 ISC11

```
#define ISC11 3
```

Interrupt Sense Control 1 Bit 1

Definition at line 64 of file Ext interrupt.h.

4.7.2.8 ISC2

```
#define ISC2 6
```

Interrupt Sense Control 2 Bit 6

Definition at line 78 of file Ext interrupt.h.

4.7.3 Enumeration Type Documentation

4.7.3.1 EN_interruptError_t

```
enum EN_interruptError_t
```

External interrupt errors.

- This enum contains the values for interrupt errors.

Enumerator

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

Definition at line 117 of file Ext interrupt.h.

4.7.3.2 EN_interruptNum_t

```
enum EN_interruptNum_t
```

External interrupt number.

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

Enumerator

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

Definition at line 89 of file Ext interrupt.h.

4.7.3.3 EN_interruptSenseControl_t

```
enum EN_interruptSenseControl_t
```

External interrupt sense control.

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

  Note

- INT2 has just rising and falling edge sense control.

Enumerator

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

Definition at line 104 of file Ext interrupt.h.

4.7.4 Function Documentation

4.7.4.1 Ext_interruptInit()

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

External interrupt init.

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

Parameters

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

Return values

INT_OK	If interruptNum and interruptSenseControl are corrects.
WRONG_INT_NUM	If interruptNum is wrong.
WRONG_SENSE_CONTROL	If interruptSenseControl is wrong.

Definition at line 9 of file Ext interrupt.c.

4.8 Interrupts driver

Modules

- ATMEGA32 external interrupts driver

  External interrupts driver.
- ATMEGA32 interrupts definitions

  Interrupts request handlers.

4.8.1 Detailed Description

4.9 ATMEGA32 interrupts definitions

Interrupts request handlers.

Macros

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

    interrupt service routine Macro.

4.9.1 Detailed Description

Interrupts request handlers.

This section contains:

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

4.9.2 Macro Definition Documentation

4.9.2.1 ADC

```
#define ADC __vector_16
```

This Macro defines ADC Conversion Complete Handler

Definition at line 63 of file Interrupt.h.

4.9.2.2 ANA_COMP

```
#define ANA_COMP __vector_18
```

This Macro defines Analog Comparator Handler

Definition at line 65 of file Interrupt.h.

4.9.2.3 cli

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

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

Definition at line 46 of file Interrupt.h.

4.9.2.4 EE_RDY

```
#define EE_RDY __vector_17
```

This Macro defines EEPROM Ready Handler

Definition at line 64 of file Interrupt.h.

4.9.2.5 EXT_INT0

```
#define EXT_INT0 __vector_1
```

This Macro defines IRQ0 Handler

Definition at line 48 of file Interrupt.h.

4.9.2.6 EXT_INT1

```
#define EXT_INT1 __vector_2
```

This Macro defines IRQ1 Handler

Definition at line 49 of file Interrupt.h.

4.9.2.7 EXT_INT2

```
#define EXT_INT2 __vector_3
```

This Macro defines IRQ2 Handler

Definition at line 50 of file Interrupt.h.

4.9.2.8 ISR

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

Value:

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

interrupt service routine Macro.

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

Precondition

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

Definition at line 78 of file Interrupt.h.

4.9.2.9 sei

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

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

Definition at line 35 of file Interrupt.h.

4.9.2.10 SPI_STC

```
#define SPI_STC __vector_12
```

This Macro defines SPI Transfer Complete Handler

Definition at line 59 of file Interrupt.h.

4.9.2.11 SPM_RDY

```
#define SPM_RDY __vector_20
```

This Macro defines Store Program Memory Ready Handler

Definition at line 67 of file Interrupt.h.

4.9.2.12 TIM0_COMP

```
#define TIM0_COMP __vector_10
```

This Macro defines Timer0 Compare Handler

Definition at line 57 of file Interrupt.h.

4.9.2.13 TIM0_OVF

```
#define TIM0_OVF __vector_11
```

This Macro defines Timer0 Overflow Handler

Definition at line 58 of file Interrupt.h.

4.9.2.14 TIM1_CAPT

```
#define TIM1_CAPT __vector_6
```

This Macro defines Timer1 Capture Handler

Definition at line 53 of file Interrupt.h.

4.9.2.15 TIM1_COMPA

```
#define TIM1_COMPA __vector_7
```

This Macro defines Timer1 CompareA Handler

Definition at line 54 of file Interrupt.h.

4.9.2.16 TIM1_COMPB

```
#define TIM1_COMPB __vector_8
```

This Macro defines Timer1 CompareB Handler

Definition at line 55 of file Interrupt.h.

4.9.2.17 TIM1_OVF

```
#define TIM1_OVF __vector_9
```

This Macro defines Timer1 Overflow Handler

Definition at line 56 of file Interrupt.h.

4.9.2.18 TIM2_COMP

```
#define TIM2_COMP __vector_4
```

This Macro defines Timer2 Compare Handler

Definition at line 51 of file Interrupt.h.

4.9.2.19 TIM2_OVF

```
#define TIM2_OVF __vector_5
```

This Macro defines Timer2 Overflow Handler

Definition at line 52 of file Interrupt.h.

4.9.2.20 TWI

```
#define TWI __vector_19
```

This Macro defines Two-wire Serial Interface Handler

Definition at line 66 of file Interrupt.h.

4.9.2.21 USART_RXC

```
#define USART_RXC __vector_13
```

This Macro defines USART RX Complete Handler

Definition at line 60 of file Interrupt.h.

4.9.2.22 USART_TXC

```
#define USART_TXC __vector_15
```

This Macro defines USART TX Complete Handler

Definition at line 62 of file Interrupt.h.

4.9.2.23 USART_UDRE

```
#define USART_UDRE __vector_14
```

This Macro defines UDR Empty Handler

Definition at line 61 of file Interrupt.h.

4.10 Timers driver

Modules

- Timer0 driver

4.10.1 Detailed Description

This contains the drivers for Atmega32 Timers

4.11 Timer0 driver

Macros

- #define TIMER0_NUM_OF_TICKS 256

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

Enumerations

- enum EN_Timer0_Mode_t { NORMAL = 0 , PWM_PHASE_CORRECR =8 , CTC =64 , FAST_PWM =72 }
- enum EN_Timer0_clkSource_t {
  NO_CLOCK_SOURCE , clkI_No_DIVISON , clkI_DIVISION_BY_8 , clkI_DIVISION_BY_64 ,
  clkI_DIVISION_BY_256 , clkI_DIVISION_BY_1024 , EXTERNAL_CLOCK_FALLING_EDGE , EXTERNAL_CLOCK_RISING_E
  }
- enum En_Timer0_Error_t { TIMER0_OK , TIMER0_WRONG_MODE , TIMER0_WRONG_CLK_SOURCE ,
  TIMER0_WRONG_INT }

Functions

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

Timer/Counter0 Interrupts Enable

- These bits enable and disable the interrupts of the counter and located in TIMSK.

- enum TIMER0_interrupt_t { TIMER0_OVER_FLOW_INT , TIMER0_OUT_CMP_MATCH_INT }
- #define TOIE0 0
- #define OCIE0 1

Bit 2:0 - CS02:0: Clock Select

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

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

Clock Select Bit Description

Note

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

- #define CS00 0
- #define CS01 1
- #define CS02 2

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

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

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

Waveform Generation Mode Bit Description

- #define WGM00 3
- #define WGM01 6

Timer/Counter0 Interrupts Flags

- These bits are flags for interrupts of the Timer 0 and located in TIFR.

- #define TOV0 0
- #define OCF0 1

4.11.1 Detailed Description

4.11.2 Macro Definition Documentation

4.11.2.1 CLR_TIMER0_CLK_SRC

```
#define CLR_TIMER0_CLK_SRC 0xF8
```

Clear Timer 0 clock source

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

Definition at line 80 of file Timer_0.h.

4.11.2.2 CLR_TIMER0_MODE

```
#define CLR_TIMER0_MODE 0xB7
```

Clear Timer 0 Mode

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

Definition at line 107 of file Timer_0.h.

4.11.2.3 CS00

```
#define CS00 0
```

Definition at line 68 of file Timer_0.h.

4.11.2.4 CS01

```
#define CS01 1
```

Definition at line 69 of file Timer_0.h.

4.11.2.5 CS02

```
#define CS02 2
```

Definition at line 70 of file Timer_0.h.

4.11.2.6 OCF0

```
#define OCF0 1
```

Bit 1 - OCF0: Output Compare Flag

Definition at line 136 of file Timer_0.h.

4.11.2.7 OCIE0

```
#define OCIE0 1
```

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

Definition at line 116 of file Timer_0.h.

4.11.2.8 TIMER0_NUM_OF_TICKS

```
#define TIMER0_NUM_OF_TICKS 256
```

System clock Macro.

- This Macro is the system clock that the user used.
- It should provided in hertz like this
  ```
  #define SYSTEM_CLK 1000000UL
  ```
  .
- Default value is 1 MHz.

Number of Ticks.

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

Definition at line 46 of file Timer_0.h.

4.11.2.9 TOIE0

```
#define TOIE0 0
```

Bit 0 - TOIE0: Timer/Counter0 Overflow Interrupt Enable

Definition at line 115 of file Timer_0.h.

4.11.2.10 TOV0

```
#define TOV0 0
```

Bit 0 - TOV0: Timer/Counter0 Overflow Flag

Definition at line 135 of file Timer_0.h.

