

Cute OS

1.0.0

Generated by Doxygen 1.9.1

<b>1 Data Structure Index</b>	<b>1</b>
1.1 Data Structures	1
<b>2 File Index</b>	<b>1</b>
2.1 File List	1
<b>3 Data Structure Documentation</b>	<b>2</b>
3.1 cuteOS_TASK_t Struct Reference	2
3.1.1 Detailed Description	2
3.1.2 Field Documentation	2
3.2 TRAFFIC_CONFIGS_t Struct Reference	3
3.2.1 Detailed Description	3
3.2.2 Field Documentation	3
<b>4 File Documentation</b>	<b>4</b>
4.1 code/include/BIT_MATH.h File Reference	4
4.1.1 Detailed Description	5
4.1.2 Macro Definition Documentation	5
4.2 BIT_MATH.h	9
4.3 code/include/cuteOS.h File Reference	9
4.3.1 Detailed Description	10
4.3.2 Function Documentation	10
4.4 cuteOS.h	16
4.5 code/include/main.h File Reference	16
4.5.1 Detailed Description	17
4.5.2 Macro Definition Documentation	17
4.6 main.h	18
4.7 code/include/port.h File Reference	19
4.7.1 Detailed Description	19
4.7.2 Variable Documentation	20
4.8 port.h	21
4.9 code/include/STD_TYPES.h File Reference	21
4.9.1 Detailed Description	22
4.9.2 Macro Definition Documentation	22
4.9.3 Typedef Documentation	22
4.9.4 Enumeration Type Documentation	23
4.10 STD_TYPES.h	25
4.11 code/include/traffic.h File Reference	25
4.11.1 Detailed Description	26
4.11.2 Enumeration Type Documentation	26
4.11.3 Function Documentation	27
4.12 traffic.h	30
4.13 code/include/traffic_cfg.h File Reference	30

4.13.1 Detailed Description . . . . .	31
4.13.2 Enumeration Type Documentation . . . . .	31
4.13.3 Variable Documentation . . . . .	32
4.14 traffic_cfg.h . . . . .	32
4.15 code/src/cuteOS.c File Reference . . . . .	32
4.15.1 Detailed Description . . . . .	33
4.15.2 Macro Definition Documentation . . . . .	34
4.15.3 Function Documentation . . . . .	35
4.15.4 Variable Documentation . . . . .	38
4.16 cuteOS.c . . . . .	39
4.17 code/src/main.c File Reference . . . . .	41
4.17.1 Detailed Description . . . . .	41
4.17.2 Function Documentation . . . . .	42
4.18 main.c . . . . .	46
4.19 code/src/traffic.c File Reference . . . . .	47
4.19.1 Detailed Description . . . . .	47
4.19.2 Function Documentation . . . . .	48
4.20 traffic.c . . . . .	50
4.21 code/src/traffic_cfg.c File Reference . . . . .	52
4.21.1 Detailed Description . . . . .	52
4.21.2 Variable Documentation . . . . .	53
4.22 traffic_cfg.c . . . . .	53
<b>Index</b>	<b>55</b>

## 1 Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<code>cuteOS_TASK_t</code>	2
<code>TRAFFIC_CONFIGS_t</code>	3

## 2 File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<code>code/include/BIT_MATH.h</code> Common bit manipulation operations	4
--	---

<code>code/include/cuteOS.h</code>	
Simple EOS interfaces header file. See <a href="#">cuteOS.c</a> for more details	9
<code>code/include/main.h</code>	
Project Header for <a href="#">main.c</a>	16
<code>code/include/port.h</code>	
Port Header file for the milk pasteurization example	19
<code>code/include/STD_TYPES.h</code>	
Standard data types For AVR Microcontrollers	21
<code>code/include/traffic.h</code>	
Traffic Light System interfaces header file. See <a href="#">traffic.c</a> for more details	25
<code>code/include/traffic_cfg.h</code>	
Traffic Light System interfaces header file. See <a href="#">traffic.c</a> for more details	30
<code>code/src/cuteOS.c</code>	
Main file for Cute Embedded Operating System (cuteOS) for 8051	32
<code>code/src/main.c</code>	
Testing cute OS	41
<code>code/src/traffic.c</code>	
This is a traffic Light project (Chapter 8 - Embedded C by Professor j. Pont)	47
<code>code/src/traffic_cfg.c</code>	
Configurations of Traffic Light System	52

## 3 Data Structure Documentation

### 3.1 cuteOS\_TASK\_t Struct Reference

#### Data Fields

- [ERROR\\_t](#)(\* [callback](#))(void)
- [u16](#) ticks
- [u8](#) id

#### 3.1.1 Detailed Description

Definition at line 59 of file [cuteOS.c](#).

#### 3.1.2 Field Documentation

##### 3.1.2.1 [callback](#) [ERROR\\_t](#)(\* [callback](#)) (void)

Pointer to the task function

Definition at line 60 of file [cuteOS.c](#).

### 3.1.2.2 id `u8 id`

Task ID

Definition at line 62 of file [cuteOS.c](#).

### 3.1.2.3 ticks `u16 ticks`

Number of ticks after which the task will run

Definition at line 61 of file [cuteOS.c](#).

The documentation for this struct was generated from the following file:

- [code/src/cuteOS.c](#)

## 3.2 TRAFFIC\_CONFIGS\_t Struct Reference

```
#include <traffic_cfg.h>
```

### Data Fields

- [TRAFFIC\\_SEQUENCE\\_DURATION\\_t red\\_duration](#)
- [TRAFFIC\\_SEQUENCE\\_DURATION\\_t red\\_amber\\_duration](#)
- [TRAFFIC\\_SEQUENCE\\_DURATION\\_t green\\_duration](#)
- [TRAFFIC\\_SEQUENCE\\_DURATION\\_t amber\\_duration](#)

### 3.2.1 Detailed Description

Definition at line 29 of file [traffic\\_cfg.h](#).

### 3.2.2 Field Documentation

#### 3.2.2.1 amber\_duration `TRAFFIC_SEQUENCE_DURATION_t amber_duration`

Definition at line 33 of file [traffic\\_cfg.h](#).

#### 3.2.2.2 green\_duration `TRAFFIC_SEQUENCE_DURATION_t green_duration`

Definition at line 32 of file [traffic\\_cfg.h](#).

### 3.2.2.3 red\_amber\_duration [TRAFFIC\\_SEQUENCE\\_DURATION\\_t](#) red\_amber\_duration

Definition at line 31 of file [traffic\\_cfg.h](#).

### 3.2.2.4 red\_duration [TRAFFIC\\_SEQUENCE\\_DURATION\\_t](#) red\_duration

Definition at line 30 of file [traffic\\_cfg.h](#).

The documentation for this struct was generated from the following file:

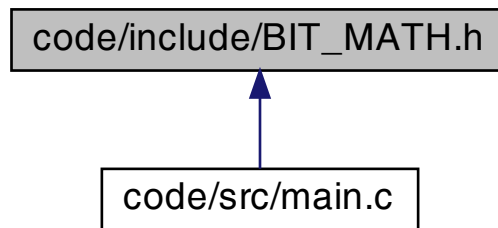
- [code/include/traffic\\_cfg.h](#)

## 4 File Documentation

### 4.1 [code/include/BIT\\_MATH.h](#) File Reference

Common bit manipulation operations.

This graph shows which files directly or indirectly include this file:



### Macros

- `#define GET_BIT(REGISTER, BIT) ( 1 & ( (REGISTER) >> (BIT) ) )`  
*Read state of a specific bit.*
- `#define SET_BIT(REGISTER, BIT) ( (REGISTER) |= (1 << (BIT)) )`  
*Set state of a specific bit (set to 1)*
- `#define CLR_BIT(REGISTER, BIT) ( (REGISTER) &= ~(1 << (BIT)) )`  
*Clear state of a specific bit (set to 0)*
- `#define TOG_BIT(REGISTER, BIT) ( (REGISTER) ^= (1 << (BIT)) )`  
*Toggle state of a specific bit (set to 0)*
- `#define BIT_IS_SET(REGISTER, Bit) ( (REGISTER) & (1 << (Bit)) )`  
*Check if state of a specific bit is set (state = 1)*
- `#define BIT_IS_CLEAR(REGISTER, Bit) ( !( (REGISTER) & (1 << (Bit)) ) )`  
*Check if state of a specific bit is Cleared (state = 0)*
- `#define CONCAT_8BITS(b7, b6, b5, b4, b3, b2, b1, b0) (0b##b7##b6##b5##b4##b3##b2##b1##b0)`
- `#define CONCAT_7BITS(b6, b5, b4, b3, b2, b1, b0) (0b##b6##b5##b4##b3##b2##b1##b0)`
- `#define CONCAT_6BITS(b5, b4, b3, b2, b1, b0) (0b##b5##b4##b3##b2##b1##b0)`
- `#define CONCAT_5BITS(b4, b3, b2, b1, b0) (0b##b4##b3##b2##b1##b0)`
- `#define CONCAT_4BITS(b3, b2, b1, b0) (0b##b3##b2##b1##b0)`
- `#define CONCAT_3BITS(b2, b1, b0) (0b##b2##b1##b0)`
- `#define CONCAT_2BITS(b1, b0) (0b##b1##b0)`

### 4.1.1 Detailed Description

Common bit manipulation operations.

#### Author

Mahmoud Karam ( [ma.karam272@gmail.com](mailto:ma.karam272@gmail.com) )

#### Version

1.0.0

#### Date

2021-07-31

Definition in file [BIT\\_MATH.h](#).

### 4.1.2 Macro Definition Documentation

**4.1.2.1 BIT\_IS\_CLEAR** `#define BIT_IS_CLEAR(  
REGISTER,  
Bit ) ( !( (REGISTER) & (1 << (Bit)) ) )`

Check if state of a specific bit is Cleared (state = 0)

#### Parameters

in	<i>REGISTER</i>	the register includes the bit
in	<i>BIT</i>	the required bit number to be set

#### Returns

1 or 0: 1 if the bit is cleared, 0 if the bit is set

For example:

[BIT\\_IS\\_CLEAR\(PORT\\_A, PIN0\)](#) will return 1 if bit 0 of PORT\_A is LOW or 0 if it is HIGH

Definition at line 67 of file [BIT\\_MATH.h](#).

**4.1.2.2 BIT\_IS\_SET** `#define BIT_IS_SET(  
REGISTER,  
Bit ) ( (REGISTER) & (1 << (Bit)) )`

Check if state of a specific bit is set (state = 1)

**Parameters**

in	<i>REGISTER</i>	the register includes the bit
in	<i>BIT</i>	the required bit number to be set

**Returns**

1 or 0: 1 if the bit is set, 0 if the bit is cleared

**For example:**

`BIT_IS_SET(PORT_A, PIN0)` will return 1 if bit 0 of PORT\_A is HIGH or 0 if it is LOW

Definition at line 56 of file [BIT\\_MATH.h](#).

**4.1.2.3 CLR\_BIT** `#define CLR_BIT(  
 REGISTER,  
 BIT ) ( (REGISTER) &= ~(1 << (BIT)) )`

Clear state of a specific bit (set to 0)

**Parameters**

in	<i>REGISTER</i>	the register includes the bit
in	<i>BIT</i>	the required bit number to be cleared

**For example:**

`CLEAR_BIT(PORT_A, PIN0)` will set bit 0 of PORT\_A to LOW (0)

Definition at line 37 of file [BIT\\_MATH.h](#).

**4.1.2.4 CONCAT\_2BITS** `#define CONCAT_2BITS(  
 b1,  
 b0 ) (0b##b1##b0)`

Definition at line 75 of file [BIT\\_MATH.h](#).

**4.1.2.5 CONCAT\_3BITS** `#define CONCAT_3BITS(  
 b2,  
 b1,  
 b0 ) (0b##b2##b1##b0)`

Definition at line 74 of file [BIT\\_MATH.h](#).



**4.1.2.6 CONCAT\_4BITS** `#define CONCAT_4BITS(  
 b3,  
 b2,  
 b1,  
 b0 ) (0b##b3##b2##b1##b0)`

Definition at line 73 of file [BIT\\_MATH.h](#).

