Cute OS

1.1.0

Generated by Doxygen 1.9.1

1 Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

3 Data Structure Documentation

3.1 cuteOS_TASK_t Struct Reference

Data Fields

- ERROR_t(∗ callback)(void)
- u32_t delay_ms
- u16_t ticks
- u8_t id

3.1.1 Detailed Description

Definition at line 65 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 66 of file cuteOS.c.

3.1.2.2 delay_ms `u32_t delay_ms`

Delay in ms

Definition at line 67 of file cuteOS.c.

3.1.2.3 id `u8_t id`

Task ID

Definition at line 69 of file cuteOS.c.

3.1.2.4 ticks `u16_t ticks`

Number of ticks after which the task will run

Definition at line 68 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:

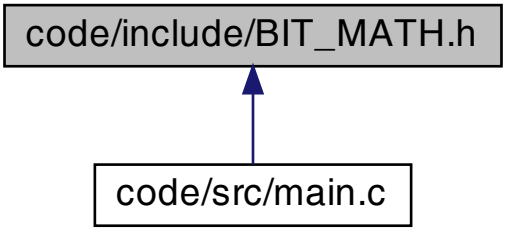
- code/include/traffic_cfg.h

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)

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	*REGISTER*	the register includes the bit
in	*BIT*	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	*REGISTER*	the register includes the bit
in	*BIT*	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	*REGISTER*	the register includes the bit
in	*BIT*	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	*REGISTER*	the register includes the bit
in	*BIT*	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	*REGISTER*	the register includes the bit
in	*BIT*	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	*REGISTER*	is the register includes the bit
in	*BIT*	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

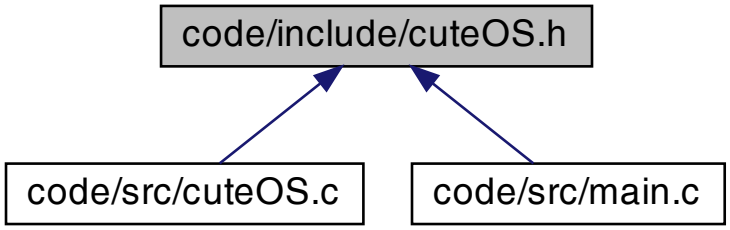
```
00001
00008 #ifndef BIT_MATH_H
00009 #define BIT_MATH_H
00010
00011
00019 #define GET_BIT(REGISTER, BIT)      ( 1 & ( (REGISTER) » (BIT) ) )
00020
00021
00028 #define SET_BIT(REGISTER, BIT)      ( (REGISTER) |= (1 « (BIT)) )
00029
00030
00037 #define CLR_BIT(REGISTER, BIT)      ( (REGISTER) &= ~(1 « (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_Init (void)

  Sets up Timer 2 to drive the simple EOS.
- ERROR_t cuteOS_TaskCreate (ERROR_t(∗const task_ptr)(void), const u16_t TICK_TIME_MS)

  Create a task with the given task function and the given tick time.
- ERROR_t cuteOS_TaskRemove (ERROR_t(∗const task_ptr)(void))

  Remove a task from the tasks array.
- void cuteOS_Start (void)

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

4.3.1 Detailed Description

Simple EOS interfaces header file. See cuteOS.c for more details.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.1.0

Date

2022-03-22

Copyright

Copyright (c) 2022

Definition in file cuteOS.h.

4.3.2 Function Documentation

4.3.2.1 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)

< increments = (Number of mSec) ∗ (Number of Instructions per mSec)

< Number of mSec = tick_time_ms

< 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

< Load T2 and reload capt. reg. high bytes

< Load T2 and reload capt. reg. low bytes

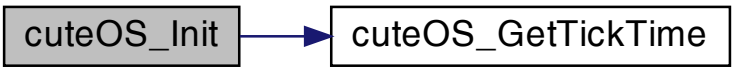
< Enable Timer 2 interrupt

< Start Timer 2

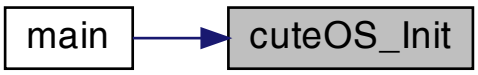
< Globally enable interrupts

Definition at line 179 of file cuteOS.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.2.2 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.

Go to idle mode for some time = tickTimeInMs by disabling all interrupts and setting the sleep mode to Idle.

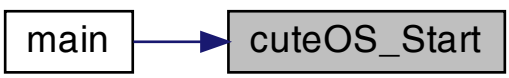
Note

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

< Enter idle mode to save power

Definition at line 149 of file cuteOS.c.

Here is the caller graph for this function:



4.3.2.3 cuteOS_TaskCreate()