4.11.2.11 WGM00

```
#define WGM00 3
```

Definition at line 96 of file Timer_0.h.

4.11.2.12 WGM01

```
#define WGM01 6
```

Definition at line 97 of file Timer_0.h.

4.11.3 Enumeration Type Documentation

4.11.3.1 EN_Timer0_clkSource_t

```
enum EN_Timer0_clkSource_t
```

Timer 0 clock source

- This enum contains the values for Timer0 clock source that needed to be written in TCCR0 register.
- There is no need to check for each source, just (orring) | the TCCR0 register with enum value shifted left by CS00 will do the job.
- After setting the clock source the timer will start automatically.

- `TCCR0 |= clkI_No_DIVISON«CS00;  //this will make the timer clock sorce as the system clock (No prescaling)`

Enumerator

NO_CLOCK_SOURCE	No clock source (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.

4.11.3.2 En_Timer0_Error_t

```
enum En_Timer0_Error_t
```

Timer 0 errors

- This enum contains the values for Timer 0 errors.

Enumerator

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

Definition at line 204 of file Timer_0.h.

4.11.3.3 EN_Timer0_Mode_t

```
enum EN_Timer0_Mode_t
```

Timer 0 Modes

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

example

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

enum representation in TCCR0 register:

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

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

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

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

Enumerator

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

Definition at line 170 of file Timer_0.h.

4.11.3.4 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.

4.11.4 Function Documentation

4.11.4.1 Timer0_delay_ms()

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

Timer 0 delay

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

Parameters

in	*delay_ms*	Delay time in mile seconds.
out	*void*	No output arguments.

Return values

void	This function doesn't return anything.

Definition at line 127 of file Timer_0.c.

4.11.4.2 Timer0_init()

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

Timer0 init

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

Parameters

in	*Timer0_mode*	The mode for Timer 0 it can be selected from EN_Timer0_Mode_t.
in	*Timer0_clkSource*	The clock source for Timer 0 it can be selected from EN_Timer0_clkSource_t.

Return values

TIMER0_OK	If timer 0 parameters are correct
WRONG_MODE	If timer 0 mode is wrong
WRONG_CLK_SOURCE	If timer 0 clock source is wrong

Definition at line 42 of file Timer_0.c.

4.11.4.3 Timer0_interruptDiable()

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

Timer0 interrupt disable

- This function disables Timer 0 interrupt.

Parameters

in	*TIMER0_interrupt*	Timer 0 interrupt number.
out	*void*	No output arguments.

Return values

TIMER0_OK	If timer 0 parameters are correct.
TIMER0_WRONG_INT	If timer 0 interrupt number is wrong.

Definition at line 98 of file Timer_0.c.

4.11.4.4 Timer0_interruptEnable()

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

Timer0 interrupt enable

- This function enables Timer 0 interrupt.

Parameters

in	*TIMER0_interrupt*	Timer 0 interrupt number.
out	*void*	No output arguments.

Return values

TIMER0_OK	If timer 0 parameters are correct.
TIMER0_WRONG_INT	If timer 0 interrupt number is wrong.

Definition at line 112 of file Timer_0.c.

4.11.4.5 Timer0_reset()

```
void Timer0_reset (
             void  )
```

Timer0 reset

- This function resets Timer 0 without stopping it.

Parameters

in	*void*	No input arguments.
out	*void*	No output arguments.

Return values

void	This function doesn't return anything.

Definition at line 92 of file Timer_0.c.

4.11.4.6 Timer0_start()

```
void Timer0_start (
             void  )
```

Timer0 start

- This function starts Timer 0.

Parameters

`in`	*void*	No input arguments.
`out`	*void*	No output arguments.

Return values

void	This function doesn't return anything.

Definition at line 77 of file Timer_0.c.

4.11.4.7 Timer0_stop()

```
void Timer0_stop (
             void  )
```

Timer0 stop

- This function stops Timer 0.

Parameters

`in`	*void*	No input arguments.
`out`	*void*	No output arguments.

Return values

void	This function doesn't return anything.

Definition at line 85 of file Timer_0.c.

4.12 MCU ports

Macros

- #define PORTA_OFFSET 0
- #define PORTB_OFFSET 8
- #define PORTC_OFFSET 16
- #define PORTD_OFFSET 24

Enumerations

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

4.12.1 Detailed Description

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

4.12.2 Macro Definition Documentation

4.12.2.1 PORTA_OFFSET

```
#define PORTA_OFFSET 0
```

This macro defines the start of the PORTA pins

Definition at line 62 of file ATmega32Port.h.

4.12.2.2 PORTB_OFFSET

```
#define PORTB_OFFSET 8
```

This macro defines the start of the PORTB pins

Definition at line 63 of file ATmega32Port.h.

4.12.2.3 PORTC_OFFSET

```
#define PORTC_OFFSET 16
```

This macro defines the start of the PORTC pins

Definition at line 64 of file ATmega32Port.h.

4.12.2.4 PORTD_OFFSET

```
#define PORTD_OFFSET 24
```

This macro defines the start of the PORTD pins

Definition at line 65 of file ATmega32Port.h.

4.12.3 Enumeration Type Documentation

4.12.3.1 EN_pinDirection_t

```
enum EN_pinDirection_t
```

Enumerator

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

Definition at line 72 of file ATmega32Port.h.

4.12.3.2 EN_pinErro_t

```
enum EN_pinErro_t
```

Enumerator

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

Definition at line 77 of file ATmega32Port.h.

4.12.3.3 EN_pinNum_t

```
enum EN_pinNum_t
```

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

Enumerator

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

Definition at line 22 of file ATmega32Port.h.

4.12.3.4 EN_pinState_t

```
enum EN_pinState_t
```

Enumerator

Low	enum value for Low output
High	enum value for high output

Definition at line 67 of file ATmega32Port.h.

4.13 Bit math

Macros

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

    this Macro writes 1 to the bit.
- #define clrBit(reg, bitNum) reg &= (~(1<<bitNum))

    this Macro clear the bit.
- #define toggleBit(reg, bitNum) reg ^= (1<<bitNum)

    This Macro toggle the bit logic.
- #define getBit(reg, bitNum) ((reg>>bitNum) & 0x01)

    This Macro read this bit value.

4.13.1 Detailed Description

Author : Ehab Omara
Date : 8/10/2022 12:46:40 PM
File name: BitMath.h

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

4.13.2 Macro Definition Documentation

4.13.2.1 clrBit

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

this Macro clear the bit.

clrBit function

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

Parameters

in	*reg*	this is register that needed to be changed.
in	*bitNum*	this is bit number that needed to be written to 0 in the register.

Definition at line 37 of file BitMath.h.

4.13.2.2 getBit

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

This Macro read this bit value.

getBit function

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

Parameters

in	*reg*	This is register where it reads the value from it.
in	*bitNum*	This is the bit number that needed to be read.

Definition at line 62 of file BitMath.h.

4.13.2.3 setBit

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

this Macro writes 1 to the bit.

setBit function

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

Parameters

in	*reg*	this is register that needed to be changed.
in	*bitNum*	this is bit number that needed to be written to 1 in the register.

Definition at line 26 of file BitMath.h.

4.13.2.4 toggleBit

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

This Macro toggle the bit logic.

#togBit function

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

Parameters

`in`	*reg*	this is register that needed to be changed.
`in`	*bitNum*	this is bit number that needed to be changed in the register.

Definition at line 50 of file BitMath.h.

4.14 Definition of data types

Typedefs

- typedef unsigned char uint8_t
- typedef signed char sint8_t
- typedef unsigned short int uint16_t
- typedef signed short int sint16_t
- typedef unsigned long int uint32_t
- typedef signed long int sint32_t
- typedef float float32_t
- typedef double float64_t
- typedef long double float128_t

4.14.1 Detailed Description

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

4.14.2 Typedef Documentation

4.14.2.1 float128_t

```
typedef long double float128_t
```

This is define a memory size of 16 byte float

Definition at line 23 of file dataTypes.h.

4.14.2.2 float32_t

```
typedef float float32_t
```

This is define a memory size of 4 byte float

Definition at line 21 of file dataTypes.h.

4.14.2.3 float64_t

```
typedef double float64_t
```

This is define a memory size of 8 byte float

Definition at line 22 of file dataTypes.h.

4.14.2.4 sint16_t

```
typedef signed short int sint16_t
```

This is define a memory size of 2 byte signed

Definition at line 18 of file dataTypes.h.

4.14.2.5 sint32_t

```
typedef signed long int sint32_t
```

This is define a memory size of 4 byte signed

Definition at line 20 of file dataTypes.h.

4.14.2.6 sint8_t

```
typedef signed char sint8_t
```

This is define a memory size of 1 byte signed

Definition at line 16 of file dataTypes.h.

4.14.2.7 uint16_t

```
typedef unsigned short int uint16_t
```

This is define a memory size of 2 byte

Definition at line 17 of file dataTypes.h.

4.14.2.8 uint32_t

```
typedef unsigned long int uint32_t
```

This is define a memory size of 4 byte

Definition at line 19 of file dataTypes.h.

4.14.2.9 uint8_t

```
typedef unsigned char uint8_t
```

This is define a memory size of 1 byte

Definition at line 15 of file dataTypes.h.