**4.1.2.7 CONCAT\_5BITS** `#define CONCAT_5BITS(  
 b4,  
 b3,  
 b2,  
 b1,  
 b0 ) (0b##b4##b3##b2##b1##b0)`

Definition at line 72 of file [BIT\\_MATH.h](#).

**4.1.2.8 CONCAT\_6BITS** `#define CONCAT_6BITS(  
 b5,  
 b4,  
 b3,  
 b2,  
 b1,  
 b0 ) (0b##b5##b4##b3##b2##b1##b0)`

Definition at line 71 of file [BIT\\_MATH.h](#).

**4.1.2.9 CONCAT\_7BITS** `#define CONCAT_7BITS(  
 b6,  
 b5,  
 b4,  
 b3,  
 b2,  
 b1,  
 b0 ) (0b##b6##b5##b4##b3##b2##b1##b0)`

Definition at line 70 of file [BIT\\_MATH.h](#).

**4.1.2.10 CONCAT\_8BITS** `#define CONCAT_8BITS(  
 b7,  
 b6,  
 b5,  
 b4,  
 b3,  
 b2,  
 b1,  
 b0 ) (0b##b7##b6##b5##b4##b3##b2##b1##b0)`

Definition at line 69 of file [BIT\\_MATH.h](#).

**4.1.2.11 GET\_BIT** `#define GET_BIT(  
 REGISTER,  
 BIT ) ( 1 & ( (REGISTER) >> (BIT) ) )`

Read state of a specific bit.

#### Parameters

in	<i>REGISTER</i>	the register includes the bit
in	<i>BIT</i>	the required bit number to be read

#### Returns

state of the bit: 1 or 0

For example:

`GET_BIT(PORT_A, PIN0)` will return 1 if bit 0 of PORT\_A is HIGH or 0 if it is LOW

Definition at line 19 of file [BIT\\_MATH.h](#).

**4.1.2.12 SET\_BIT** `#define SET_BIT(  
 REGISTER,  
 BIT ) ( (REGISTER) |= (1 << (BIT)) )`

Set state of a specific bit (set to 1)

#### Parameters

in	<i>REGISTER</i>	the register includes the bit
in	<i>BIT</i>	the required bit number to be set

For example:

`SET_BIT(PORT_A, PIN0)` will set bit 0 of PORT\_A to HIGH (1)

Definition at line 28 of file [BIT\\_MATH.h](#).

**4.1.2.13 TOG\_BIT** `#define TOG_BIT(  
 REGISTER,  
 BIT ) ( (REGISTER) ^= (1 << (BIT)) )`

Toggle state of a specific bit (set to 0)

## Parameters

in	<i>REGISTER</i>	is the register includes the bit
in	<i>BIT</i>	the required bit number to be toggled

For example:

`TOG_BIT(PORT_A, PIN0)` will toggle bit 0 of PORT\_A. So if it was HIGH, it will be LOW, and if it was LOW, it will be HIGH.

Definition at line 46 of file `BIT_MATH.h`.

## 4.2 BIT\_MATH.h

```

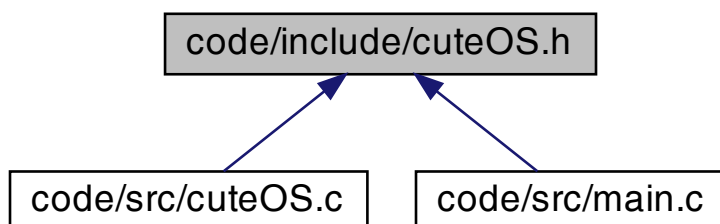
00001
00008 #ifndef BIT_MATH_H
00009 #define BIT_MATH_H
00010
00011
00019 #define GET_BIT(REGISTER, BIT)      ( 1 & ( (REGISTER) >> (BIT) ) )
00020
00021
00028 #define SET_BIT(REGISTER, BIT)      ( (REGISTER) |= (1 << (BIT)) )
00029
00030
00037 #define CLR_BIT(REGISTER, BIT)      ( (REGISTER) &= ~(1 << (BIT)) )
00038
00039
00046 #define TOG_BIT(REGISTER, BIT)      ( (REGISTER) ^= (1 << (BIT)) )
00047
00048
00056 #define BIT_IS_SET(REGISTER, Bit)    ( (REGISTER) & (1 << (Bit)) )
00057
00058
00059
00067 #define BIT_IS_CLEAR(REGISTER, Bit)  ( !( (REGISTER) & (1 << (Bit)) ) )
00068
00069 #define CONCAT_8BITS(b7, b6, b5, b4, b3, b2, b1, b0) (0b##b7##b6##b5##b4##b3##b2##b1##b0)
00070 #define CONCAT_7BITS(b6, b5, b4, b3, b2, b1, b0)    (0b##b6##b5##b4##b3##b2##b1##b0)
00071 #define CONCAT_6BITS(b5, b4, b3, b2, b1, b0)        (0b##b5##b4##b3##b2##b1##b0)
00072 #define CONCAT_5BITS(b4, b3, b2, b1, b0)            (0b##b4##b3##b2##b1##b0)
00073 #define CONCAT_4BITS(b3, b2, b1, b0)                (0b##b3##b2##b1##b0)
00074 #define CONCAT_3BITS(b2, b1, b0)                    (0b##b2##b1##b0)
00075 #define CONCAT_2BITS(b1, b0)                        (0b##b1##b0)
00076
00077 #endif          /* BIT_MATH_H */

```

## 4.3 code/include/cuteOS.h File Reference

Simple EOS interfaces header file. See [cuteOS.c](#) for more details.

This graph shows which files directly or indirectly include this file:



## Functions

- [ERROR\\_t cuteOS\\_SetCallback](#) ([ERROR\\_t](#)(\*const taskPtr)(void))  
*Set callback function for Simple EOS.*
- [ERROR\\_t cuteOS\\_Init](#) (void)  
*Sets up Timer 2 to drive the simple EOS.*
- [ERROR\\_t cuteOS\\_TaskCreate](#) ([ERROR\\_t](#)(\*const taskPtr)(void), const [u16](#) TICK\_MS)  
*Create a task with the given task function and the given tick time.*
- [ERROR\\_t cuteOS\\_TaskRemove](#) ([ERROR\\_t](#)(\*const taskPtr)(void))  
*Remove a task from the tasks array.*
- void [cuteOS\\_Start](#) (void)  
*The OS enters 'idle mode' between clock ticks to save power.*
- [ERROR\\_t cuteOS\\_SetTickTime](#) (const [u8](#) TICK\_MS)  
*Set the tick time in milliseconds.*
- [ERROR\\_t cuteOS\\_GetTickTime](#) ([u8](#) \*const TICK\_MS)  
*Get the tick time in milliseconds.*

### 4.3.1 Detailed Description

Simple EOS interfaces header file. See [cuteOS.c](#) for more details.

#### Author

Mahmoud Karam ( [ma.karam272@gmail.com](mailto:ma.karam272@gmail.com) )

#### Version

1.0.0

#### Date

2022-03-22

#### Copyright

Copyright (c) 2022

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

### 4.3.2 Function Documentation

#### 4.3.2.1 [cuteOS\\_GetTickTime\(\)](#) [ERROR\\_t](#) [cuteOS\\_GetTickTime](#) ( [u8](#) \*const *tickTimeInMsPtr* )

Get the tick time in milliseconds.

## Parameters

<i>TICK_MS</i>	pointer to the tick time in milliseconds
----------------	--

## Returns

ERROR Status: Check the options in the global enum [ERROR\\_t](#).

## Example

```
u8 tickTimeInMs;
// Get the tick time in milliseconds and store it in tickTimeInMs
cuteOS_GetTickTime(&tickTimeInMs);
```

Get the tick time in milliseconds.

Definition at line 209 of file [cuteOS.c](#).

**4.3.2.2** `cuteOS_Init()` [ERROR\\_t](#) `cuteOS_Init` (  
void )

Sets up Timer 2 to drive the simple EOS.

Initialize the Cute OS using Timer 2 overflow:

- Timer mode
- Tick time
- Interrupt enable
- Auto-reload mode

< Disable Timer 2

Enable Timer 2 (16-bit timer) and configure it as a timer and automatically reloaded its value at overflow and

< Load Timer 2 control register

< Number of timer increments required (max 65536)

< Inc = (Number of mSec) \* (Number of Instructions per mSec)

< Number of mSec = `cuteOS_TickTimeMs`

< Number of Instructions per mSec = (Number of Oscillations per mSec) \* (Number of Instructions per Oscillation)

< Number of Oscillations per mSec = `OSC_FREQ(MHz)` / 1000

< Number of Instructions per Oscillation = 1 / `OSC_PER_INST`

< 16-bit reload value

< 8-bit reload values (High & Low)

< High byte

< Low byte

< Load T2 and reload capt. reg. high bytes

< Load T2 and reload capt. reg. low bytes

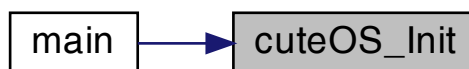
< Enable Timer 2 interrupt

< Start Timer 2

< Globally enable interrupts

Definition at line 228 of file `cuteOS.c`.

Here is the caller graph for this function:



**4.3.2.3 `cuteOS_SetCallback()`** `ERROR_t` `cuteOS_SetCallback` (  
`ERROR_t` (\*) (void) `taskPtr` )

Set callback function for Simple EOS.

Parameters

<code>taskPtr</code>	pointer to the task function
----------------------	------------------------------

**Returns**

ERROR Status: Check the options in the global enum [ERROR\\_t](#).

**Note**

This function is called by the user to set the callback function for the Simple EOS

**Example**

```
// Set the callback function for the Simple EOS to the function LED_Toggle()
cuteOS_setCallback(LED_Toggle); // LED_Toggle() is a function that toggles the LED
```

#### 4.3.2.4 **cuteOS\_SetTickTime()** [ERROR\\_t](#) cuteOS\_SetTickTime (     const [u8](#) *TICK\_MS* )

Set the tick time in milliseconds.

**Parameters**

<i>TICK_MS</i>	tick time in milliseconds
----------------	---------------------------

**Returns**

ERROR Status: Check the options in the global enum [ERROR\\_t](#).

**Example**

```
cuteOS_SetTickTime(1000); // Set the tick time to 1 second
```

Set the value of the tick time in milliseconds. So, the timing of the tasks is determined by the frequency of Timer 2 overflow. Overflow occurs every tickTimeInMs milliseconds. < Set the value of the tick time in ms

Definition at line [191](#) of file [cuteOS.c](#).

#### 4.3.2.5 **cuteOS\_Start()** void cuteOS\_Start (     void )

The OS enters 'idle mode' between clock ticks to save power.

**Note**

The next clock tick will return the processor to the normal operating state.

The OS enters 'idle mode' between clock ticks to save power. < Super loop

< Enter idle mode to save power

Definition at line 179 of file [cuteOS.c](#).

Here is the caller graph for this function:



**4.3.2.6 cuteOS\_TaskCreate()** `ERROR_t cuteOS_TaskCreate (`  
`ERROR_t (*) (void) callback,`  
`const ul6 TICK_MS )`

Create a task with the given task function and the given tick time.

**Parameters**

in	<i>taskPtr</i>	Pointer to the task function.
in	<i>TICK_MS</i>	the frequency of task execution in milliseconds.

**Returns**

ERROR Status: Check the options in the global enum [ERROR\\_t](#).

**Example**

```
cuteOS_TaskCreate(task1, 1000); // task1 will run every 1 second
cuteOS_TaskCreate(task2, 2000); // task2 will run every 2 seconds
```

This function does the following:

- Increment the task counter.
- Set the task ID.
- Set the pointer to the task function.
- Set the number of scheduler ticks after which the task will run.



Definition at line 126 of file [cuteOS.c](#).

Here is the caller graph for this function:



**4.3.2.7 cuteOS\_TaskRemove()** `ERROR_t cuteOS_TaskRemove (`  
`ERROR_t (*) (void) callback )`

Remove a task from the tasks array.

#### Parameters

in	<i>taskPtr</i>	Pointer to the task function.
----	----------------	-------------------------------

#### Returns

ERROR Status: Check the options in the global enum [ERROR\\_t](#).

#### Example

```
cuteOS_TaskRemove(task1);    // remove task1
cuteOS_TaskRemove(task2);    // remove task2
```

This function does the following:

- Search for the task in the tasks array.
- If found, remove the task from the tasks array.
- Rearrange the tasks array.
- Decrement the task counter.
- If the task is not available, an error is returned.