```
ERROR_t cuteOS_TaskCreate (
        ERROR_t(*)(void) callback,
        const u16_t TICK_TIME_MS )
```

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

Parameters

in		

Definition at line 91 of file cuteOS.c.

Here is the caller graph for this function:

4.3.2.4 cuteOS_TaskRemove()

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

Remove a task from the tasks array.

Parameters

in		

< Find the task in the task array

< Task found

< Rearrange the tasks array

Definition at line 118 of file cuteOS.c.

Here is the caller graph for this function:

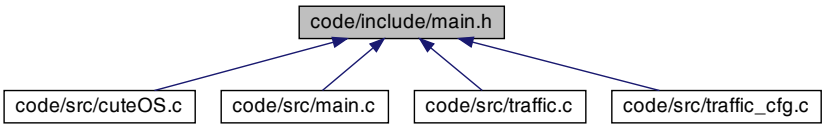
4.4 cuteOS.h

```
00001
00009 #ifndef CUTE_OS_H
00010 #define CUTE_OS_H
00011
00012
00015 ERROR_t cuteOS_Init(void);
00016
00017
00027 ERROR_t cuteOS_TaskCreate(ERROR_t (* const task_ptr)(void), const u16_t TICK_TIME_MS);
00028
00029
00037 ERROR_t cuteOS_TaskRemove(ERROR_t (* const task_ptr)(void));
00038
00039
00040
00044 void cuteOS_Start(void);
00045
00046
```

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)

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.

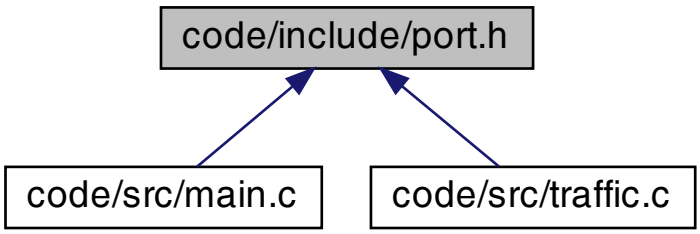
4.6 main.h

```
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)

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.