4.15 Service layer

Modules

- MCU ports
- Bit math
- Definition of data types
- MCU Registers

4.15.1 Detailed Description

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

4.16 MCU Registers

Modules

- I/O registers
- Interrupt registers
- Timers Registers

4.16.1 Detailed Description

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

4.17 I/O registers

Modules

- Port A registers
- Port B registers
- Port C registers
- Port D registers

4.17.1 Detailed Description

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

Note

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

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

4.18 Port A registers

Macros

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

4.18.1 Detailed Description

4.18.2 Macro Definition Documentation

4.18.2.1 DDRA

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

Direction register for port A

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

Definition at line 68 of file RegisterFile.h.

4.18.2.2 PINA

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

Input register for port A

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

Definition at line 76 of file RegisterFile.h.

4.18.2.3 PORTA

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

Output register for port A

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

Definition at line 60 of file RegisterFile.h.

4.19 Port B registers

Macros

- #define PORTB (∗((volatile uint8_t∗)0x38))
- #define DDRB (∗((volatile uint8_t∗)0x37))
- #define PINB (∗((volatile uint8_t∗)0x36))

4.19.1 Detailed Description

4.19.2 Macro Definition Documentation

4.19.2.1 DDRB

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

Direction register for port B

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

Definition at line 102 of file RegisterFile.h.

4.19.2.2 PINB

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

Input register for port A

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

Definition at line 110 of file RegisterFile.h.

4.19.2.3 PORTB

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

Output register for port B

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

Definition at line 94 of file RegisterFile.h.

4.20 Port C registers

Macros

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

4.20.1 Detailed Description

4.20.2 Macro Definition Documentation

4.20.2.1 DDRC

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

Direction register for port C

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

Definition at line 133 of file RegisterFile.h.

4.20.2.2 PINC

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

Input register for port C

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

Definition at line 141 of file RegisterFile.h.

4.20.2.3 PORTC

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

Output register for port C

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

Definition at line 125 of file RegisterFile.h.

4.21 Port D registers

Macros

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

4.21.1 Detailed Description

4.21.2 Macro Definition Documentation

4.21.2.1 DDRD

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

Direction register for port D

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

Definition at line 164 of file RegisterFile.h.

4.21.2.2 PIND

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

Input register for port D

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

Definition at line 172 of file RegisterFile.h.

4.21.2.3 PORTD

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

Output register for port D

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

Definition at line 156 of file RegisterFile.h.

4.22 Interrupt registers

Macros

- #define GICR (*((volatile uint8_t*)0x5B))
- #define GIFR (*((volatile uint8_t*)0x5A))
- #define MCUCR (*((volatile uint8_t*)0x55))
- #define MCUCSR (*((volatile uint8_t*)0x54))

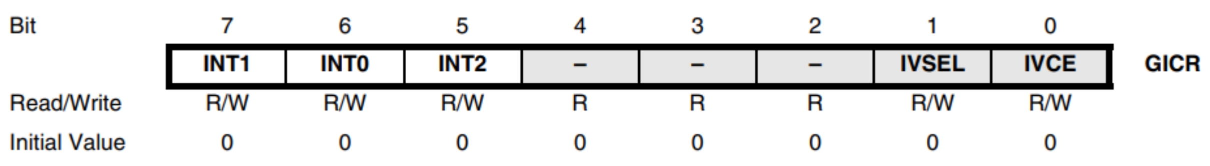
4.22.1 Detailed Description

4.22.2 Macro Definition Documentation

4.22.2.1 GICR

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

General Interrupt Control Register.



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

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

Definition at line 189 of file RegisterFile.h.

4.22.2.2 GIFR

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

General Interrupt Flag Register.

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

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

Definition at line 200 of file RegisterFile.h.

4.22.2.3 MCUCR

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

MCU Control Register.

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
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.
Interrupt 0 and interrupt 1 Sense Control		

Note

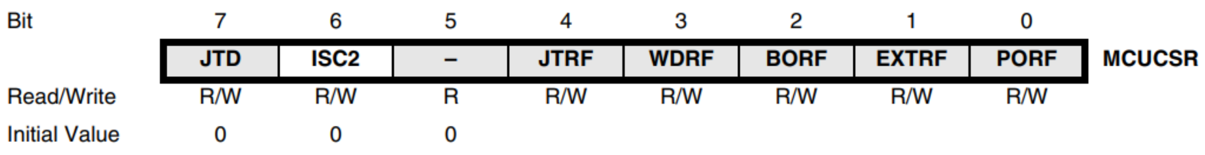
x may be 0 or 1.

Definition at line 218 of file RegisterFile.h.

4.22.2.4 MCUCSR

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

MCU Control and Status Register.



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

- Bit 6 - ISC2: Interrupt Sense Control 2

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

Definition at line 231 of file RegisterFile.h.

4.23 Timers Registers

Modules

- Timer0 Registers
- General Timers registers

4.23.1 Detailed Description

4.24 Timer0 Registers

Macros

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

4.24.1 Detailed Description

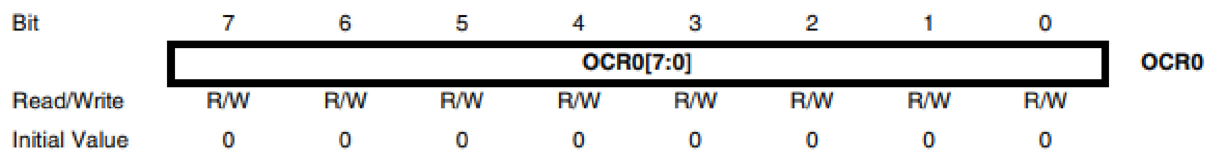
- This contains all the registers to control Timer0.

4.24.2 Macro Definition Documentation

4.24.2.1 OCR0

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

Output Compare Register



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

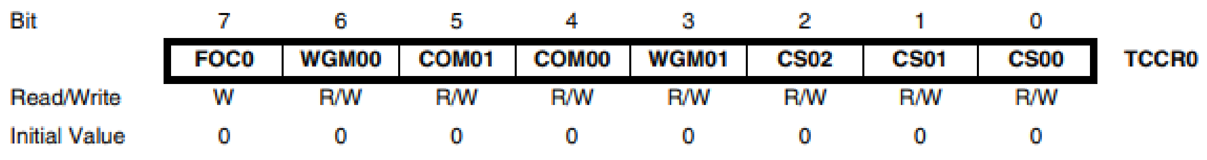
- The Output Compare Register contains an 8-bit value that is continuously compared with the counter value (TCNT0).
- A match can be used to generate an output compare interrupt, or to generate a waveform output on the OC0 pin.

Definition at line 344 of file RegisterFile.h.

4.24.2.2 TCCR0

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

Timer/Counter Control Register.



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

Bit 7 - FOC0: Force Output Compare

- When writing a logical one to the FOC0 bit, an immediate compare match is forced on the Waveform Generation unit.

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

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

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)
Compare Output Mode, Fast PWM Mode		

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.

Compare Output Mode, Fast PWM Mode

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.

Clock Select Bit Description

Note

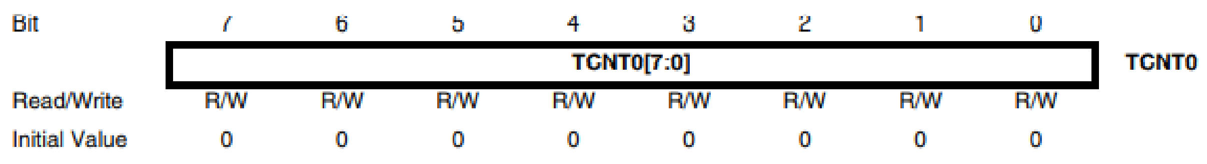
If external pin modes are used for the Timer/Counter0, transitions on the T0 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

Definition at line 321 of file RegisterFile.h.

4.24.2.3 TCNT0

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

Timer/Counter Register



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

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

Definition at line 333 of file RegisterFile.h.

4.25 General Timers registers

Macros

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

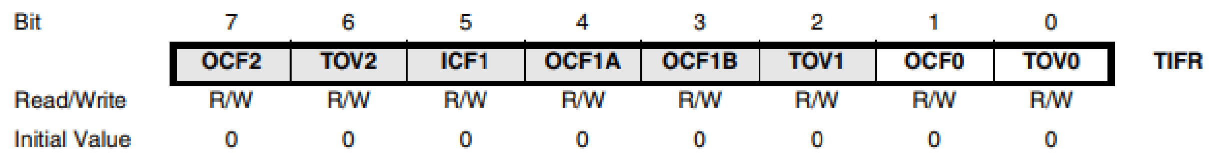
4.25.1 Detailed Description

4.25.2 Macro Definition Documentation

4.25.2.1 TIFR

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

Timer/Counter Interrupt Flag Register



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

Bit 1 - OCF0: Output Compare Flag 0

- The OCF0 bit is set (one) when a compare match occurs between the Timer/Counter0 and the data in OCR0 - Output Compare Register0.
- OCF0 is cleared by hardware when executing the corresponding interrupt handling vector.
- Alternatively, OCF0 is cleared by writing a logic one to the flag.
- When the I-bit in SREG, OCIE0 (Timer/Counter0 Com pare Match Interrupt Enable), and OCF0 are set (one), the Timer/Counter0 Compare Match Interrupt is executed.