#### Parameters

in	<i>callback</i>	Pointer to the task function.
----	-----------------	-------------------------------

### Returns

ERROR Status: Check the options in the global enum [ERROR\\_t](#).

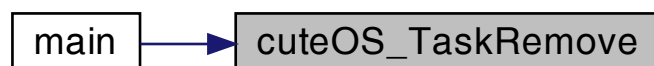
< Find the task in the task array

< Task found

< Decrement the number of tasks

Definition at line 152 of file [cuteOS.c](#).

Here is the caller graph for this function:



## 4.4 cuteOS.h

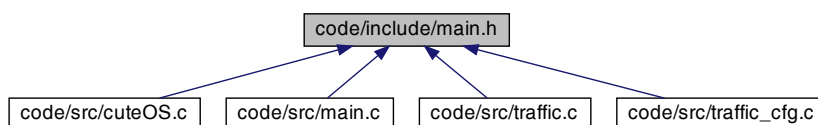
```

00001
00009 #ifndef CUTE_OS_H
00010 #define CUTE_OS_H
00011
00012
00025 ERROR_t cuteOS_SetCallback( ERROR_t (* const taskPtr)(void) );
00026
00027
00030 ERROR_t cuteOS_Init(void);
00031
00032
00042 ERROR_t cuteOS_TaskCreate(ERROR_t (* const taskPtr)(void), const u16 TICK_MS);
00043
00044
00052 ERROR_t cuteOS_TaskRemove(ERROR_t (* const taskPtr)(void));
00053
00054
00055
00059 void cuteOS_Start(void);
00060
00061
00071 ERROR_t cuteOS_SetTickTime(const u8 TICK_MS);
00072
00073
00085 ERROR_t cuteOS_GetTickTime(u8 * const TICK_MS);
00086
00087 #endif /* SIMPLE_EOS_H */
  
```

## 4.5 code/include/main.h File Reference

Project Header for [main.c](#).

This graph shows which files directly or indirectly include this file:



## Macros

- `#define OSC_FREQ (12000000UL)`
- `#define OSC_PER_INST (12)`  
*Number of oscillations per instruction (12, etc)*
- `#define INTERRUPT_Timer_0_Overflow 1`
- `#define INTERRUPT_Timer_1_Overflow 3`
- `#define INTERRUPT_Timer_2_Overflow 5`

### 4.5.1 Detailed Description

Project Header for [main.c](#).

#### Author

Mahmoud Karam ( [ma.karam272@gmail.com](mailto:ma.karam272@gmail.com))

#### Version

1.0.0

#### Date

2022-03-22

#### Copyright

Copyright (c) 2022

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

### 4.5.2 Macro Definition Documentation

#### 4.5.2.1 INTERRUPT\_Timer\_0\_Overflow `#define INTERRUPT_Timer_0_Overflow 1`

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

#### 4.5.2.2 INTERRUPT\_Timer\_1\_Overflow `#define INTERRUPT_Timer_1_Overflow 3`

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

#### 4.5.2.3 INTERRUPT\_Timer\_2\_Overflow `#define INTERRUPT_Timer_2_Overflow 5`

Definition at line 38 of file [main.h](#).

#### 4.5.2.4 OSC\_FREQ `#define OSC_FREQ (12000000UL)`

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

#### 4.5.2.5 OSC\_PER\_INST `#define OSC_PER_INST (12)`

Number of oscillations per instruction (12, etc)

Options:

- 12: Original 8051 / 8052 and numerous modern versions
- 6 : Various Infineon and Philips devices, etc.
- 4 : Dallas 320, 520 etc.
- 1 : Dallas 420, etc.

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

## 4.6 main.h

```

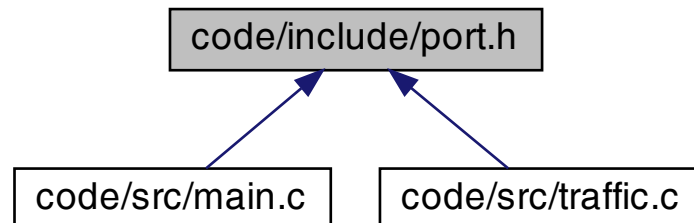
00001
00009 #ifndef MAIN_H
00010 #define MAIN_H
00011
00012 /*-----*/
00013 /* WILL NEED TO EDIT THIS SECTION FOR EVERY PROJECT */
00014 /*-----*/
00015 /* Oscillator / resonator frequency (in Hz) e.g. (11059200UL) */
00016 #define OSC_FREQ (12000000UL)
00017
00018
00026 #define OSC_PER_INST (12)
00027
00028
00029
00030
00031
00032 /*-----*/
00033 /* SHOULD NOT NEED TO EDIT THE SECTIONS BELOW */
00034 /*-----*/
00035 /* Interrupts number of Timers overflow from the vector table of the 8051 */
00036 #define INTERRUPT_Timer_0_Overflow 1
00037 #define INTERRUPT_Timer_1_Overflow 3
00038 #define INTERRUPT_Timer_2_Overflow 5
00039
00040
00041 #endif /* MAIN_H */

```

## 4.7 code/include/port.h File Reference

Port Header file for the milk pasteurization example.

This graph shows which files directly or indirectly include this file:



### Variables

- sbit `redPin` =  $P1^0$
- sbit `amberPin` =  $P1^1$
- sbit `greenPin` =  $P1^2$
- sbit `led1Pin` =  $P1^3$
- sbit `led2Pin` =  $P1^4$
- sbit `led3Pin` =  $P1^5$
- sbit `motorPin` =  $P1^6$
- sbit `buzzerPin` =  $P1^7$

### 4.7.1 Detailed Description

Port Header file for the milk pasteurization example.

#### Author

Mahmoud Karam ( [ma.karam272@gmail.com](mailto:ma.karam272@gmail.com) )

#### Version

1.0.0

#### Date

2022-03-22

#### Copyright

Copyright (c) 2022

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

## 4.7.2 Variable Documentation

### 4.7.2.1 **amberPin** `sbit amberPin = P1^1`

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

### 4.7.2.2 **buzzerPin** `sbit buzzerPin = P1^7`

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

### 4.7.2.3 **greenPin** `sbit greenPin = P1^2`

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

### 4.7.2.4 **led1Pin** `sbit led1Pin = P1^3`

In file [main.C](#)

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

### 4.7.2.5 **led2Pin** `sbit led2Pin = P1^4`

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

### 4.7.2.6 **led3Pin** `sbit led3Pin = P1^5`

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

### 4.7.2.7 **motorPin** `sbit motorPin = P1^6`

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

#### 4.7.2.8 redPin `sbit redPin = P1^0`

In file [traffic.C](#)

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

## 4.8 port.h

```

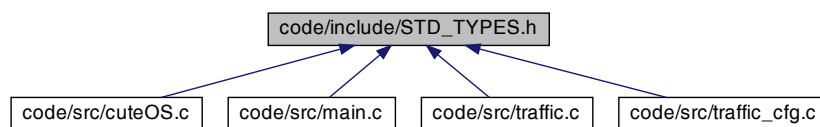
00001
00009 #ifndef PORT_H
00010 #define PORT_H
00011
00012
00015 sbit redPin    = P1^0;  /* Port 1 pin 0 */
00016 sbit amberPin  = P1^1;  /* Port 1 pin 1 */
00017 sbit greenPin  = P1^2;  /* Port 1 pin 2 */
00018
00019
00022 sbit led1Pin   = P1^3;
00023 sbit led2Pin   = P1^4;
00024 sbit led3Pin   = P1^5;
00025 sbit motorPin  = P1^6;
00026 sbit buzzerPin = P1^7;
00027
00028 #endif /* _PORT_H */

```

## 4.9 code/include/STD\_TYPES.h File Reference

Standard data types For AVR Microcontrollers.

This graph shows which files directly or indirectly include this file:



### Macros

- `#define NULL ((void *)0)`
- `#define NULL_BYTE ('\0')`

### Typedefs

- typedef signed long int [s32](#)
- typedef signed short int [s16](#)
- typedef signed char [s8](#)
- typedef unsigned long int [u32](#)
- typedef unsigned short int [u16](#)
- typedef unsigned char [u8](#)
- typedef float [f32](#)
- typedef double [f64](#)
- typedef [u16](#) [size\\_t](#)

## Enumerations

- enum `STATE_t` { `LOW` , `HIGH` , `NORMAL` }
- enum `ACTIVATION_STATUS_t` { `ACTIVE_LOW` , `ACTIVE_HIGH` }
- enum `BOOL_t` { `FALSE` , `TRUE` }
- enum `ERROR_t` {  
    `ERROR_NO` = 0 , `ERROR_YES` = 0x1 , `ERROR_TIMEOUT` = 0x2 , `ERROR_NULL_POINTER` = 0x4 ,  
    `ERROR_BUSY` = 0x8 , `ERROR_NOT_INITIALIZED` = 0x10 , `ERROR_ILLEGAL_PARAM` = 0x20 ,  
    `ERROR_OUT_OF_RANGE` = 0x40 }

### 4.9.1 Detailed Description

Standard data types For AVR Microcontrollers.

#### Author

Mahmoud Karam ( [ma.karam272@gmail.com](mailto:ma.karam272@gmail.com) )

#### Date

2022-03-20

#### Version

1.0.0

Definition in file [STD\\_TYPES.h](#).

### 4.9.2 Macro Definition Documentation

#### 4.9.2.1 NULL `#define NULL ((void *)0)`

NULL pointer

Definition at line 61 of file [STD\\_TYPES.h](#).

#### 4.9.2.2 NULL\_BYTE `#define NULL_BYTE ('\0')`

Definition at line 64 of file [STD\\_TYPES.h](#).

### 4.9.3 Typedef Documentation



**4.9.3.1 f32** typedef float f32

Definition at line 22 of file [STD\\_TYPES.h](#).

**4.9.3.2 f64** typedef double f64

Definition at line 23 of file [STD\\_TYPES.h](#).

**4.9.3.3 s16** typedef signed short int s16

Definition at line 13 of file [STD\\_TYPES.h](#).

**4.9.3.4 s32** typedef signed long int s32

Definition at line 12 of file [STD\\_TYPES.h](#).

**4.9.3.5 s8** typedef signed char s8

Definition at line 14 of file [STD\\_TYPES.h](#).

**4.9.3.6 size\_t** typedef u16 size\_t

< This is a macro defined in the C standard library <stddef.h> for the size\_t type size\_t is an unsigned integer type of the result of the sizeof operator

Definition at line 27 of file [STD\\_TYPES.h](#).

**4.9.3.7 u16** typedef unsigned short int u16

Definition at line 18 of file [STD\\_TYPES.h](#).

**4.9.3.8 u32** typedef unsigned long int u32

Definition at line 17 of file [STD\\_TYPES.h](#).

**4.9.3.9 u8** typedef unsigned char u8

Definition at line 19 of file [STD\\_TYPES.h](#).

**4.9.4 Enumeration Type Documentation****4.9.4.1 ACTIVATION\_STATUS\_t** enum ACTIVATION\_STATUS\_t

## Enumerator

ACTIVE_LOW	Active low means that the pin is pulled low when the pin is set to high
ACTIVE_HIGH	Active high means that the pin is pulled high when the pin is set to low

Definition at line 37 of file [STD\\_TYPES.h](#).

#### 4.9.4.2 **BOOL\_t** enum [BOOL\\_t](#)

## Enumerator

FALSE	
TRUE	

Definition at line 43 of file [STD\\_TYPES.h](#).

#### 4.9.4.3 **ERROR\_t** enum [ERROR\\_t](#)

## Enumerator

ERROR_NO	No error occurred
ERROR_YES	Error occurred
ERROR_TIMEOUT	Timeout occurred
ERROR_NULL_POINTER	Null pointer occurred
ERROR_BUSY	Busy state occurred
ERROR_NOT_INITIALIZED	Not initialized state occurred
ERROR_ILLEGAL_PARAM	Invalid input state occurred
ERROR_OUT_OF_RANGE	Out of range state occurred

Definition at line 48 of file [STD\\_TYPES.h](#).