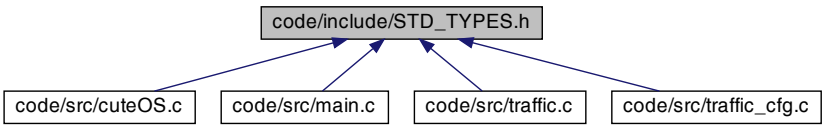
4.8 port.h

```
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 8051 Microcontrollers.

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



Macros

- #define LOW ((STATE_t)0)
- #define HIGH ((STATE_t)1)
- #define NORMAL ((STATE_t)2)
- #define ACTIVE_LOW ((ACTIVATION_STATUS_t)0)
- #define ACTIVE_HIGH ((ACTIVATION_STATUS_t)1)
- #define FALSE ((BOOL_t)0)
- #define TRUE ((BOOL_t)1)
- #define ERROR_NO ((ERROR_t)0)
- #define ERROR_YES ((ERROR_t)0x1)
- #define ERROR_TIMEOUT ((ERROR_t)0x2)
- #define ERROR_NULL_POINTER ((ERROR_t)0x4)
- #define ERROR_BUSY ((ERROR_t)0x8)
- #define ERROR_NOT_INITIALIZED ((ERROR_t)0x10)
- #define ERROR_ILLEGAL_PARAM ((ERROR_t)0x20)
- #define ERROR_OUT_OF_RANGE ((ERROR_t)0x40)
- #define NULL ((void *)0)
- #define NULL_BYTE ('\0')

Typedefs

- typedef signed long int s32_t
- typedef signed short int s16_t
- typedef signed char s8_t
- typedef unsigned long int u32_t
- typedef unsigned short int u16_t
- typedef unsigned char u8_t
- typedef float f32
- typedef double f64
- typedef u16_t size_t
- typedef u8_t STATE_t
- typedef u8_t ACTIVATION_STATUS_t
- typedef u8_t BOOL_t
- typedef u8_t ERROR_t

4.9.1 Detailed Description

Standard data types For 8051 Microcontrollers.

Author

Mahmoud Karam (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 ACTIVE_HIGH `#define ACTIVE_HIGH ((ACTIVATION_STATUS_t)1)`

The pin is low when it is pulled high

Definition at line 38 of file STD_TYPES.h.

4.9.2.2 ACTIVE_LOW `#define ACTIVE_LOW ((ACTIVATION_STATUS_t)0)`

The pin is high when it is pulled low

Definition at line 37 of file STD_TYPES.h.

4.9.2.3 ERROR_BUSY `#define ERROR_BUSY ( (ERROR_t)0x8 )`

Busy state occured

Definition at line 50 of file STD_TYPES.h.

4.9.2.4 ERROR_ILLEGAL_PARAM `#define ERROR_ILLEGAL_PARAM ( (ERROR_t)0x20 )`

Invalid input state occured

Definition at line 52 of file STD_TYPES.h.

4.9.2.5 ERROR_NO `#define ERROR_NO ( (ERROR_t)0 )`

No error occured

Definition at line 46 of file STD_TYPES.h.

4.9.2.6 ERROR_NOT_INITIALIZED `#define ERROR_NOT_INITIALIZED ( (ERROR_t)0x10 )`

Not initialized state occured

Definition at line 51 of file STD_TYPES.h.

4.9.2.7 ERROR_NULL_POINTER `#define ERROR_NULL_POINTER ( (ERROR_t)0x4 )`

Null pointer occured

Definition at line 49 of file STD_TYPES.h.

4.9.2.8 ERROR_OUT_OF_RANGE `#define ERROR_OUT_OF_RANGE ( (ERROR_t)0x40 )`

Out of range state occured

Definition at line 53 of file STD_TYPES.h.

4.9.2.9 ERROR_TIMEOUT `#define ERROR_TIMEOUT ( (ERROR_t)0x2 )`

Timeout occured

Definition at line 48 of file STD_TYPES.h.

4.9.2.10 ERROR_YES `#define ERROR_YES ( (ERROR_t)0x1 )`

Error occured

Definition at line 47 of file STD_TYPES.h.

4.9.2.11 FALSE `#define FALSE ((BOOL_t)0)`

Definition at line 42 of file STD_TYPES.h.

4.9.2.12 HIGH `#define HIGH ((STATE_t)1)`

Definition at line 33 of file STD_TYPES.h.

4.9.2.13 LOW `#define LOW ((STATE_t)0)`

Definition at line 32 of file STD_TYPES.h.

4.9.2.14 NORMAL `#define NORMAL ((STATE_t)2)`

Used for any normal state

Definition at line 34 of file STD_TYPES.h.

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

NULL pointer

Definition at line 57 of file STD_TYPES.h.

4.9.2.16 NULL_BYTE `#define NULL_BYTE ('\0')`

Definition at line 60 of file STD_TYPES.h.

4.9.2.17 TRUE `#define TRUE ((BOOL_t)1)`

Definition at line 43 of file STD_TYPES.h.

4.9.3 Typedef Documentation

4.9.3.1 ACTIVATION_STATUS_t `typedef u8_t ACTIVATION_STATUS_t`

Definition at line 36 of file STD_TYPES.h.

4.9.3.2 BOOL_t `typedef u8_t BOOL_t`

Definition at line 41 of file STD_TYPES.h.

4.9.3.3 ERROR_t `typedef u8_t ERROR_t`

Definition at line 45 of file STD_TYPES.h.

4.9.3.4 f32 `typedef float f32`

Definition at line 22 of file STD_TYPES.h.

4.9.3.5 f64 `typedef double f64`

Definition at line 23 of file STD_TYPES.h.

4.9.3.6 s16_t `typedef signed short int s16_t`

Definition at line 13 of file STD_TYPES.h.

4.9.3.7 s32_t `typedef signed long int s32_t`

Definition at line 12 of file STD_TYPES.h.

4.9.3.8 s8_t `typedef signed char s8_t`

Definition at line 14 of file STD_TYPES.h.

4.9.3.9 size_t `typedef u16_t size_t`

< This macro is defined in <stddef.h> for the size_t type

Definition at line 27 of file STD_TYPES.h.

4.9.3.10 STATE_t `typedef u8_t STATE_t`

Definition at line 31 of file STD_TYPES.h.

4.9.3.11 u16_t `typedef unsigned short int u16_t`

Definition at line 18 of file STD_TYPES.h.

4.9.3.12 u32_t typedef unsigned long int u32_t

Definition at line 17 of file STD_TYPES.h.

4.9.3.13 u8_t typedef unsigned char u8_t

Definition at line 19 of file STD_TYPES.h.

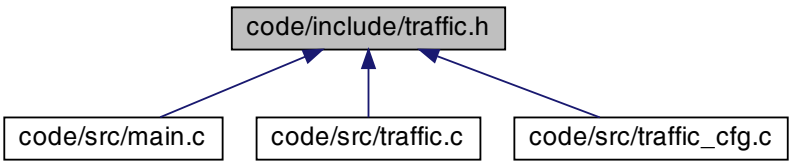
4.10 STD_TYPES.h