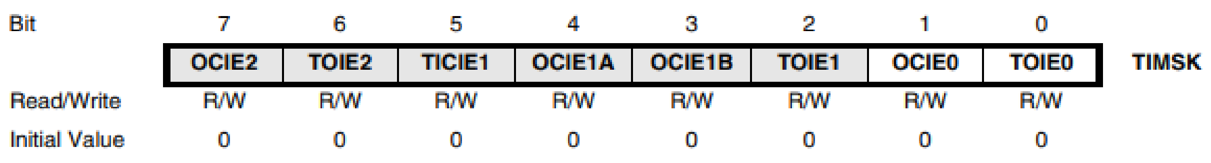
- The bit TOV0 is set (one) when an overflow occurs in Timer/Counter0.
- TOV0 is cleared by hardware when executing the corresponding interrupt handling vector.
- Alternatively, TOV0 is cleared by writing a logic one to the flag. When the SREG I-bit, TOIE0 (Timer/Counter0 Overflow Interrupt Enable),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

Definition at line 374 of file RegisterFile.h.

4.25.2.2 TIMSK

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

Timer/Counter Interrupt Mask Register



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

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

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

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

Definition at line 391 of file RegisterFile.h.

Chapter 5

File Documentation

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

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

Macros

- #define SYSTEM_CLK 1000000

Functions

- void app_start (void)

  app_start this function used to initialize the MCU for the application
- void carsGreenState (void)

  carsGreenState
- void carsYellowState (void)

  carsYellowState
- void carsRedState (void)

  carsRedState
- void trafficLightOperation (void)
- void pedestriansGreenState (void)

  pedestriansGreenState
- void pedestriansYellowState (void)

  pedestriansYellowState
- void pedestriansRedState (void)

  pedestriansRedState
- ISR (EXT_INT0)

Variables

- static EN_TrafficStates_t system_CurrentState = NORMAL_GREEN
  system current state

5.1.1 Macro Definition Documentation

5.1.1.1 SYSTEM_CLK

```
#define SYSTEM_CLK 1000000
```

Definition at line 7 of file app.c.

5.1.2 Function Documentation

5.1.2.1 ISR()

```
ISR (
            EXT_INT0   )
```

Definition at line 180 of file app.c.

5.1.3 Variable Documentation

5.1.3.1 system_CurrentState

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

system current state

- This variable store the current state of the system.

Definition at line 19 of file app.c.

5.2 app.c

Go to the documentation of this file.

```
/*****************************************************************************************************************************
/*                                                          Author    :  Ehab Omara
*/
/*                                                          Date      :  8/10/2022 12:03:35 PM
*/
/*                                                          File name:  app.c
*/

/*****************************************************************************************************************************

#define SYSTEM_CLK 1000000
#include "../ECUAL/LED driver/LED.h"
#include "../ECUAL/Button driver/Button.h"
#include "../MCAL/Timer driver/Timer_0.h"
#include "../MCAL/Ext interrupt driver/Ext interrupt.h"
#include "app.h"

EN_TrafficStates_t static system_CurrentState = NORMAL_GREEN;

void app_start(void)
{
    //initialize cars leds as output
    ledInit(CARS_GREEN_LED);
    ledInit(CARS_YELOW_LED);
    ledInit(CARS_RED_LED);

    //initialize pedestrians leds as output
    ledInit(PEDESTRIANS_GREEN_LED);
    ledInit(PEDESTRIANS_YELOW_LED);
    ledInit(PEDESTRIANS_RED_LED);

    //enable #INT0 to trigger at falling edge
    Ext_interruptInit(INT0,FALLING_EDGE);

    //configure Timer 0 at normal mode and prescaler 1024
    Timer0_init(NORMAL,clkI_DIVISION_BY_1024);

}

/*****************************************************************************************************************************
void carsGreenState(void)
{
    ledOff(CARS_RED_LED);//turn cars red led off
    ledOn(CARS_GREEN_LED);//turns card green led on
    Timer0_delay_ms(5000);//wait for 5 sec
}

void carsYellowState(void)
{
    ledOff(CARS_YELOW_LED);
    //toggle cars yellow led for 5 sec
    for(uint8_t i = 0; i < 10; i++)
    {
        ledToggle(CARS_YELOW_LED);
        Timer0_delay_ms(500);
        //check if the system state changed due to button pressed then break from the loop
        if (system_CurrentState != NORMAL_YELLOW_AFTER_GREEN && system_CurrentState !=
NORMAL_YELLOW_AFTER_RED)
        {
            break;
        }
    }
    ledOff(CARS_YELOW_LED);
}
void carsRedState(void)
{
    ledOff(CARS_GREEN_LED);//turn cars green led off
    ledOn(CARS_RED_LED);    //turns card red led on
    Timer0_delay_ms(5000); //wait for 5 sec
}

/*****************************************************************************************************************************
void trafficLightOperation(void)
{
/************************************************<Normal mode
states>*****************************************/
    ledOn(PEDESTRIANS_RED_LED);//turn pedestrians red led on
    /*check if the current state is NORMAL_GREEN*/
    if (system_CurrentState == NORMAL_GREEN)
    {
        //execute cars green state
        carsGreenState();
        //after the state finished execution
        //check if i still in the same state
        //as the state might be changed due to a button pressed
        // if i still in the same state then change the current state to the next one which is in this example NORMAL_YELLOW_AFTER_GREEN
        //if the current state changed then do nothing just go to the state which assigned by the button
        if (system_CurrentState == NORMAL_GREEN)
        {
            system_CurrentState = NORMAL_YELLOW_AFTER_GREEN;
        }
    }
    /*normal yellow after green state*/
    if (system_CurrentState == NORMAL_YELLOW_AFTER_GREEN)
    {
        carsYellowState();
        if (system_CurrentState == NORMAL_YELLOW_AFTER_GREEN)
        {
            system_CurrentState = NORMAL_RED;
        }
    }
    /*normal red state */
    if (system_CurrentState == NORMAL_RED)
    {
        carsRedState();
        if (system_CurrentState == NORMAL_RED)
        {
            system_CurrentState = NORMAL_YELLOW_AFTER_RED;
        }
    }
    /*normal yellow after red state*/
    if (system_CurrentState == NORMAL_YELLOW_AFTER_RED)
    {
        carsYellowState();
        if (system_CurrentState == NORMAL_YELLOW_AFTER_RED)
        {
            system_CurrentState = NORMAL_GREEN;
        }
    }
/***********************************************<Pedestrians mode states>*****************************************/
    //if the button pressed at green state or one of yellow state
    if(system_CurrentState == BUTTON_PRESSED_AT_GREEN || system_CurrentState == BUTTON_PRESSED_AT_YELLOW )
    {
        ledOn(PEDESTRIANS_RED_LED);

        pedestriansYellowState();

        pedestriansGreenState();

        pedestriansYellowState();

        pedestriansRedState();
        system_CurrentState = NORMAL_GREEN;

    }
    //if the button pressed at red state
    if (system_CurrentState == BUTTON_PRESSED_AT_RED)
    {
        pedestriansGreenState();

        pedestriansYellowState();

        pedestriansRedState();
        system_CurrentState = NORMAL_GREEN;
    }
}




/*********************************************************************************************************************
void pedestriansGreenState(void)
{
    ledOn(PEDESTRIANS_GREEN_LED);//turn pedestrians green led on
    ledOn(CARS_RED_LED);//turn cars red led on
    ledOff(CARS_GREEN_LED);//turn cars green led off
    ledOff(PEDESTRIANS_RED_LED);//turn pedestrians red led off
    Timer0_delay_ms(5000);//wait for 5 sec
}

void pedestriansYellowState(void)
{
    ledOff(CARS_YELOW_LED);
    ledOff(PEDESTRIANS_YELOW_LED);
    //blink pedestrians and cars yellow led for 5 sec
    for(uint8_t i = 0; i < 10; i++)
    {
        ledToggle(CARS_YELOW_LED);
        ledToggle(PEDESTRIANS_YELOW_LED);
        Timer0_delay_ms(500);
    }
    ledOff(CARS_YELOW_LED);
    ledOff(PEDESTRIANS_YELOW_LED);
}

void pedestriansRedState(void)
{
    ledOn(CARS_GREEN_LED);//turn cars green led on
    ledOn(PEDESTRIANS_RED_LED);//turn pedestrians red led on
    ledOff(PEDESTRIANS_GREEN_LED);//turn pedestrians green led off
    ledOff(CARS_RED_LED);//turn cars red led off
}

/*********************************************************************************************************
ISR(EXT_INT0)
{
    //if the button pressed at normal green state then switch to #BUTTON_PRESSED_AT_GREEN state
    if (system_CurrentState == NORMAL_GREEN)
    {
        system_CurrentState = BUTTON_PRESSED_AT_GREEN;
    }
    //if the button pressed at one of yellow states then switch to #BUTTON_PRESSED_AT_YELLOW state
    else if (system_CurrentState == NORMAL_YELLOW_AFTER_GREEN || system_CurrentState ==
NORMAL_YELLOW_AFTER_RED)
    {
        system_CurrentState = BUTTON_PRESSED_AT_YELLOW;
    }
    //if the button pressed at red state the switch to #BUTTON_PRESSED_AT_RED state
    else if (system_CurrentState == NORMAL_RED)
    {
        system_CurrentState = BUTTON_PRESSED_AT_RED;
    }
    else
    {
        //do nothing as at this state the button pressed at pedestrians mode.
        //this will neglect two press and long press.
    }
}
```

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

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

Macros

Cars LEDS Macros

These macros represent the pins which cars leds connected to.