#### 4.9.4.4 **STATE\_t** enum [STATE\\_t](#)

## Enumerator

LOW	
HIGH	
NORMAL	

Definition at line 31 of file [STD\\_TYPES.h](#).

## 4.10 STD\_TYPES.h

```

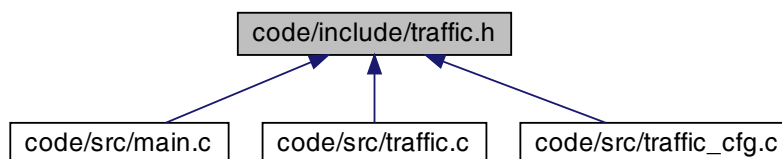
00001
00008 #ifndef STD_TYPES_H
00009 #define STD_TYPES_H
00010
00011 /* Signed integers */
00012 typedef signed long int s32;
00013 typedef signed short int s16;
00014 typedef signed char s8;
00015
00016 /* Unsigned integers */
00017 typedef unsigned long int u32;
00018 typedef unsigned short int u16;
00019 typedef unsigned char u8;
00020
00021 /* Float numbers */
00022 typedef float f32;
00023 typedef double f64;
00024
00025 /* Special types */
00026 #undef __SIZE_TYPE__
00027 typedef u16 size_t;
00028 #undef HIGH
00029 #undef LOW
00030 #undef LOW
00031 typedef enum{
00032     LOW,
00033     HIGH,
00034     NORMAL, /* Used for any normal state */
00035 }STATE_t;
00036
00037 typedef enum{
00038     ACTIVE_LOW,
00039     ACTIVE_HIGH,
00040 }ACTIVATION_STATUS_t;
00041
00042 /* Boolean type */
00043 typedef enum{
00044     FALSE,
00045     TRUE
00046 }BOOL_t;
00047
00048 typedef enum{
00049     ERROR_NO = 0,
00050     ERROR_YES = 0x1,
00051     ERROR_TIMEOUT = 0x2,
00052     ERROR_NULL_POINTER = 0x4,
00053     ERROR_BUSY = 0x8,
00054     ERROR_NOT_INITIALIZED = 0x10,
00055     ERROR_ILLEGAL_PARAM = 0x20,
00056     ERROR_OUT_OF_RANGE = 0x40,
00057 }ERROR_t;
00058
00059 /* Pointers */
00060 #undef NULL
00061 #define NULL ((void *)0)
00062 #undef NULL_BYTE
00063 #define NULL_BYTE ('\0')
00064
00065 #endif /* STD_TYPES_H */

```

## 4.11 code/include/traffic.h File Reference

Traffic Light System interfaces header file. See [traffic.c](#) for more details.

This graph shows which files directly or indirectly include this file:



## Enumerations

- enum [TRAFFIC\\_SEQUENCE\\_t](#) { [RED](#) , [RED\\_AMBER](#) , [GREEN](#) , [AMBER](#) }

## Functions

- [ERROR\\_t](#) [TRAFFIC\\_Init](#) (void)  
*Initialize the traffic light system to RED state.*
- [ERROR\\_t](#) [TRAFFIC\\_DeInit](#) (void)  
*De Initialize the traffic light system by turning off all the lights.*
- [ERROR\\_t](#) [TRAFFIC\\_Update](#) (void)
- [ERROR\\_t](#) [TRAFFIC\\_SetColor](#) (const [TRAFFIC\\_SEQUENCE\\_t](#) Copy\_color)  
*Set the traffic light color sequence to the given color sequence.*
- [ERROR\\_t](#) [TRAFFIC\\_GetColor](#) ([TRAFFIC\\_SEQUENCE\\_t](#) \*const Copy\_color)  
*Get the traffic light color sequence.*

### 4.11.1 Detailed Description

Traffic Light System interfaces header file. See [traffic.c](#) for more details.

#### Author

Mahmoud Karam ( [ma.karam272@gmail.com](mailto:ma.karam272@gmail.com) )

#### Version

1.0.0

#### Date

2022-03-22

#### Copyright

Copyright (c) 2022

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

### 4.11.2 Enumeration Type Documentation

#### 4.11.2.1 [TRAFFIC\\_SEQUENCE\\_t](#) enum [TRAFFIC\\_SEQUENCE\\_t](#)

##### Enumerator

<a href="#">RED</a>	
<a href="#">RED_AMBER</a>	
<a href="#">GREEN</a>	
<a href="#">AMBER</a>	

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

### 4.11.3 Function Documentation

**4.11.3.1 TRAFFIC\_DeInit()** [ERROR\\_t](#) TRAFFIC\_DeInit (   
void )

De Initialize the traffic light system by turning off all the lights.

#### Returns

ERROR Status: Check the options in the global enum [ERROR\\_t](#).

This function does the following:

- Turning off all the traffic lights.
- Assign the callback function of the OS delay to NULL.

< Setting traffic light to red

< Setting callback function to NULL

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

**4.11.3.2 TRAFFIC\_GetColor()** [ERROR\\_t](#) TRAFFIC\_GetColor (   
[TRAFFIC\\_SEQUENCE\\_t](#) \*const *Copy\_color* )

Get the traffic light color sequence.

#### Parameters

out	<i>Copy_color</i>	pointer to the variable to store the color sequence. Expected values: <ul style="list-style-type: none"><li>• <a href="#">RED</a></li><li>• <a href="#">RED_AMBER</a></li><li>• <a href="#">GREEN</a></li><li>• <a href="#">AMBER</a> Those colors are members of the global enumeration <a href="#">TRAFFIC_SEQUENCE_t</a>.</li></ul>
-----	-------------------	--

#### Returns

ERROR Status: Check the options in the global enum [ERROR\\_t](#).

Definition at line 171 of file [traffic.c](#).

**4.11.3.3 TRAFFIC\_Init()** `ERROR_t TRAFFIC_Init (`  
    `void )`

Initialize the traffic light system to RED state.

#### Returns

ERROR Status: Check the options in the global enum [ERROR\\_t](#).

This function does the following:

- Initialize the traffic light system to [RED](#) sequence.
- Set the callback function of the OS delay to the function [TRAFFIC\\_Update](#).

< Reset the time counter

< Initialize the colorSequence

Definition at line 128 of file [traffic.c](#).

Here is the caller graph for this function:



**4.11.3.4 TRAFFIC\_SetColor()** `ERROR_t TRAFFIC_SetColor (`  
    `const TRAFFIC_SEQUENCE_t Copy_color )`

Set the traffic light color sequence to the given color sequence.

## Parameters

in	color	sequence: The color sequence to set: <ul style="list-style-type: none"><li>• <a href="#">RED</a></li><li>• <a href="#">RED_AMBER</a></li><li>• <a href="#">GREEN</a></li><li>• <a href="#">AMBER</a> Those colors are members of the global enumeration <a href="#">TRAFFIC_SEQUENCE_t</a>.</li></ul>
----	-------	---

## Returns

ERROR Status: Check the options in the global enum [ERROR\\_t](#).

Definition at line [162](#) of file [traffic.c](#).

**4.11.3.5 TRAFFIC\_Update()** [ERROR\\_t](#) TRAFFIC\_Update (  
void )

This function does the following:

- Setting the traffic light color sequence according to the current color sequence.
- Update the OS delay for the current color sequence.
- Update the color sequence value to the next color sequence. So, when calling this function again, the color sequence will be changed.

< Switch on the current color sequence

< Illegal color sequence

Definition at line [195](#) of file [traffic.c](#).

Here is the caller graph for this function:



## 4.12 traffic.h

```

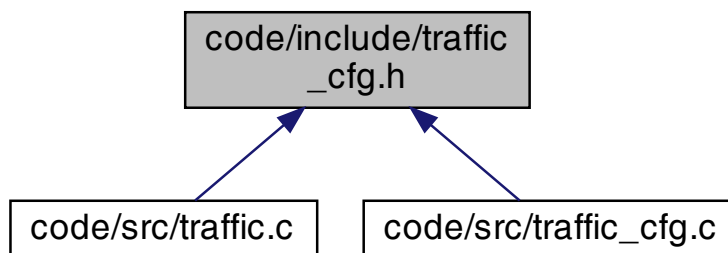
00001
00009 #ifndef TRAFFIC_H
00010 #define TRAFFIC_H
00011
00012 /*-----*/
00013 /*                                TYPE DEFINITIONS                                */
00014 /*-----*/
00015 typedef enum {
00016     RED,
00017     RED_AMBER,
00018     GREEN,
00019     AMBER
00020 }TRAFFIC_SEQUENCE_t;
00021
00022
00023
00024
00025 /*-----*/
00026 /*                                API FUNCTIONS                                */
00027 /*-----*/
00028
00029
00033 ERROR_t TRAFFIC_Init(void);
00034
00035
00039 ERROR_t TRAFFIC_DeInit(void);
00040
00041 ERROR_t TRAFFIC_Update(void);
00042
00043
00053 ERROR_t TRAFFIC_SetColor(const TRAFFIC_SEQUENCE_t Copy_color);
00054
00055
00066 ERROR_t TRAFFIC_GetColor(TRAFFIC_SEQUENCE_t * const Copy_color);
00067
00068
00069 #endif          /* TRAFFIC_H */

```

## 4.13 code/include/traffic\_cfg.h File Reference

Traffic Light System interfaces header file. See [traffic.c](#) for more details.

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [TRAFFIC\\_CONFIGS\\_t](#)



## Enumerations

- enum [TRAFFIC\\_SEQUENCE\\_DURATION\\_t](#) { [TRAFFIC\\_DURATION\\_RED](#) = 10 , [TRAFFIC\\_DURATION\\_RED\\_AMBER](#) = 2 , [TRAFFIC\\_DURATION\\_GREEN](#) = 10 , [TRAFFIC\\_DURATION\\_AMBER](#) = 2 }

## Variables

- [TRAFFIC\\_CONFIGS\\_t](#) [TRAFFIC\\_Configs](#)  
*Traffic Light System pins connections.*

### 4.13.1 Detailed Description

Traffic Light System interfaces header file. See [traffic.c](#) for more details.

#### Author

Mahmoud Karam ( [ma.karam272@gmail.com](mailto:ma.karam272@gmail.com) )

#### Version

1.0.0

#### Date

2022-03-22

#### Copyright

Copyright (c) 2022

Definition in file [traffic\\_cfg.h](#).

### 4.13.2 Enumeration Type Documentation

#### 4.13.2.1 [TRAFFIC\\_SEQUENCE\\_DURATION\\_t](#) enum [TRAFFIC\\_SEQUENCE\\_DURATION\\_t](#)

##### Enumerator

<a href="#">TRAFFIC_DURATION_RED</a>	Red light duration in seconds
<a href="#">TRAFFIC_DURATION_RED_AMBER</a>	Red-Amber light duration in seconds
<a href="#">TRAFFIC_DURATION_GREEN</a>	Green light duration in seconds
<a href="#">TRAFFIC_DURATION_AMBER</a>	Amber light duration in seconds

Definition at line 15 of file [traffic\\_cfg.h](#).

### 4.13.3 Variable Documentation

#### 4.13.3.1 TRAFFIC\_Configs `TRAFFIC_CONFIGS_t` TRAFFIC\_Configs [extern]

Traffic Light System pins connections.

Definition at line 22 of file `traffic_cfg.c`.

### 4.14 traffic\_cfg.h

```

00001
00009 #ifndef TRAFFIC_CFG_H
00010 #define TRAFFIC_CFG_H
00011
00012 /*-----*/
00013 /*          YOU CAN CHANGE THE FOLLOWING PARAMETERS          */
00014 /*-----*/
00015 typedef enum {
00016     TRAFFIC_DURATION_RED = 10,
00017     TRAFFIC_DURATION_RED_AMBER = 2,
00018     TRAFFIC_DURATION_GREEN = 10,
00019     TRAFFIC_DURATION_AMBER = 2,
00020 }TRAFFIC_SEQUENCE_DURATION_t;
00021
00022
00023
00024
00025
00026 /*-----*/
00027 /*          YOU MUST «<NOT»> CHANGE THE FOLLOWING PARAMETERS          */
00028 /*-----*/
00029 typedef struct {
00030     TRAFFIC_SEQUENCE_DURATION_t red_duration;
00031     TRAFFIC_SEQUENCE_DURATION_t red_amber_duration;
00032     TRAFFIC_SEQUENCE_DURATION_t green_duration;
00033     TRAFFIC_SEQUENCE_DURATION_t amber_duration;
00034 }TRAFFIC_CONFIGS_t;
00035
00036 extern TRAFFIC_CONFIGS_t TRAFFIC_Configs;
00037
00038 #endif /* TRAFFIC_CFG_H */

```

### 4.15 code/src/cuteOS.c File Reference

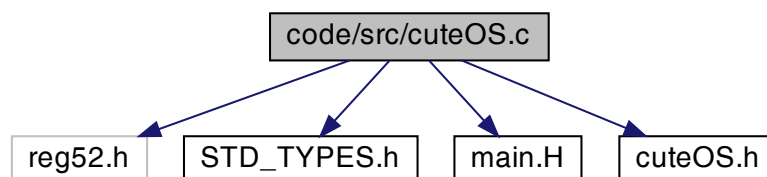
Main file for Cute Embedded Operating System (cuteOS) for 8051.

```

#include <reg52.h>
#include "STD_TYPES.h"
#include "main.H"
#include "cuteOS.h"

```

Include dependency graph for cuteOS.c:



## Data Structures

- struct [cuteOS\\_TASK\\_t](#)

## Macros

- `#define MAX_TICK_TIME_MS 65`  
*Maximum tick time in milliseconds.*
- `#define MAX_TASKS_NUM 10`  
*Maximum number of tasks the OS can handle.*

## Functions

- [ERROR\\_t](#) [cuteOS\\_TaskCreate](#) ([ERROR\\_t](#)(\*const callback)(void), const [u16](#) TICK\_MS)  
*Create a task with the given task function and the given tick time.*
- [ERROR\\_t](#) [cuteOS\\_TaskRemove](#) ([ERROR\\_t](#)(\*const callback)(void))  
*Remove a task from the tasks array.*
- void [cuteOS\\_Start](#) (void)  
*Start the Cute Embedded Operating System (cuteOS)*
- [ERROR\\_t](#) [cuteOS\\_SetTickTime](#) (const [u8](#) TICK\_MS)  
*Set the tick time in milliseconds.*
- [ERROR\\_t](#) [cuteOS\\_GetTickTime](#) ([u8](#) \*const tickTimeInMsPtr)  
*Get the value of the tick time in milliseconds.*
- [ERROR\\_t](#) [cuteOS\\_Init](#) (void)  
*Sets up Timer 2 to drive the simple EOS.*

## Variables

- [cuteOS\\_TASK\\_t](#) [tasks](#) [[MAX\\_TASKS\\_NUM](#)] = {0}

### 4.15.1 Detailed Description

Main file for Cute Embedded Operating System (cuteOS) for 8051.

#### Author

Mahmoud Karam ( [ma.karam272@gmail.com](mailto:ma.karam272@gmail.com) )

cuteOS schedules the tasks in a cooperative manner. It invokes the scheduler ([cuteOS\\_ISR\(\)](#)) periodically by Timer overflow. So, the timing of the tasks is determined by the frequency of Timer overflow defined by the variable [cuteOS\\_TICK\\_TIME](#).

#### Note

cuteOS uses the timer2 for scheduling.

#### Version

1.0.0

**Date**

2022-03-22

**Copyright**

Copyright (c) 2022

**Application usage:**

- At [main.c](#):
  1. Initialize the Cute OS.  
`cuteOS_Init();`
  2. Initialize the tasks.  
`cuteOS_TaskCreate(task1, 1000); // task1 will run every 1 second`  
`cuteOS_TaskCreate(task2, 2000); // task2 will run every 2 seconds`
  3. Start the Cute OS scheduler.  
`cuteOS_Start();`

Definition in file [cuteOS.c](#).**4.15.2 Macro Definition Documentation****4.15.2.1 MAX\_TASKS\_NUM** `#define MAX_TASKS_NUM 10`

Maximum number of tasks the OS can handle.

Number of tasks created by the user.

Definition at line 55 of file [cuteOS.c](#).**4.15.2.2 MAX\_TICK\_TIME\_MS** `#define MAX_TICK_TIME_MS 65`

Maximum tick time in milliseconds.

This variable is used to set the maximum tick time in milliseconds. The maximum tick time is used to set the maximum time of the tasks. It has a maximum value of 65 ms because:

1. The maximum value of the timer 2 is 65535 (16-bit timer).
2. The 8051 microcontroller has 1 MIPS (1 million instructions per second), with 12MHz clock, and 12 clock cycles per instruction. So, the maximum tick time =  $(65535 * 12) / 12000000 = 65$  ms. Tick time in ms (must be less than MAX\_TICK\_TIME\_MS).

Definition at line 44 of file [cuteOS.c](#).

### 4.15.3 Function Documentation

**4.15.3.1 cuteOS\_GetTickTime()** `ERROR_t cuteOS_GetTickTime ( u8 *const tickTimeInMsPtr )`

Get the value of the tick time in milliseconds.

Get the tick time in milliseconds.

Definition at line 209 of file [cuteOS.c](#).

**4.15.3.2 cuteOS\_Init()** `ERROR_t cuteOS_Init ( void )`

Sets up Timer 2 to drive the simple EOS.

Initialize the Cute OS using Timer 2 overflow:

- Timer mode
- Tick time
- Interrupt enable
- Auto-reload mode

< Disable Timer 2

Enable Timer 2 (16-bit timer) and configure it as a timer and automatically reloaded its value at overflow and

< Load Timer 2 control register

< Number of timer increments required (max 65536)

< Inc = (Number of mSec) \* (Number of Instructions per mSec)

< Number of mSec = cuteOS\_TickTimeMs

< Number of Instructions per mSec = (Number of Oscillations per mSec) \* (Number of Instructions per Oscillation)

< Number of Oscillations per mSec = [OSC\\_FREQ\(MHz\)](#) / 1000

< Number of Instructions per Oscillation = 1 / OSC\_PER\_INST

< 16-bit reload value

< 8-bit reload values (High & Low)

< High byte

< Low byte

< Load T2 and reload capt. reg. high bytes

< Load T2 and reload capt. reg. low bytes

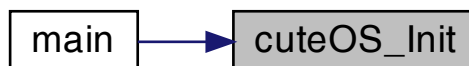
< Enable Timer 2 interrupt

< Start Timer 2

< Globally enable interrupts

Definition at line [228](#) of file [cuteOS.c](#).

Here is the caller graph for this function:



**4.15.3.3 cuteOS\_SetTickTime()** `ERROR_t cuteOS_SetTickTime (`  
`const u8 TICK_MS )`

Set the tick time in milliseconds.

Set the value of the tick time in milliseconds. So, the timing of the tasks is determined by the frequency of Timer 2 overflow. Overflow occurs every tickTimeInMs milliseconds. < Set the value of the tick time in ms

Definition at line [191](#) of file [cuteOS.c](#).

**4.15.3.4** `cuteOS_Start()` `void cuteOS_Start (`  
`void )`

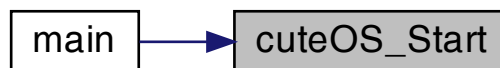
Start the Cute Embedded Operating System (cuteOS)

The OS enters 'idle mode' between clock ticks to save power. < Super loop

< Enter idle mode to save power

Definition at line 179 of file [cuteOS.c](#).

Here is the caller graph for this function:



**4.15.3.5** `cuteOS_TaskCreate()` `ERROR_t cuteOS_TaskCreate (`  
`ERROR_t (*) (void) callback,`  
`const u16 TICK_MS )`

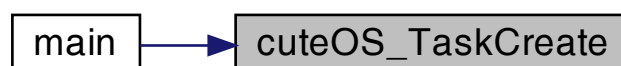
Create a task with the given task function and the given tick time.

This function does the following:

- Increment the task counter.
- Set the task ID.
- Set the pointer to the task function.
- Set the number of scheduler ticks after which the task will run.

Definition at line 126 of file [cuteOS.c](#).

Here is the caller graph for this function:



**4.15.3.6 `cuteOS_TaskRemove()`** `ERROR_t` `cuteOS_TaskRemove` (  
`ERROR_t (*) (void)` `callback` )

Remove a task from the tasks array.

This function does the following:

- Search for the task in the tasks array.
- If found, remove the task from the tasks array.
- Rearrange the tasks array.
- Decrement the task counter.
- If the task is not available, an error is returned.

#### Parameters

<code>in</code>	<code>callback</code>	Pointer to the task function.
-----------------	-----------------------	-------------------------------

#### Returns

ERROR Status: Check the options in the global enum `ERROR_t`.

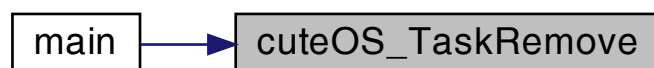
< Find the task in the task array

< Task found

< Decrement the number of tasks

Definition at line 152 of file `cuteOS.c`.

Here is the caller graph for this function:



## 4.15.4 Variable Documentation

**4.15.4.1 `tasks`** `cuteOS_TASK_t` `tasks`[`MAX_TASKS_NUM`] = {0}

Definition at line 66 of file `cuteOS.c`.



## 4.16 cuteOS.c

```

00001
00023 #include <reg52.h>
00024 #include "STD_TYPES.h"
00025 #include "main.H"
00026 #include "cuteOS.h"
00027
00028
00029 /*-----*/
00030 /*                                     PRIVATE DATA                                     */
00031 /*-----*/
00032
00042 #define MAX_TICK_TIME_MS 65
00043
00045 static u8 cuteOS_TickTimeMs = 50;
00046
00048 static u16 cuteOS_TickCount = 0;
00049
00050
00053 #define MAX_TASKS_NUM 10
00054
00056 static u8 cuteOS_TaskCounter = 0;
00059 typedef struct {
00060     ERROR_t (*callback) (void);
00061     u16 ticks;
00062     u8 id;
00063 }cuteOS_TASK_t;
00064
00066 cuteOS_TASK_t tasks[MAX_TASKS_NUM] = {0};
00067
00068
00069
00070
00071
00072
00073
00074 /*-----*/
00075 /*                                     PRIVATE FUNCTIONS                                     */
00076 /*-----*/
00077
00081 static void cuteOS_ISR() interrupt INTERRUPT_Timer_2_Overflow {
00082     u8 i = 0;
00083
00085     TF2 = 0;
00086
00088     ++cuteOS_TickCount;
00089
00091     for(i = 0; i < cuteOS_TaskCounter; ++i) {
00092         if( (cuteOS_TickCount % tasks[i].ticks) == 0) {
00094             // cuteOS_TickCount = 0;
00095
00097             if(tasks[i].callback != NULL) {
00098                 tasks[i].callback();
00099             }
00100         }
00101     }
00102 }
00103
00104
00110 static void cuteOS_Sleep(void) {
00112     PCON |= 0x01;
00113 }
00114
00115
00116 /*-----*/
00117 /*                                     PUBLIC FUNCTIONS                                     */
00118 /*-----*/
00119
00126 ERROR_t cuteOS_TaskCreate(ERROR_t (* const callback) (void), const u16 TICK_MS) {
00127     ERROR_t error = ERROR_NO;
00128
00129     ++cuteOS_TaskCounter;
00130
00131     if(cuteOS_TaskCounter > MAX_TASKS_NUM) {
00132         error = ERROR_OUT_OF_RANGE;
00133     } else {
00134         tasks[cuteOS_TaskCounter - 1].id = cuteOS_TaskCounter - 1;
00135         tasks[cuteOS_TaskCounter - 1].ticks = TICK_MS / cuteOS_TickTimeMs;
00136         tasks[cuteOS_TaskCounter - 1].callback = callback;
00137     }
00138
00139     return error;
00140 }
00141
00142
00152 ERROR_t cuteOS_TaskRemove(ERROR_t (* const callback) (void)) {