- #define CARS_GREEN_LED PA0
- #define CARS_YELOW_LED PA1
- #define CARS_RED_LED PA2

Pedestrians LEDS Macros

These macros represent the pins which pedestrians leds connected to.

- #define PEDESTRIANS_GREEN_LED PB0
- #define PEDESTRIANS_YELOW_LED PB1
- #define PEDESTRIANS_RED_LED PB2

Enumerations

- enum EN_TrafficStates_t {
NORMAL_GREEN , NORMAL_YELLOW_AFTER_GREEN , NORMAL_RED , NORMAL_YELLOW_AFTER_RED ,
BUTTON_PRESSED_AT_GREEN , BUTTON_PRESSED_AT_RED , BUTTON_PRESSED_AT_YELLOW }

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

Functions

- void app_start (void)

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

Cars led control function

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

Parameters

`in`	None	*this function doesn't take any input parameters.*
`out`	None	*this function doesn't take any output parameters.*

Return values

None	*This function doesn't return anything.*

- void carsGreenState (void)

  carsGreenState
- void carsYellowState (void)

  carsYellowState
- void carsRedState (void)

  carsRedState

Pedestrians led control function

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

Parameters

`in`	None	*this function doesn't take any input parameters.*
`out`	None	*this function doesn't take any output parameters.*

Return values

None	*This function doesn't return anything.*

- void pedestriansGreenState (void)

  pedestriansGreenState
- void pedestriansYellowState (void)

  pedestriansYellowState

- void pedestriansRedState (void)
  pedestriansRedState

5.4 app.h

Go to the documentation of this file.

```
/*************************************************************************************************************
/*                                                          Author   :  Ehab Omara
*/
/*                                                          Date     :  8/10/2022 12:03:55 PM
*/
/*                                                          File name:  app.h
*/
/*************************************************************************************************************

#ifndef APP_H_
#define APP_H_
#include "../Service/ATmega32Port.h"
#include "../Service/dataTypes.h"
#define CARS_GREEN_LED  PA0
#define CARS_YELOW_LED  PA1
#define CARS_RED_LED    PA2

#define PEDESTRIANS_GREEN_LED   PB0
#define PEDESTRIANS_YELOW_LED   PB1
#define PEDESTRIANS_RED_LED     PB2

typedef enum
{
    NORMAL_GREEN,
    NORMAL_YELLOW_AFTER_GREEN,
    NORMAL_RED,
    NORMAL_YELLOW_AFTER_RED,
    BUTTON_PRESSED_AT_GREEN,
    BUTTON_PRESSED_AT_RED,
    BUTTON_PRESSED_AT_YELLOW,
}EN_TrafficStates_t;

void app_start(void);

void trafficLightOperation(void);



void carsGreenState(void);
void carsYellowState(void);
void carsRedState(void);


void pedestriansGreenState(void);
void pedestriansYellowState(void);
void pedestriansRedState(void);

#endif /* APP_H_ */
```

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

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

Functions

- EN_pinErro_t buttonInit (EN_pinNum_t buttonPin)
  initialize the button pin.
- EN_pinErro_t buttonRead (EN_pinNum_t buttonPin, EN_pinState_t ∗pinState)
  reads the value of the button.

5.6 Button.c

Go to the documentation of this file.

```
/*************************************************************************************************************
/*                                                            Author   :  Ehab Omara
*/
/*                                                            Date     :  8/11/2022 8:25:13 PM
*/
/*                                                            File name:  Button.c
*/
/*************************************************************************************************************


#include "Button.h"

EN_pinErro_t buttonInit(EN_pinNum_t buttonPin)
{
    return DIO_pinInit(buttonPin,Input);
}
/******************************************************************/
EN_pinErro_t buttonRead(EN_pinNum_t buttonPin,EN_pinState_t *pinState)
{
    return DIO_pinRead(buttonPin,pinState);
}
```

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

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

Functions

- EN_pinErro_t buttonInit (EN_pinNum_t buttonPin)

  initialize the button pin.
- EN_pinErro_t buttonRead (EN_pinNum_t buttonPin, EN_pinState_t ∗pinState)

  reads the value of the button.

5.8 Button.h

Go to the documentation of this file.

```
/*************************************************************************************************************
/*                                                            Author   :  Ehab Omara
*/
/*                                                            Date     :  8/11/2022 8:24:25 PM
*/
/*                                                            File name:  Button.h
*/
/*************************************************************************************************************

#ifndef BUTTON_H_
#define BUTTON_H_

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


EN_pinErro_t buttonInit(EN_pinNum_t buttonPin);

/**********************************************************************************************************/
EN_pinErro_t buttonRead(EN_pinNum_t buttonPin,EN_pinState_t *pinState);
#endif /* BUTTON_H_ */
```

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

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

Functions

- EN_pinErro_t ledInit (EN_pinNum_t ledPin)

  initialize the led pin.
- EN_pinErro_t ledOn (EN_pinNum_t ledPin)

  turn the led on.
- EN_pinErro_t ledOff (EN_pinNum_t ledPin)

  turn the led off.
- EN_pinNum_t ledToggle (EN_pinNum_t ledPin)

  toggle the led state.

5.10 LED.c

Go to the documentation of this file.

```
/*****************************************************************************************************
/*                                                         Author   :  Ehab Omara
*/
/*                                                         Date     :  8/12/2022 9:42:19 PM
*/
/*                                                         File name:  LED.c
*/
/*****************************************************************************************************
#include "LED.h"



EN_pinErro_t ledInit(EN_pinNum_t ledPin)
{
    return DIO_pinInit(ledPin,Output);
}
/*******************************************************************************/
EN_pinErro_t ledOn(EN_pinNum_t ledPin)
{
    return DIO_pinWrite(ledPin,High);
}
/*******************************************************************************/
EN_pinErro_t ledOff(EN_pinNum_t ledPin)
{
    return DIO_pinWrite(ledPin,Low);
}
/*******************************************************************************/
EN_pinNum_t ledToggle(EN_pinNum_t ledPin)
{
    return DIO_pinToggle(ledPin);
}
```

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

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

Functions

- EN_pinErro_t ledInit (EN_pinNum_t ledPin)

    initialize the led pin.
- EN_pinErro_t ledOn (EN_pinNum_t ledPin)

    turn the led on.
- EN_pinErro_t ledOff (EN_pinNum_t ledPin)

    turn the led off.
- EN_pinNum_t ledToggle (EN_pinNum_t ledPin)

    toggle the led state.

5.12 LED.h

Go to the documentation of this file.

```
/*********************************************************************************************************************
/*                                                          Author   :  Ehab Omara
*/
/*                                                          Date     :  8/12/2022 9:42:50 PM
*/
/*                                                          File name:  LED.h
*/
/*********************************************************************************************************************

#ifndef LED_H_
#define LED_H_

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

EN_pinErro_t ledInit(EN_pinNum_t ledPin);
/**********************************************************************************/
EN_pinErro_t ledOn(EN_pinNum_t ledPin);
/**********************************************************************************/
EN_pinErro_t ledOff(EN_pinNum_t ledPin);
/**********************************************************************************/
EN_pinNum_t ledToggle(EN_pinNum_t ledPin);
#endif /* LED_H_ */
```

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

5.14 IncFile1.h

Go to the documentation of this file.

```
/*********************************************************************************************************************
/*                                                          Author   :  Ehab Omara
*/
/*                                                          Date     :  8/17/2022 8:27:24 PM
*/
/*                                                          File name:  IncFile1.h
*/
/*********************************************************************************************************************

#ifndef INCFILE1_H_
#define INCFILE1_H_
#endif /* INCFILE1_H_ */
```

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

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

Functions

- int main (void)

5.15.1 Function Documentation

5.15.1.1 main()

```
int main (
            void  )
```

Definition at line 9 of file main.c.

5.16 main.c

Go to the documentation of this file.

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

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

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

Functions

- EN_pinErro_t DIO_pinInit (EN_pinNum_t pinNum, EN_pinDirection_t pinDirection)

  Set the direction of the pin.
- EN_pinErro_t DIO_pinWrite (EN_pinNum_t pinNum, EN_pinState_t pinState)

  This function writes High or Low on the pin.
- EN_pinErro_t DIO_pinRead (EN_pinNum_t pinNum, EN_pinState_t ∗pinState)

  This function reads the state of the pin.
- EN_pinErro_t DIO_pinToggle (EN_pinNum_t pinNum)

  This function toggles the state of the pin.