```

```

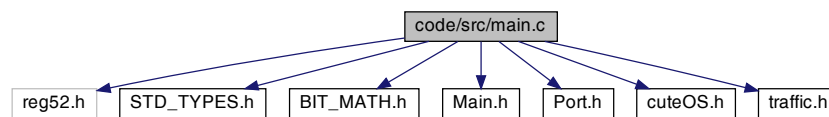
00153     ERROR_t error = ERROR_YES;
00154     u8 i = 0;
00155
00157     for(i = 0; i < cuteOS_TaskCounter; ++i) {
00158         if(tasks[i].callback == callback) {
00159             error = ERROR_NO;
00161             for(; i < cuteOS_TaskCounter - 1; ++i) {
00162                 tasks[i] = tasks[i + 1];
00163             }
00165             tasks[cuteOS_TaskCounter - 1].callback = NULL;
00166
00168             --cuteOS_TaskCounter;
00169             break;
00170         }
00171     }
00172
00173     return error;
00174 }
00175
00176
00179 void cuteOS_Start(void) {
00181     while(1) {
00182         cuteOS_Sleep();
00183     }
00184 }
00185
00186
00191 ERROR_t cuteOS_SetTickTime(const u8 TICK_MS){
00192     ERROR_t error = ERROR_NO;
00193
00194     cuteOS_TickTimeMs = TICK_MS;
00195
00196     if(cuteOS_TickTimeMs > MAX_TICK_TIME_MS) {
00197         error = ERROR_OUT_OF_RANGE;
00198     } else {
00200         cuteOS_Init();
00201     }
00202
00203     return ERROR_NO;
00204 }
00205
00206
00209 ERROR_t cuteOS_GetTickTime(u8 * const tickTimeInMsPtr){
00210     ERROR_t error = ERROR_NO;
00211
00212     if(tickTimeInMsPtr != NULL) {
00213         *tickTimeInMsPtr = cuteOS_TickTimeMs;
00214     } else {
00215         error |= ERROR_NULL_POINTER;
00216     }
00217
00218     return error;
00219 }
00220
00221
00228 ERROR_t cuteOS_Init(void) {
00229     ERROR_t error = ERROR_NO;
00230     u32 Inc;
00231     u16 Reload_16;
00232     u8 Reload_08H, Reload_08L;
00233
00234     TR2 = 0;
00236
00240     T2CON = 0x04;
00248     Inc = ((u32)cuteOS_TickTimeMs * (OSC_FREQ/1000)) / (u32)OSC_PER_INST;
00249
00251     Reload_16 = (u16) (65536UL - Inc);
00252
00254     Reload_08H = (u8)(Reload_16 / 256);
00255     Reload_08L = (u8)(Reload_16 % 256);
00257     // Used for manually checking timing (in simulator)
00258     //P2 = Reload_08H;
00259     //P3 = Reload_08L;
00260     RCAP2H = TH2 = Reload_08H;
00261     RCAP2L = TL2 = Reload_08L;
00263     ET2 = 1;
00264     TR2 = 1;
00265     EA = 1;
00267     return error;
00268 }
00269

```

## 4.17 code/src/main.c File Reference

Testing cute OS.

```
#include <reg52.h>
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "Main.h"
#include "Port.h"
#include "cuteOS.h"
#include "traffic.h"
Include dependency graph for main.c:
```



### Functions

- void [led1\\_toggle](#) (void)
- void [led2\\_toggle](#) (void)
- void [led3\\_toggle](#) (void)
- void [motor\\_toggle](#) (void)
- void [buzzer\\_toggle](#) (void)
- void [Init\\_Others](#) (void)
- void [main](#) (void)

#### 4.17.1 Detailed Description

Testing cute OS.

##### Author

Mahmoud Karam ( [ma.karam272@gmail.com](mailto:ma.karam272@gmail.com) )

Traffic light system (Chapter 8 - Embedded C by Professor j. Pont).  
See [traffic.c](#) for the implementation and sequence of the system.

##### Version

1.0.0

##### Date

2022-03-24

## Copyright

Copyright (c) 2022

Application usage:

- Run the application.
- The application will run the traffic light system.
- The system will run in a loop:
  1. Red for some seconds, then
  2. Red-Amber for some seconds, then
  3. Green for some seconds, then
  4. Amberfor some seconds, then
  5. Repeat from step 1. The duration of each state is defined in the enum [TRAFFIC\\_SEQUENCE\\_DURATION\\_t](#) in [traffic\\_cfg.h](#) file.

Code explanation:

1. Initialize the Cute OS.

```
cuteOS_Init();
```

Initialize the tasks.

```
TRAFFIC_Init();
```

See [TRAFFIC\\_Init\(\)](#) for more details.

1. Create the tasks.

```
cuteOS_TaskCreate(task1, 1000); // task1 will run every 1 second  
cuteOS_TaskCreate(task2, 2000); // task2 will run every 2 seconds
```

Start the Cute OS scheduler.

```
cuteOS_Start();
```

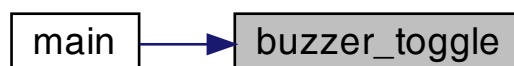
Definition in file [main.c](#).

## 4.17.2 Function Documentation

**4.17.2.1 buzzer\_toggle()** `void buzzer_toggle (  
void )`

Definition at line [63](#) of file [main.c](#).

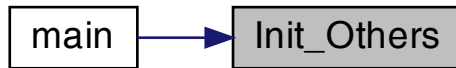
Here is the caller graph for this function:



**4.17.2.2 Init\_Others()** `void Init_Others (`  
    `void )`

Definition at line 67 of file [main.c](#).

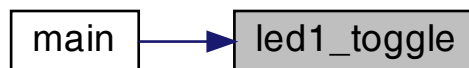
Here is the caller graph for this function:



**4.17.2.3 led1\_toggle()** `void led1_toggle (`  
    `void )`

Definition at line 47 of file [main.c](#).

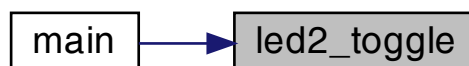
Here is the caller graph for this function:



**4.17.2.4 led2\_toggle()** `void led2_toggle (`  
    `void )`

Definition at line 51 of file [main.c](#).

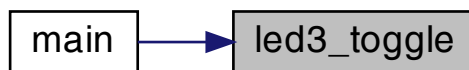
Here is the caller graph for this function:



**4.17.2.5 led3\_toggle()** `void led3_toggle (`  
`void )`

Definition at line 55 of file [main.c](#).

Here is the caller graph for this function:



**4.17.2.6 main()** `void main (`  
`void )`

< Initialize Cute OS

< Initialize the traffic light system

< Initialize other peripherals

< Create the tasks

< Create a task to run the traffic light system

< Create a task to toggle the first LED

< Create a task to toggle the second LED

< Create a task to toggle the third LED

< Create a task to toggle the buzzer

< Create a task to toggle the motor

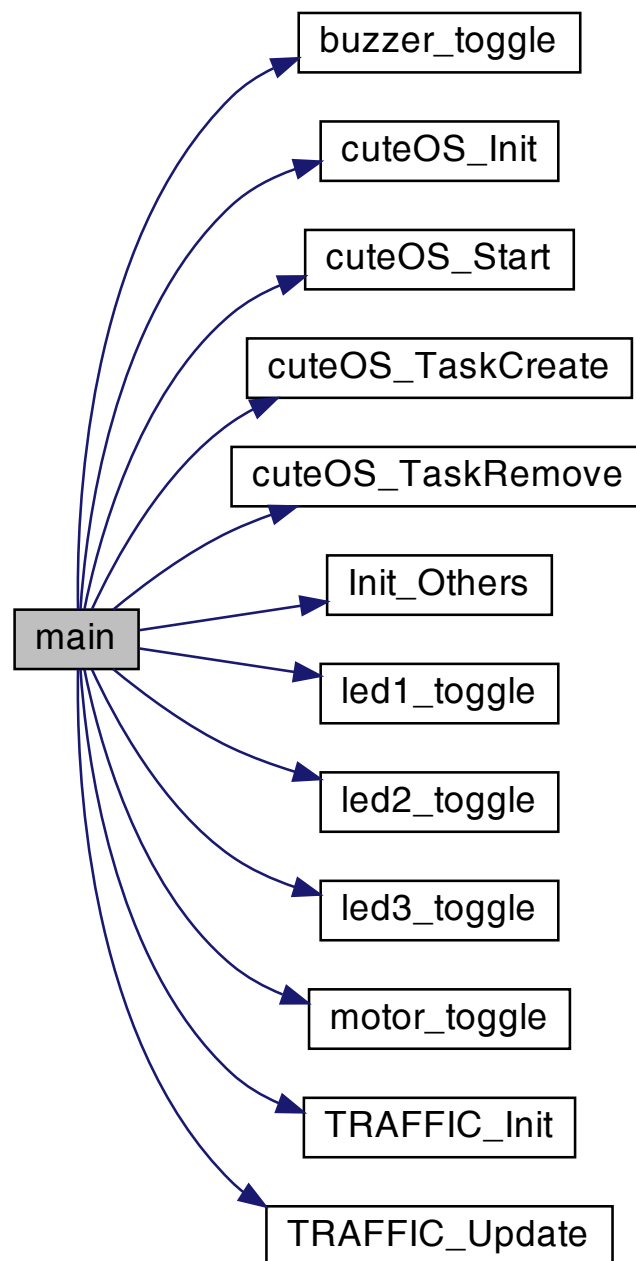
< Remove the task to toggle the buzzer

< Start the Cute OS scheduler

< The scheduler will never return from here

Definition at line 83 of file [main.c](#).

Here is the call graph for this function:



**4.17.2.7 motor\_toggle()** `void motor_toggle (`  
`void )`

Definition at line 59 of file [main.c](#).

Here is the caller graph for this function:



## 4.18 main.c

```

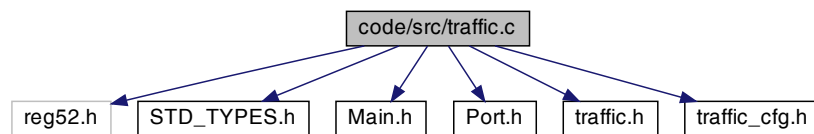
00001
00035 #include <reg52.h>
00036 #include "STD_TYPES.h"
00037 #include "BIT_MATH.h"
00038 #include "Main.h"
00039 #include "Port.h"
00040 #include "cuteOS.h"
00041 #include "traffic.h"
00042
00043 /*-----*/
00044 /* THE FOLLOWING ARE ONLY FOR TESTING THE SIMPLE OS. */
00045 /* DO NOT USE THEM IN YOUR APPLICATION. */
00046 /*-----*/
00047 void led1_toggle(void){
00048     led1Pin = !led1Pin;
00049 }
00050
00051 void led2_toggle(void){
00052     led2Pin = !led2Pin;
00053 }
00054
00055 void led3_toggle(void){
00056     led3Pin = !led3Pin;
00057 }
00058
00059 void motor_toggle(void){
00060     motorPin = !motorPin;
00061 }
00062
00063 void buzzer_toggle(void){
00064     buzzerPin = !buzzerPin;
00065 }
00066
00067 void Init_Others(void) {
00068     led1Pin = 1;
00069     led2Pin = 0;
00070     led3Pin = 0;
00071     motorPin = 1;
00072     buzzerPin = 0;
00073 }
00074
00075
00076
00077
00078
00079
00080 /*-----*/
00081 /* APPLICATION MAIN FUNCTION */
00082 /*-----*/
00083 void main(void) {
00084     /* Initialize the system */
00085     cuteOS_Init();
00086     TRAFFIC_Init();
00087     Init_Others();
00090     cuteOS_TaskCreate(TRAFFIC_Update, 1000);
00091     cuteOS_TaskCreate(led1_toggle, 1000);
00092     cuteOS_TaskCreate(led2_toggle, 2000);
00093     cuteOS_TaskCreate(led3_toggle, 4000);
00094     cuteOS_TaskCreate(buzzer_toggle, 1000);
00095     cuteOS_TaskCreate(motor_toggle, 5000);
00097     cuteOS_TaskRemove(buzzer_toggle);
00100     cuteOS_Start();
00101
00103     while(1);
00104 }
  
```



## 4.19 code/src/traffic.c File Reference

This is a traffic Light project (Chapter 8 - Embedded C by Professor j. Pont).

```
#include <reg52.h>
#include "STD_TYPES.h"
#include "Main.h"
#include "Port.h"
#include "traffic.h"
#include "traffic_cfg.h"
Include dependency graph for traffic.c:
```



### Functions

- [ERROR\\_t TRAFFIC\\_Init](#) (void)  
*Initialize the traffic light system to RED state.*
- [ERROR\\_t TRAFFIC\\_DeInit](#) (void)  
*De Initialize the traffic light system by turning off all the lights.*
- [ERROR\\_t TRAFFIC\\_SetColor](#) (const [TRAFFIC\\_SEQUENCE\\_t](#) Copy\_color)  
*Set the traffic light color sequence to the given color sequence.*
- [ERROR\\_t TRAFFIC\\_GetColor](#) ([TRAFFIC\\_SEQUENCE\\_t](#) \*const Copy\_color)  
*Get the traffic light color sequence.*

### 4.19.1 Detailed Description

This is a traffic Light project (Chapter 8 - Embedded C by Professor j. Pont).

#### Author

Mahmoud Karam ( [ma.karam272@gmail.com](mailto:ma.karam272@gmail.com) )

A basic version of the traffic-light sequencer requires no inputs from the environment and will perform well by executing a sequence of pre-determined manoeuvres. It is a classic example of a Multi-State (Timed) system.

#### Version

1.0.0

#### Date

2022-03-22

#### Copyright

Copyright (c) 2022

Definition in file [traffic.c](#).

## 4.19.2 Function Documentation

### 4.19.2.1 **TRAFFIC\_DeInit()** `ERROR_t TRAFFIC_DeInit ( void )`

De Initialize the traffic light system by turning off all the lights.

This function does the following:

- Turning off all the traffic lights.
- Assign the callback function of the OS delay to NULL.

< Setting traffic light to red

< Setting callback function to NULL

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

### 4.19.2.2 **TRAFFIC\_GetColor()** `ERROR_t TRAFFIC_GetColor ( TRAFFIC_SEQUENCE_t *const Copy_color )`

Get the traffic light color sequence.

#### Parameters

out	<i>Copy_color</i>	pointer to the variable to store the color sequence. Expected values: <ul style="list-style-type: none"><li>• <a href="#">RED</a></li><li>• <a href="#">RED_AMBER</a></li><li>• <a href="#">GREEN</a></li><li>• <a href="#">AMBER</a> Those colors are members of the global enumeration <a href="#">TRAFFIC_SEQUENCE_t</a>.</li></ul>
-----	-------------------	--

#### Returns

ERROR Status: Check the options in the global enum [ERROR\\_t](#).

Definition at line 171 of file [traffic.c](#).

#### 4.19.2.3 TRAFFIC\_Init() `ERROR_t TRAFFIC_Init ( void )`

Initialize the traffic light system to RED state.

This function does the following:

- Initialize the traffic light system to [RED](#) sequence.
- Set the callback function of the OS delay to the function [TRAFFIC\\_Update](#).

< Reset the time counter

< Initialize the colorSequence

Definition at line [128](#) of file [traffic.c](#).

Here is the caller graph for this function:



#### 4.19.2.4 TRAFFIC\_SetColor() `ERROR_t TRAFFIC_SetColor ( const TRAFFIC_SEQUENCE_t Copy_color )`

Set the traffic light color sequence to the given color sequence.

##### Parameters

in	color	sequence: The color sequence to set:
		<ul style="list-style-type: none"> <li>• <a href="#">RED</a></li> <li>• <a href="#">RED_AMBER</a></li> <li>• <a href="#">GREEN</a></li> <li>• <a href="#">AMBER</a> Those colors are members of the global enumeration <a href="#">TRAFFIC_SEQUENCE_t</a>.</li> </ul>

##### Returns

ERROR Status: Check the options in the global enum [ERROR\\_t](#).

Definition at line [162](#) of file [traffic.c](#).

## 4.20 traffic.c

```

00001
00014 #include <reg52.h>
00015 #include "STD_TYPES.h"
00016 #include "Main.h"
00017 #include "Port.h"
00018 #include "traffic.h"
00019 #include "traffic_cfg.h"
00020
00021 /*-----*/
00022 /*                                PRIVATE DATA                                */
00023 /*-----*/
00024 static TRAFFIC_SEQUENCE_t colorSequence = RED;
00025 static u16 timeInState = 0;
00026
00027 /*-----*/
00028 /*                                PRIVATE FUNCTIONS PROTOTYPES                                */
00029 /*-----*/
00030 /*-----*/
00031
00039 static ERROR_t TRAFFIC_Update(void);
00040
00041
00050 static ERROR_t TRAFFIC_RedSequence(void);
00051
00052
00061 static ERROR_t TRAFFIC_RedAmberSequence(void);
00062
00063
00072 static ERROR_t TRAFFIC_GreenSequence(void);
00073
00074
00083 static ERROR_t TRAFFIC_AmberSequence(void);
00084
00085
00111 static ERROR_t TRAFFIC_GenericSequence(const STATE_t red, const STATE_t amber, const STATE_t green,
    TRAFFIC_SEQUENCE_DURATION_t duration);
00112
00113
00114
00115
00116
00117
00118
00119 /*-----*/
00120 /*                                PUBLIC FUNCTIONS                                */
00121 /*-----*/
00122
00128 ERROR_t TRAFFIC_Init(void) {
00129     ERROR_t error = ERROR_NO;
00130
00132     timeInState = 0;
00133
00135     colorSequence = RED;
00136     redPin = HIGH;
00137     amberPin = LOW;
00138     greenPin = LOW;
00139
00140     return error;
00141 }
00142
00143
00148 ERROR_t TRAFFIC_DeInit(void) {
00149     ERROR_t error = ERROR_NO;
00150
00152     redPin = LOW;
00153     amberPin = LOW;
00154     greenPin = LOW;
00155
00157     //cuteOS_(NULL);
00158
00159     return error;
00160 }
00161
00162 ERROR_t TRAFFIC_SetColor(const TRAFFIC_SEQUENCE_t Copy_color) {
00163     ERROR_t error = ERROR_NO;
00164     colorSequence = Copy_color;
00165
00166     error |= TRAFFIC_Update();
00167
00168     return error;
00169 }
00170
00171 ERROR_t TRAFFIC_GetColor(TRAFFIC_SEQUENCE_t * const Copy_color) {
00172     ERROR_t error = ERROR_NO;
00173
00174     *Copy_color = colorSequence;
00175

```

```

00176     return error;
00177 }
00178
00179
00180
00181
00182
00183
00184 /*-----*/
00185 /*          PRIVATE FUNCTIONS DEFINITIONS          */
00186 /*-----*/
00187
00195 ERROR_t TRAFFIC_Update(void) {
00196     ERROR_t error = ERROR_NO;
00197
00199     switch(colorSequence) {
00200         case RED:;
00201             error |= TRAFFIC_RedSequence();
00202             break;
00203         case RED_AMBER:;
00204             error |= TRAFFIC_RedAmberSequence();
00205             break;
00206         case GREEN:;
00207             error |= TRAFFIC_GreenSequence();
00208             break;
00209         case AMBER:;
00210             error |= TRAFFIC_AmberSequence();
00211             break;
00212         default:;
00213             error |= ERROR_ILLEGAL_PARAM;
00214             break;
00215     }
00216
00217     return error;
00218 }
00219
00220 static ERROR_t TRAFFIC_RedSequence(void) {
00221     ERROR_t error = ERROR_NO;
00222
00223     error |= TRAFFIC_GenericSequence(HIGH, LOW, LOW, TRAFFIC_Configs.red_duration);
00224
00225     return error;
00226 }
00227
00228 static ERROR_t TRAFFIC_RedAmberSequence(void) {
00229     ERROR_t error = ERROR_NO;
00230
00231     error |= TRAFFIC_GenericSequence(HIGH, HIGH, LOW, TRAFFIC_Configs.red_amber_duration);
00232
00233     return error;
00234 }
00235
00236 static ERROR_t TRAFFIC_GreenSequence(void) {
00237     ERROR_t error = ERROR_NO;
00238
00239     error |= TRAFFIC_GenericSequence(LOW, LOW, HIGH, TRAFFIC_Configs.green_duration);
00240
00241     return error;
00242 }
00243
00244 static ERROR_t TRAFFIC_AmberSequence(void) {
00245     ERROR_t error = ERROR_NO;
00246
00247     error |= TRAFFIC_GenericSequence(LOW, HIGH, LOW, TRAFFIC_Configs.amber_duration);
00248
00249     return error;
00250 }
00251
00252
00260 static ERROR_t TRAFFIC_GenericSequence(const STATE_t redState, const STATE_t amberState, const STATE_t
greenState, TRAFFIC_SEQUENCE_DURATION_t duration) {
00261     ERROR_t error = ERROR_NO;
00262     u8 tickTime = 0;
00263
00265     if(++timeInState >= duration) {
00266         timeInState = 0;
00267         switch(colorSequence) {
00268             case RED:
00269                 colorSequence = RED_AMBER;
00270                 redPin = HIGH;
00271                 amberPin = HIGH;
00272                 greenPin = LOW;
00273                 break;
00274             case RED_AMBER:
00275                 colorSequence = GREEN;
00276                 redPin = LOW;
00277                 amberPin = LOW;

```

```

00278         greenPin = HIGH;
00279         break;
00280     case GREEN:
00281         colorSequence = AMBER;
00282         redPin = LOW;
00283         amberPin = HIGH;
00284         greenPin = LOW;
00285         break;
00286     case AMBER:
00287         colorSequence = RED;
00288         redPin = HIGH;
00289         amberPin = LOW;
00290         greenPin = LOW;
00291         break;
00292     default:
00293         error |= ERROR_ILLEGAL_PARAM;
00294         break;
00295     }
00296 } else {
00297     redPin = redState;
00298     amberPin = amberState;
00299     greenPin = greenState;
00300 }
00301
00302 return error;
00303 }

```

## 4.21 code/src/traffic\_cfg.c File Reference

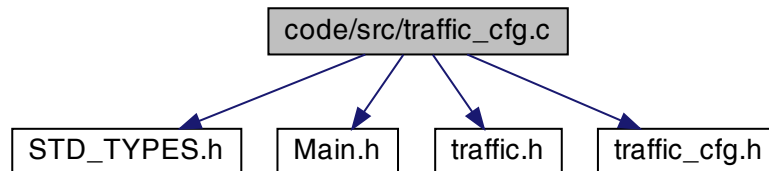
Configurations of Traffic Light System.

```

#include "STD_TYPES.h"
#include "Main.h"
#include "traffic.h"
#include "traffic_cfg.h"

```

Include dependency graph for traffic\_cfg.c:



### Variables

- [TRAFFIC\\_CONFIGS\\_t TRAFFIC\\_Configs](#)

*Traffic Light System pins connections.*

### 4.21.1 Detailed Description

Configurations of Traffic Light System.

#### Author

Mahmoud Karam ( [ma.karam272@gmail.com](mailto:ma.karam272@gmail.com) )

**Version**

1.0.0

**Date**

2022-03-22

**Copyright**

Copyright (c) 2022

Definition in file [traffic\\_cfg.c](#).**4.21.2 Variable Documentation****4.21.2.1 TRAFFIC\_Configs** [TRAFFIC\\_CONFIGS\\_t](#) TRAFFIC\_Configs**Initial value:**

```
= {  
    TRAFFIC_DURATION_RED,  
    TRAFFIC_DURATION_AMBER,  
    TRAFFIC_DURATION_GREEN,  
    TRAFFIC_DURATION_RED_AMBER,  
}
```

Traffic Light System pins connections.