5.18 DIO.c

Go to the documentation of this file.

```
/*****************************************************************************************************************************
/*                                                        Author    :  Ehab Omara
*/
/*                                                        Date      :  8/10/2022 3:39:46 PM
*/
/*                                                        File name:  DIO.c
*/
/*****************************************************************************************************************************

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



EN_pinErro_t DIO_pinInit(EN_pinNum_t pinNum,EN_pinDirection_t pinDirection)
{
    EN_pinErro_t error = OK;
    //check if the pin is located in port A
    if (pinNum <= PA7)
    {
        if (pinDirection == Output)
        {
            setBit(DDRA,pinNum);
        }
        else if (pinDirection == Input)
        {
            clrBit(DDRA,pinNum);
        }
        else
        {
            error = WRONG_PIN_DIR;
        }
    }
    //check if the pin is located in port B
    else if (pinNum <= PB7)
    {
        pinNum-=PORTB_OFFSET;
        if (pinDirection == Output)
        {
            setBit(DDRB,pinNum);
        }
        else if (pinDirection == Input)
        {
            clrBit(DDRB,pinNum);
        }
        else
        {
            error = WRONG_PIN_DIR;
        }
    }
    //check if the pin is located in port C
    else if (pinNum <= PC7)
    {
        pinNum-=PORTC_OFFSET;
        if (pinDirection == Output)
        {
            setBit(DDRC,pinNum);
        }
        else if (pinDirection == Input)
        {
            clrBit(DDRC,pinNum);
        }
        else
        {
            error = WRONG_PIN_DIR;
        }
    }
    //check if the pin is located in port D
    else if (pinNum <= PD7)
    {
        pinNum-=PORTD_OFFSET;
        if (pinDirection == Output)
        {
            setBit(DDRD,pinNum);
        }
        else if (pinDirection == Input)
        {
            clrBit(DDRD,pinNum);
        }
        else
        {
            error = WRONG_PIN_DIR;
        }
    }
    //if the pinNum is wrong
    else
    {
        error = WRONG_PIN_NUM;
    }
    return error;
}
/*************************************************************************************************************
EN_pinErro_t DIO_pinWrite(EN_pinNum_t pinNum,EN_pinState_t pinState)
{
    EN_pinErro_t error = OK;
    //check if the pin is located in port A
    if (pinNum <= PA7)
    {
        if (pinState == High)
        {
            setBit(PORTA,pinNum);
        }
        else if (pinState == Low)
        {
            clrBit(PORTA,pinNum);
        }
        else
        {
            error = WRONG_PIN_STATE;
        }
    }
    //check if the pin is located in port B
    else if (pinNum <= PB7)
    {
        pinNum-=PORTB_OFFSET;
        if (pinState == High)
        {
            setBit(PORTB,pinNum);
        }
        else if (pinState == Low)
        {
            clrBit(PORTB,pinNum);
        }
        else
        {
            error = WRONG_PIN_STATE;
        }
    }
    //check if the pin is located in port C
    else if (pinNum <= PC7)
    {
        if (pinState == High)
        {
            setBit(PORTC,pinNum);
        }
        else if (pinState == Low)
        {
            clrBit(PORTC,pinNum);
        }
        else
        {
            error = WRONG_PIN_STATE;
        }
    }
    //check if the pin is located in port D
    else if (pinNum <= PD7)
    {
        if (pinState == High)
        {
            setBit(PORTD,pinNum);
        }
        else if (pinState == Low)
        {
            clrBit(PORTD,pinNum);
        }
        else
        {
            error = WRONG_PIN_STATE;
        }
    }
    //if the pinNum is wrong
    else
    {
        error = WRONG_PIN_NUM;
    }
    return error;
}

/*******************************************************************************************************************
EN_pinErro_t DIO_pinRead(EN_pinNum_t pinNum,EN_pinState_t *pinState)
{
    EN_pinErro_t error = OK;
    //check if the pin is located in port A
    if (pinNum <= PA7)
    {
        *pinState = getBit(PINA,pinNum);
    }
    //check if the pin is located in port B
    else if (pinNum <= PB7)
    {
        pinNum-=PORTB_OFFSET;
        *pinState = getBit(PINB,pinNum);
    }
    //check if the pin is located in port C
    else if (pinNum <= PC7)
    {
        *pinState = getBit(PINC,pinNum);
    }
    //check if the pin is located in port D
    else if (pinNum <= PD7)
    {
        *pinState = getBit(PIND,pinNum);
    }
    //if the pinNum is wrong
    else
    {
        error = WRONG_PIN_NUM;
    }
    return error;
}

/*******************************************************************************************************************
EN_pinErro_t DIO_pinToggle(EN_pinNum_t pinNum)
{
    EN_pinErro_t error = OK;
    //check if the pin is located in port A
    if (pinNum <= PA7)
    {
        toggleBit(PORTA,pinNum);
    }
    //check if the pin is located in port B
    else if (pinNum <= PB7)
    {
        pinNum-=PORTB_OFFSET;
        toggleBit(PORTB,pinNum);
    }
    //check if the pin is located in port C
    else if (pinNum <= PC7)
    {
        toggleBit(PORTC,pinNum);
    }
    //check if the pin is located in port D
    else if (pinNum <= PD7)
    {
        toggleBit(PORTD,pinNum);
    }
    //if the pinNum is wrong
    else
    {
        error = WRONG_PIN_NUM;
    }
    return error;
```

```
}
/*****************************************************************************************************************************
```

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

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

Functions

- EN_pinErro_t DIO_pinInit (EN_pinNum_t pinNum, EN_pinDirection_t pinDirection)

  Set the direction of the pin.
- EN_pinErro_t DIO_pinWrite (EN_pinNum_t pinNum, EN_pinState_t pinState)

  This function writes High or Low on the pin.
- EN_pinErro_t DIO_pinToggle (EN_pinNum_t pinNum)

  This function toggles the state of the pin.
- EN_pinErro_t DIO_pinRead (EN_pinNum_t pinNum, EN_pinState_t ∗pinState)

  This function reads the state of the pin.

5.19.1 Detailed Description

Author

: Ehab Omara

Date

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

Definition in file DIO.h.

5.20 DIO.h

Go to the documentation of this file.

```
/*****************************************************************************************************************************
#ifndef DIO_H_
#define DIO_H_

#include "../../Service/ATmega32Port.h"
#include "../../Service/BitMath.h"
#include "../../Service/dataTypes.h"
#include "../../Service/RegisterFile.h"
EN_pinErro_t DIO_pinInit(EN_pinNum_t pinNum,EN_pinDirection_t pinDirection);
EN_pinErro_t DIO_pinWrite(EN_pinNum_t pinNum,EN_pinState_t pinState);
EN_pinErro_t DIO_pinToggle(EN_pinNum_t pinNum);
EN_pinErro_t DIO_pinRead(EN_pinNum_t pinNum,EN_pinState_t *pinState);
#endif /* DIO_H_ */
```

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

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

Functions

- EN_interruptError_t Ext_interruptInit (EN_interruptNum_t interruptNum, EN_interruptSenseControl_t interruptSenseControl)

  External interrupt init.

5.22 Ext interrupt.c

Go to the documentation of this file.

```
/*****************************************************************************************************************************
/*                                                          Author   :  Ehab Omara                                          */
/*                                                          Date     :  8/13/2022 4:40:08 AM                                */
/*                                                          File name:  Ext interrupt.c                                     */
/*****************************************************************************************************************************


#include "Ext interrupt.h"
EN_interruptError_t Ext_interruptInit(EN_interruptNum_t interruptNum,EN_interruptSenseControl_t interruptSenseControl)
{
    EN_interruptError_t interruptError = INT_OK;
    if (interruptNum == INT0)
    {
        //check if the value of the interruptSenseControl is correct
        if (interruptSenseControl >=  LOW_LEVEL &&interruptSenseControl <= RISING_EDGE)
        {
            //enable INT0
            setBit(GICR,INT0);
            //clearing interruptSenseControl old value
            MCUCR&=(~(ISC00«0x03));
            //setting interruptSenseControl new value
            MCUCR|=interruptSenseControl«ISC00;
            //set INT0 pin as input
            clrBit(DDRD,INT0_PIN);
        }
        else
        {
            interruptError = WRONG_SENSE_CONTROL;
        }
    }
    else if (interruptNum == INT1)
    {

        //check if the value of the interruptSenseControl is correct
        if (interruptSenseControl >=  LOW_LEVEL &&interruptSenseControl <= RISING_EDGE)
        {
            //enable INT1
            setBit(GICR,INT1);
            //clearing interruptSenseControl old value
            MCUCR&=(~(0x03«ISC10));
            //setting interruptSenseControl new value
            MCUCR|=interruptSenseControl«ISC10;
            //set INT1 pin as input
            clrBit(DDRD,INT1_PIN);
        }
        else
        {
            interruptError = WRONG_SENSE_CONTROL;
        }
    }
    else if (interruptNum == INT2)
    {

        //check if the value of the interruptSenseControl is correct
        if (interruptSenseControl ==  FALLING_EDGE )
        {
            //enable INT1
            setBit(GICR,INT2);
            clrBit(MCUCSR,ISC2);
            //set INT2 pin as input
            clrBit(DDRB,INT2_PIN);
        }
        else if(interruptSenseControl == RISING_EDGE)
        {
            //enable INT1
            setBit(GICR,INT2);
            setBit(MCUCSR,ISC2);
            //set INT2 pin as input
            clrBit(DDRB,INT2_PIN);
        }
        else
        {
            interruptError = WRONG_SENSE_CONTROL;
        }
    }
    else
    {
        interruptError = WRONG_INT_NUM;
    }
    if (interruptError == INT_OK)
    {
        //enable global interrupt
        sei();
    }
    return interruptError;
}


```

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

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

Macros

External interrupts pins

- *These are the pins which connected to each interrupt.*
- *It should be configured as Input.*

- #define INT0_PIN (PD2 - PORTD_OFFSET)
- #define INT1_PIN (PD3 - PORTD_OFFSET)
- #define INT2_PIN (PB2 - PORTB_OFFSET)

INT0 sense control

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

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

- #define ISC00 0
- #define ISC01 1

INT1 sense control

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

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

- #define ISC10 2
- #define ISC11 3

INT2 sense control

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

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

- #define ISC2 6

Enumerations

- enum EN_interruptNum_t { INT2 = 5 , INT0 , INT1 }

  External interrupt number.
- enum EN_interruptSenseControl_t { LOW_LEVEL , ANY_LOGICAL_CHANGE , FALLING_EDGE , RISING_EDGE }

  External interrupt sense control.
- enum EN_interruptError_t { INT_OK , WRONG_INT_NUM , WRONG_SENSE_CONTROL }

  External interrupt errors.

Functions

- EN_interruptError_t Ext_interruptInit (EN_interruptNum_t interruptNum, EN_interruptSenseControl_t interruptSenseControl)

  External interrupt init.

5.24 Ext interrupt.h

Go to the documentation of this file.

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

#ifndef EXT_INTERRUPT_H_
#define EXT_INTERRUPT_H_

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

#define INT0_PIN (PD2 - PORTD_OFFSET)
#define INT1_PIN (PD3 - PORTD_OFFSET)
#define INT2_PIN (PB2 - PORTB_OFFSET)

#define ISC00 0
#define ISC01 1
/***************************************************************************************************************

#define ISC10 2
#define ISC11 3


/***************************************************************************************************************

#define ISC2 6

/***************************************************************************************************************

typedef enum
{
    INT2 = 5,
    INT0,
    INT1
}EN_interruptNum_t;

/***************************************************************************************************************
typedef enum
{
    LOW_LEVEL,
    ANY_LOGICAL_CHANGE,
    FALLING_EDGE,
    RISING_EDGE
}EN_interruptSenseControl_t;

/***************************************************************************************************************
typedef enum
{
    INT_OK,
    WRONG_INT_NUM,
    WRONG_SENSE_CONTROL
}EN_interruptError_t;

/***************************************************************************************************************
EN_interruptError_t Ext_interruptInit(EN_interruptNum_t interruptNum,EN_interruptSenseControl_t
interruptSenseControl);
#endif /* EXT_INTERRUPT_H_ */
```

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

Macros

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

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

  interrupt service routine Macro.

5.26 Interrupt.h

Go to the documentation of this file.

```
/************************************************************************************************
/*                                                        Author   :  Ehab Omara                 */
/*                                                        Date     :  8/13/2022 1:08:16 AM       */
/*                                                        File name:  Interrupt.h                */
/************************************************************************************************

#ifndef INTERRUPT_H_
#define INTERRUPT_H_
# 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)   void INT_VECT(void)__attribute__((signal,used));\
void INT_VECT(void)
#endif /* INTERRUPT_H_ */
```

5.27 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Timer driver/Timer_0.c File Reference

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

Functions

- En_Timer0_Error_t Timer0_init (EN_Timer0_Mode_t Timer0_mode, EN_Timer0_clkSource_t Timer0_clk←
  Source)
- 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.

5.27.1 Function Documentation

5.27.1.1 ISR()

```
ISR (
            TIM0_OVF   )
```

Definition at line 147 of file Timer_0.c.

5.27.2 Variable Documentation

5.27.2.1 Timer0_globalClkSource

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

Global static variable for Timer 0 clock source.

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

Definition at line 15 of file Timer_0.c.

5.27.2.2 Timer0_globalNumOfOverFlows

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

Global static variable for Timer 0 over flows number.

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

Definition at line 25 of file Timer_0.c.

5.27.2.3 Timer0_globalOverFlowTime

```
float64_t volatile Timer0_globalOverFlowTime = 0  [static]
```

Global static variable for Timer 0 over flow time.

- This variable stores the time for one over flow.

Definition at line 33 of file Timer_0.c.

5.27.2.4 Timer0_globalTickTime

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

Global static variable for Timer 0 tick time.

- This variable stores the time for one tick.

Definition at line 40 of file Timer_0.c.

5.28 Timer_0.c

Go to the documentation of this file.

```
/*******************************************************************************************************************************
/*                                                        Author    :  Ehab Omara                                             */
/*                                                        Date      :  8/14/2022 12:55:03 PM                                  */
/*                                                        File name:  Timer_0.c                                               */
/*******************************************************************************************************************************
#include "Timer_0.h"
#include <math.h>
#include "../Interrupt/Interrupt.h"
static EN_Timer0_clkSource_t Timer0_globalClkSource = clkI_No_DIVISON;
/*********************************************************************************************************************/
static uint32_t volatile Timer0_globalNumOfOverFlows = 0;
/*********************************************************************************************************************/
static float64_t volatile Timer0_globalOverFlowTime = 0;
static float64_t volatile Timer0_globalTickTime = 0;
/*********************************************************************************************************************/
En_Timer0_Error_t Timer0_init(EN_Timer0_Mode_t Timer0_mode,EN_Timer0_clkSource_t Timer0_clkSource)
{
    En_Timer0_Error_t Timer0_error = TIMER0_OK;
    //selecting Timer 0 mode
    if (Timer0_mode == NORMAL || Timer0_mode == PWM_PHASE_CORRECR || Timer0_mode == CTC || Timer0_mode == FAST_PWM)
    {

        //clear the old mode value
        TCCR0 &= CLR_TIMER0_MODE;
        //set the new mode value
        TCCR0 |= Timer0_mode;
    }
    else
    {
        Timer0_error = TIMER0_WRONG_MODE;
    }
    //selecting Timer 0 clock source
    if (Timer0_clkSource >= NO_CLOCK_SOURCE && Timer0_clkSource <= EXTERNAL_CLOCK_RISING_EDGE)
    {
        Timer0_globalClkSource = Timer0_clkSource;


        uint16_t Timer0_localClkPrescaler[] = {0,1,8,64,256,1024};
        //calculate Timer 0 tick time
        Timer0_globalTickTime = (float64_t)Timer0_localClkPrescaler[Timer0_clkSource]/SYSTEM_CLK;
        //calculate Timer 0 over flow time
        Timer0_globalOverFlowTime = Timer0_globalTickTime * TIMER0_NUM_OF_TICKS;
    }
    else
    {
        Timer0_error = TIMER0_WRONG_CLK_SOURCE;
    }
    return Timer0_error;
}
/*********************************************************************************************************************/
void Timer0_start(void)
{
    //clear the old clock source value
    TCCR0 &= CLR_TIMER0_CLK_SRC;
    //set the new clock source value
    TCCR0 |= Timer0_globalClkSource « CS00;
}
/*********************************************************************************************************************/
void Timer0_stop(void)
{
    //clear the value of Timer 0 clock source
    //this is done by clearing the three bits #CS00, #CS01 and #CS02
    TCCR0 &= CLR_TIMER0_CLK_SRC;
}
/*********************************************************************************************************************/
void Timer0_reset(void)
{
    TCNT0 = 0x00;
    Timer0_globalNumOfOverFlows = 0;
}
```

```
/*****************************************************************************************************************/
En_Timer0_Error_t Timer0_interruptDiable(TIMER0_interrupt_t Timer0_interrupt)
{
    En_Timer0_Error_t Timer0_error = TIMER0_OK;
    if (Timer0_interrupt == TIMER0_OVER_FLOW_INT || Timer0_interrupt == TIMER0_OUT_CMP_MATCH_INT)
    {
        clrBit(TIMSK,Timer0_interrupt);
    }
    else
    {
        Timer0_error = TIMER0_WRONG_INT;
    }
    return Timer0_error;
}

/*****************************************************************************************************************/
En_Timer0_Error_t Timer0_interruptEnable(TIMER0_interrupt_t Timer0_interrupt)
{
    En_Timer0_Error_t Timer0_error = TIMER0_OK;
    if (Timer0_interrupt == TIMER0_OVER_FLOW_INT || Timer0_interrupt == TIMER0_OUT_CMP_MATCH_INT)
    {
        sei();
        setBit(TIMSK,Timer0_interrupt);
    }
    else
    {
        Timer0_error = TIMER0_WRONG_INT;
    }
    return Timer0_error;
}

/*****************************************************************************************************************/
void Timer0_delay_ms(uint32_t delay_ms)
{
    //reset Timer 0
    Timer0_reset();
    //convert delay time from mile seconds to seconds
    float64_t neededTimeInsecond = (float64_t)delay_ms/1000;
    //calculate number of over flows needed to reach the desired time
    uint32_t numberOfoverFlows = ceil(neededTimeInsecond/Timer0_globalOverFlowTime);
    //calculate the initial value for #TCNT0 register
    TCNT0 = TIMER0_NUM_OF_TICKS -(neededTimeInsecond/Timer0_globalTickTime)/numberOfoverFlows;
    //enable Timer 0 over flow interrupt
    Timer0_interruptEnable(TIMER0_OVER_FLOW_INT);
    //start Timer 0 to count
    Timer0_start();
    //wait until reaching needed number over flows
    while(Timer0_globalNumOfOverFlows < numberOfoverFlows);
    //stop Timer 0 after reaching the desired time.
    Timer0_stop();
}

ISR(TIM0_OVF)
{
    Timer0_globalNumOfOverFlows++;
}
```

5.29 E:/Ehab/EGFWD embedded/On-Demand Traffic Control/On-Demand Traffic Control/On-Demand Traffic Control/MCAL/Timer driver/Timer_0.h File Reference

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

Macros

- #define TIMER0_NUM_OF_TICKS 256

    System clock Macro.

- #define CLR_TIMER0_CLK_SRC 0xF8
- #define CLR_TIMER0_MODE 0xB7

Bit 2:0 - CS02:0: Clock Select

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

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

Note

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

- #define CS00 0
- #define CS01 1
- #define CS02 2

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

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

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

- #define WGM00 3
- #define WGM01 6

Timer/Counter0 Interrupts Flags

- *These bits are flags for interrupts of the Timer 0 and located in TIFR.*

- #define TOV0 0
- #define OCF0 1

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 , clkI_No_DIVISON , clkI_DIVISION_BY_8 , clkI_DIVISION_BY_64 ,
  clkI_DIVISION_BY_256 , clkI_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.

- #define TOIE0 0
- #define OCIE0 1
- enum TIMER0_interrupt_t { TIMER0_OVER_FLOW_INT , TIMER0_OUT_CMP_MATCH_INT }

5.30 Timer_0.h

Go to the documentation of this file.