Definition at line 22 of file [traffic\\_cfg.c](#).**4.22 traffic\_cfg.c**

```
00001  
00009 #include "STD_TYPES.h"  
00010 #include "Main.h"  
00011 #include "traffic.h"  
00012 #include "traffic_cfg.h"  
00013  
00014  
00015 /*-----*/  
00016 /*          YOU CAN CHANGE THE FOLLOWING PARAMETERS          */  
00017 /*-----*/  
00018  
00022 TRAFFIC_CONFIGS_t TRAFFIC_Configs = {  
00023     TRAFFIC_DURATION_RED,  
00024     TRAFFIC_DURATION_AMBER,  
00025     TRAFFIC_DURATION_GREEN,  
00026     TRAFFIC_DURATION_RED_AMBER,  
00027 };
```





## Index

- ACTIVATION\_STATUS\_t
  - STD\_TYPES.h, [23](#)
- ACTIVE\_HIGH
  - STD\_TYPES.h, [24](#)
- ACTIVE\_LOW
  - STD\_TYPES.h, [24](#)
- AMBER
  - traffic.h, [26](#)
- amber\_duration
  - TRAFFIC\_CONFIGS\_t, [3](#)
- amberPin
  - port.h, [20](#)
- BIT\_IS\_CLEAR
  - BIT\_MATH.h, [5](#)
- BIT\_IS\_SET
  - BIT\_MATH.h, [5](#)
- BIT\_MATH.h
  - BIT\_IS\_CLEAR, [5](#)
  - BIT\_IS\_SET, [5](#)
  - CLR\_BIT, [6](#)
  - CONCAT\_2BITS, [6](#)
  - CONCAT\_3BITS, [6](#)
  - CONCAT\_4BITS, [6](#)
  - CONCAT\_5BITS, [7](#)
  - CONCAT\_6BITS, [7](#)
  - CONCAT\_7BITS, [7](#)
  - CONCAT\_8BITS, [7](#)
  - GET\_BIT, [7](#)
  - SET\_BIT, [8](#)
  - TOG\_BIT, [8](#)
- BOOL\_t
  - STD\_TYPES.h, [24](#)
- buzzer\_toggle
  - main.c, [42](#)
- buzzerPin
  - port.h, [20](#)
- callback
  - cuteOS\_TASK\_t, [2](#)
- CLR\_BIT
  - BIT\_MATH.h, [6](#)
- code/include/BIT\_MATH.h, [4](#), [9](#)
- code/include/cuteOS.h, [9](#), [16](#)
- code/include/main.h, [16](#), [18](#)
- code/include/port.h, [19](#), [21](#)
- code/include/STD\_TYPES.h, [21](#), [25](#)
- code/include/traffic.h, [25](#), [30](#)
- code/include/traffic\_cfg.h, [30](#), [32](#)
- code/src/cuteOS.c, [32](#), [39](#)
- code/src/main.c, [41](#), [46](#)
- code/src/traffic.c, [47](#), [50](#)
- code/src/traffic\_cfg.c, [52](#), [53](#)
- CONCAT\_2BITS
  - BIT\_MATH.h, [6](#)
- CONCAT\_3BITS
  - BIT\_MATH.h, [6](#)
- CONCAT\_4BITS
  - BIT\_MATH.h, [6](#)
- CONCAT\_5BITS
  - BIT\_MATH.h, [7](#)
- CONCAT\_6BITS
  - BIT\_MATH.h, [7](#)
- CONCAT\_7BITS
  - BIT\_MATH.h, [7](#)
- CONCAT\_8BITS
  - BIT\_MATH.h, [7](#)
- cuteOS.c
  - cuteOS\_GetTickTime, [35](#)
  - cuteOS\_Init, [35](#)
  - cuteOS\_SetTickTime, [36](#)
  - cuteOS\_Start, [36](#)
  - cuteOS\_TaskCreate, [37](#)
  - cuteOS\_TaskRemove, [37](#)
  - MAX\_TASKS\_NUM, [34](#)
  - MAX\_TICK\_TIME\_MS, [34](#)
  - tasks, [38](#)
- cuteOS.h
  - cuteOS\_GetTickTime, [10](#)
  - cuteOS\_Init, [11](#)
  - cuteOS\_SetCallback, [12](#)
  - cuteOS\_SetTickTime, [13](#)
  - cuteOS\_Start, [13](#)
  - cuteOS\_TaskCreate, [14](#)
  - cuteOS\_TaskRemove, [15](#)
- cuteOS\_GetTickTime
  - cuteOS.c, [35](#)
  - cuteOS.h, [10](#)
- cuteOS\_Init
  - cuteOS.c, [35](#)
  - cuteOS.h, [11](#)
- cuteOS\_SetCallback
  - cuteOS.h, [12](#)
- cuteOS\_SetTickTime
  - cuteOS.c, [36](#)
  - cuteOS.h, [13](#)
- cuteOS\_Start
  - cuteOS.c, [36](#)
  - cuteOS.h, [13](#)
- cuteOS\_TASK\_t, [2](#)
  - callback, [2](#)
  - id, [2](#)
  - ticks, [3](#)
- cuteOS\_TaskCreate
  - cuteOS.c, [37](#)
  - cuteOS.h, [14](#)
- cuteOS\_TaskRemove
  - cuteOS.c, [37](#)
  - cuteOS.h, [15](#)
- ERROR\_BUSY

STD\_TYPES.h, [24](#)  
 ERROR\_ILLEGAL\_PARAM  
     STD\_TYPES.h, [24](#)  
 ERROR\_NO  
     STD\_TYPES.h, [24](#)  
 ERROR\_NOT\_INITIALIZED  
     STD\_TYPES.h, [24](#)  
 ERROR\_NULL\_POINTER  
     STD\_TYPES.h, [24](#)  
 ERROR\_OUT\_OF\_RANGE  
     STD\_TYPES.h, [24](#)  
 ERROR\_t  
     STD\_TYPES.h, [24](#)  
 ERROR\_TIMEOUT  
     STD\_TYPES.h, [24](#)  
 ERROR\_YES  
     STD\_TYPES.h, [24](#)  
  
 f32  
     STD\_TYPES.h, [22](#)  
 f64  
     STD\_TYPES.h, [23](#)  
 FALSE  
     STD\_TYPES.h, [24](#)  
  
 GET\_BIT  
     BIT\_MATH.h, [7](#)  
 GREEN  
     traffic.h, [26](#)  
 green\_duration  
     TRAFFIC\_CONFIGS\_t, [3](#)  
 greenPin  
     port.h, [20](#)  
  
 HIGH  
     STD\_TYPES.h, [24](#)  
  
 id  
     cuteOS\_TASK\_t, [2](#)  
 Init\_Others  
     main.c, [42](#)  
 INTERRUPT\_Timer\_0\_Overflow  
     main.h, [17](#)  
 INTERRUPT\_Timer\_1\_Overflow  
     main.h, [17](#)  
 INTERRUPT\_Timer\_2\_Overflow  
     main.h, [17](#)  
  
 led1\_toggle  
     main.c, [43](#)  
 led1Pin  
     port.h, [20](#)  
 led2\_toggle  
     main.c, [43](#)  
 led2Pin  
     port.h, [20](#)  
 led3\_toggle  
     main.c, [43](#)  
 led3Pin  
     port.h, [20](#)  
  
 LOW  
     STD\_TYPES.h, [24](#)  
  
 main  
     main.c, [44](#)  
 main.c  
     buzzer\_toggle, [42](#)  
     Init\_Others, [42](#)  
     led1\_toggle, [43](#)  
     led2\_toggle, [43](#)  
     led3\_toggle, [43](#)  
     main, [44](#)  
     motor\_toggle, [45](#)  
 main.h  
     INTERRUPT\_Timer\_0\_Overflow, [17](#)  
     INTERRUPT\_Timer\_1\_Overflow, [17](#)  
     INTERRUPT\_Timer\_2\_Overflow, [17](#)  
     OSC\_FREQ, [18](#)  
     OSC\_PER\_INST, [18](#)  
 MAX\_TASKS\_NUM  
     cuteOS.c, [34](#)  
 MAX\_TICK\_TIME\_MS  
     cuteOS.c, [34](#)  
 motor\_toggle  
     main.c, [45](#)  
 motorPin  
     port.h, [20](#)  
  
 NORMAL  
     STD\_TYPES.h, [24](#)  
 NULL  
     STD\_TYPES.h, [22](#)  
 NULL\_BYTE  
     STD\_TYPES.h, [22](#)  
  
 OSC\_FREQ  
     main.h, [18](#)  
 OSC\_PER\_INST  
     main.h, [18](#)  
  
 port.h  
     amberPin, [20](#)  
     buzzerPin, [20](#)  
     greenPin, [20](#)  
     led1Pin, [20](#)  
     led2Pin, [20](#)  
     led3Pin, [20](#)  
     motorPin, [20](#)  
     redPin, [20](#)  
  
 RED  
     traffic.h, [26](#)  
 RED\_AMBER  
     traffic.h, [26](#)  
 red\_amber\_duration  
     TRAFFIC\_CONFIGS\_t, [3](#)  
 red\_duration  
     TRAFFIC\_CONFIGS\_t, [4](#)

- redPin
  - port.h, 20
- s16
  - STD\_TYPES.h, 23
- s32
  - STD\_TYPES.h, 23
- s8
  - STD\_TYPES.h, 23
- SET\_BIT
  - BIT\_MATH.h, 8
- size\_t
  - STD\_TYPES.h, 23
- STATE\_t
  - STD\_TYPES.h, 24
- STD\_TYPES.h
  - ACTIVATION\_STATUS\_t, 23
  - ACTIVE\_HIGH, 24
  - ACTIVE\_LOW, 24
  - BOOL\_t, 24
  - ERROR\_BUSY, 24
  - ERROR\_ILLEGAL\_PARAM, 24
  - ERROR\_NO, 24
  - ERROR\_NOT\_INITIALIZED, 24
  - ERROR\_NULL\_POINTER, 24
  - ERROR\_OUT\_OF\_RANGE, 24
  - ERROR\_t, 24
  - ERROR\_TIMEOUT, 24
  - ERROR\_YES, 24
  - f32, 22
  - f64, 23
  - FALSE, 24
  - HIGH, 24
  - LOW, 24
  - NORMAL, 24
  - NULL, 22
  - NULL\_BYTE, 22
  - s16, 23
  - s32, 23
  - s8, 23
  - size\_t, 23
  - STATE\_t, 24
  - TRUE, 24
  - u16, 23
  - u32, 23
  - u8, 23
- tasks
  - cuteOS.c, 38
- ticks
  - cuteOS\_TASK\_t, 3
- TOG\_BIT
  - BIT\_MATH.h, 8
- traffic.c
  - TRAFFIC\_DeInit, 48
  - TRAFFIC\_GetColor, 48
  - TRAFFIC\_Init, 48
  - TRAFFIC\_SetColor, 49
- traffic.h
  - AMBER, 26
  - GREEN, 26
  - RED, 26
  - RED\_AMBER, 26
  - TRAFFIC\_DeInit, 27
  - TRAFFIC\_GetColor, 27
  - TRAFFIC\_Init, 28
  - TRAFFIC\_SEQUENCE\_t, 26
  - TRAFFIC\_SetColor, 28
  - TRAFFIC\_Update, 29
- traffic\_cfg.c
  - TRAFFIC\_Configs, 53
- traffic\_cfg.h
  - TRAFFIC\_Configs, 32
  - TRAFFIC\_DURATION\_AMBER, 31
  - TRAFFIC\_DURATION\_GREEN, 31
  - TRAFFIC\_DURATION\_RED, 31
  - TRAFFIC\_DURATION\_RED\_AMBER, 31
  - TRAFFIC\_SEQUENCE\_DURATION\_t, 31
- TRAFFIC\_Configs
  - traffic\_cfg.c, 53
  - traffic\_cfg.h, 32
- TRAFFIC\_CONFIGS\_t, 3
  - amber\_duration, 3
  - green\_duration, 3
  - red\_amber\_duration, 3
  - red\_duration, 4
- TRAFFIC\_DeInit
  - traffic.c, 48
  - traffic.h, 27
- TRAFFIC\_DURATION\_AMBER
  - traffic\_cfg.h, 31
- TRAFFIC\_DURATION\_GREEN
  - traffic\_cfg.h, 31
- TRAFFIC\_DURATION\_RED
  - traffic\_cfg.h, 31
- TRAFFIC\_DURATION\_RED\_AMBER
  - traffic\_cfg.h, 31
- TRAFFIC\_GetColor
  - traffic.c, 48
  - traffic.h, 27
- TRAFFIC\_Init
  - traffic.c, 48
  - traffic.h, 28
- TRAFFIC\_SEQUENCE\_DURATION\_t
  - traffic\_cfg.h, 31
- TRAFFIC\_SEQUENCE\_t
  - traffic.h, 26
- TRAFFIC\_SetColor
  - traffic.c, 49
  - traffic.h, 28
- TRAFFIC\_Update
  - traffic.h, 29
- TRUE
  - STD\_TYPES.h, 24
- u16
  - STD\_TYPES.h, 23
- u32

STD\_TYPES.h, [23](#)

u8

STD\_TYPES.h, [23](#)