```
/************************************************************************************************************************
/*                                                         Author    :  Ehab Omara                                     */
/*                                                         Date      :  8/14/2022 12:55:53 PM                          */
/*                                                         File name:  Timer_0.h                                       */
/************************************************************************************************************************

#ifndef TIMER_0_H_
#define TIMER_0_H_
#include "../../Service/BitMath.h"
#include "../../Service/ATmega32Port.h"
#include "../../Service/dataTypes.h"
#include "../../Service/RegisterFile.h"
/*****************************************************************************************************/
/*****************************************************************************************************/
#ifndef SYSTEM_CLK
/* prevent compiler error by supplying a default */
# warning "SYSTEM_CLK not defined for Timer_0.h, default value is 1MHz"
#define SYSTEM_CLK  1000000UL
#endif
/*****************************************************************************************************/
#define TIMER0_NUM_OF_TICKS 256
```

```

/******************************************************************************************************/
#define CS00    0
#define CS01    1
#define CS02    2

/******************************************************************************************************/
#define CLR_TIMER0_CLK_SRC 0xF8

/******************************************************************************************************/
#define WGM00   3
#define WGM01   6

/******************************************************************************************************/
#define CLR_TIMER0_MODE 0xB7

/******************************************************************************************************/
#define TOIE0   0
#define OCIE0   1
typedef enum
{
    TIMER0_OVER_FLOW_INT,
    TIMER0_OUT_CMP_MATCH_INT
}TIMER0_interrupt_t;

/******************************************************************************************************/
#define TOV0    0
#define OCF0    1

/******************************************************************************************************/
typedef enum
{
    NORMAL = 0,
    PWM_PHASE_CORRECR=8,
    CTC=64,
    FAST_PWM=72
}EN_Timer0_Mode_t;

/******************************************************************************************************/
typedef enum
{
    NO_CLOCK_SOURCE,
    clkI_No_DIVISON,
    clkI_DIVISION_BY_8,
    clkI_DIVISION_BY_64,
    clkI_DIVISION_BY_256,
    clkI_DIVISION_BY_1024,
    EXTERNAL_CLOCK_FALLING_EDGE,
    EXTERNAL_CLOCK_RISING_EDGE
}EN_Timer0_clkSource_t;

/******************************************************************************************************/
typedef enum
{
    TIMER0_OK,
    TIMER0_WRONG_MODE,
    TIMER0_WRONG_CLK_SOURCE,
    TIMER0_WRONG_INT
}En_Timer0_Error_t;

/******************************************************************************************************/
En_Timer0_Error_t Timer0_interruptEnable(TIMER0_interrupt_t Timer0_interrupt);

/******************************************************************************************************/
En_Timer0_Error_t Timer0_interruptDiable(TIMER0_interrupt_t Timer0_interrupt);

/******************************************************************************************************/
En_Timer0_Error_t Timer0_init(EN_Timer0_Mode_t Timer0_mode,EN_Timer0_clkSource_t Timer0_clkSource);

/******************************************************************************************************/
void Timer0_start(void);

/******************************************************************************************************/
void Timer0_stop(void);

/******************************************************************************************************/
void Timer0_reset(void);

/******************************************************************************************************/
void Timer0_delay_ms(uint32_t delay_ms);
#endif /* TIMER_0_H_ */
```

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

Macros

- #define PORTA_OFFSET 0
- #define PORTB_OFFSET 8
- #define PORTC_OFFSET 16
- #define PORTD_OFFSET 24

Enumerations

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

5.32 ATmega32Port.h

Go to the documentation of this file.

```
/**************************************************************************************************************
/*                                                           Author   :  Ehab Omara
*/
/*                                                           Date     :  8/10/2022 3:49:55 PM
*/
/*                                                           File name:  ATmega32Port.h
*/
/**************************************************************************************************************

#ifndef ATMEGA32PORT_H_
#define ATMEGA32PORT_H_


typedef enum
{
    /*PORTA pins*/
    PA0,
    PA1,
    PA2,
    PA3,
    PA4,
    PA5,
    PA6,
    PA7,
    /*PORTB pins*/
    PB0,
    PB1,
    PB2,
    PB3,
    PB4,
    PB5,
    PB6,
    PB7,
    /*PORTC pins*/
    PC0,
    PC1,
    PC2,
    PC3,
    PC4,
    PC5,
    PC6,
    PC7,
    /*PORTD pins*/
    PD0,
    PD1,
    PD2,
    PD3,
    PD4,
    PD5,
    PD6,
    PD7
}EN_pinNum_t;

#define PORTA_OFFSET    0
#define PORTB_OFFSET    8
#define PORTC_OFFSET    16
#define PORTD_OFFSET    24
typedef enum
{
    Low,
    High
}EN_pinState_t;
typedef enum
{
    Input,
    Output
}EN_pinDirection_t;
typedef enum
{
    OK,
    WRONG_PIN_NUM,
    WRONG_PIN_DIR,
    WRONG_PIN_STATE
}EN_pinErro_t;
#endif /* ATMEGA32PORT_H_ */
```

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

Macros

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

  this Macro writes 1 to the bit.
- #define clrBit(reg, bitNum) reg &= (∼(1<<bitNum))

  this Macro clear the bit.
- #define toggleBit(reg, bitNum) reg ^= (1<<bitNum)

  This Macro toggle the bit logic.
- #define getBit(reg, bitNum) ((reg>>bitNum) & 0x01)

  This Macro read this bit value.

5.34 BitMath.h

Go to the documentation of this file.

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

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

Typedefs

- typedef unsigned char uint8_t
- typedef signed char sint8_t
- typedef unsigned short int uint16_t
- typedef signed short int sint16_t
- typedef unsigned long int uint32_t
- typedef signed long int sint32_t
- typedef float float32_t
- typedef double float64_t
- typedef long double float128_t

5.36 dataTypes.h

Go to the documentation of this file.

```
/**********************************************************************************************************************
/*                                                           Author   :  Ehab Omara
*/
/*                                                           Date     :  8/10/2022 12:06:28 PM
*/
/*                                                           File name:  dataTypes.h
*/
/**********************************************************************************************************************

#ifndef DATATYPES_H_
#define DATATYPES_H_
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;
#endif /* DATATYPES_H_ */
```

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

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

Macros

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

5.38 RegisterFile.h

Go to the documentation of this file.

```
/***************************************************************************************************************************
/*                                                          Author    :  Ehab Omara
*/
/*                                                          Date      :  8/10/2022 12:06:56 PM
*/
/*                                                          File name:  RegisterFile.h
*/
/***************************************************************************************************************************

#ifndef REGISTERFILE_H_
#define REGISTERFILE_H_

#include "dataTypes.h"
/*
* if the DDRx is set to be output and we write High to the PORTx
* this will activate the internal Pull up resistor.
*/

/************************************************************ Port A registers
**********************************************************/
#define PORTA    (*((volatile uint8_t*)0x3B))  //1->high output              0->low output
#define DDRA     (*((volatile uint8_t*)0x3A))  //1->to make it output        0->to make it input
#define PINA     (*((volatile uint8_t*)0x39))  //this register to read a value from a pin
/************************************************************ Port B registers
**********************************************************/
#define PORTB    (*((volatile uint8_t*)0x38))
#define DDRB     (*((volatile uint8_t*)0x37))
#define PINB     (*((volatile uint8_t*)0x36))
/************************************************************ Port C registers
**********************************************************/
#define PORTC    (*((volatile uint8_t*)0x35))
#define DDRC     (*((volatile uint8_t*)0x34))
#define PINC     (*((volatile uint8_t*)0x33))
/************************************************************ Port D registers
**********************************************************/
#define PORTD    (*((volatile uint8_t*)0x32))
#define DDRD     (*((volatile uint8_t*)0x31))
#define PIND     (*((volatile uint8_t*)0x30))
```

```
/************************************************************ Interrupts registers
************************************************************/
#define GICR    (*((volatile uint8_t*)0x5B))

#define GIFR    (*((volatile uint8_t*)0x5A))
#define MCUCR   (*((volatile uint8_t*)0x55))

#define MCUCSR  (*((volatile uint8_t*)0x54))
/************************************************************ Timers registers
************************************************************/
#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))
#endif /* REGISTERFILE_H_ */
```

Index