```
00001
00008 #ifndef STD_TYPES_H
00009 #define STD_TYPES_H
00010
00011 /* Signed integers  */
00012 typedef     signed long int             s32_t;
00013 typedef     signed short int            s16_t;
00014 typedef     signed char                 s8_t;
00015
00016 /* Unsigned integers    */
00017 typedef     unsigned long int           u32_t;
00018 typedef     unsigned short int          u16_t;
00019 typedef     unsigned char               u8_t;
00020
00021 /* Float numbers    */
00022 typedef     float                       f32;
00023 typedef     double                      f64;
00024
00025 /* Special types    */
00026  #undef __SIZE_TYPE__
00027 typedef     u16_t   size_t;
00028
00029 #undef HIGH
00030 #undef LOW
00031 typedef     u8_t      STATE_t;
00032 #define     LOW         ((STATE_t)0)
00033 #define     HIGH        ((STATE_t)1)
00034 #define     NORMAL      ((STATE_t)2)
00036 typedef     u8_t      ACTIVATION_STATUS_t;
00037 #define     ACTIVE_LOW              ((ACTIVATION_STATUS_t)0)
00038 #define     ACTIVE_HIGH             ((ACTIVATION_STATUS_t)1)
00040 /* Boolean type */
00041 typedef     u8_t      BOOL_t;
00042 #define     FALSE       ((BOOL_t)0)
00043 #define     TRUE        ((BOOL_t)1)
00044
00045 typedef     u8_t      ERROR_t;
00046 #define     ERROR_NO                ( (ERROR_t)0     )
00047 #define     ERROR_YES               ( (ERROR_t)0x1   )
00048 #define     ERROR_TIMEOUT           ( (ERROR_t)0x2   )
00049 #define     ERROR_NULL_POINTER      ( (ERROR_t)0x4   )
00050 #define     ERROR_BUSY              ( (ERROR_t)0x8   )
00051 #define     ERROR_NOT_INITIALIZED   ( (ERROR_t)0x10  )
00052 #define     ERROR_ILLEGAL_PARAM     ( (ERROR_t)0x20  )
00053 #define     ERROR_OUT_OF_RANGE      ( (ERROR_t)0x40  )
00055 /* Pointers */
00056 #undef NULL
00057 #define NULL ((void *)0)
00059 #undef NULL_BYTE
00060 #define NULL_BYTE  ('\0')
00061
00062  #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)

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

RED	
RED_AMBER	
GREEN	
AMBER	

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_t`: 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 142 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	*Copy_color*	pointer to the variable to store the color sequence. Expected values: • RED • RED_AMBER • GREEN • AMBER Those colors are members of the global enumeration TRAFFIC_SEQUENCE_t.

Returns

ERROR_t : Check the options in the global enum ERROR_t.

Definition at line 165 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_t : Check the options in the global enum ERROR_t.

< Reset the time counter

< Initialize the colorSequence

Definition at line 122 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: • RED • RED_AMBER • GREEN • AMBER Those colors are members of the global enumeration TRAFFIC_SEQUENCE_t.

Returns

ERROR_t : Check the options in the global enum ERROR_t.

Definition at line 156 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 189 of file traffic.c.

Here is the caller graph for this function:

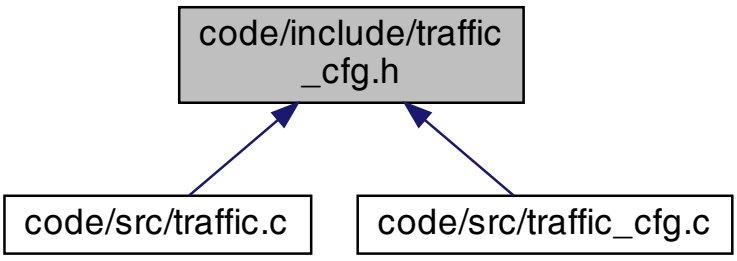
4.12 traffic.h

```
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 = 4 , TRAFFIC_DURATION_RED_AMBER = 2 , TRAFFIC_DURATION_GREEN = 4 , 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)

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

TRAFFIC_DURATION_RED	Red light duration in seconds
TRAFFIC_DURATION_RED_AMBER	Red-Amber light duration in seconds
TRAFFIC_DURATION_GREEN	Green light duration in seconds
TRAFFIC_DURATION_AMBER	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

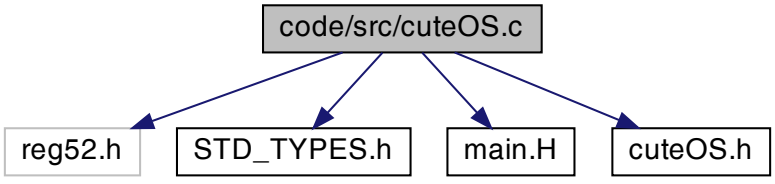
```
00001
00009 #ifndef TRAFFIC_CFG_H
00010 #define TRAFFIC_CFG_H
00011
00012 /*------------------------------------------------------------------------*/
00013 /*                YOU CAN CHANGE THE FOLLOWING PARAMETERS                 */
00014 /*------------------------------------------------------------------------*/
00015 typedef enum {
00016     TRAFFIC_DURATION_RED = 4,
00017     TRAFFIC_DURATION_RED_AMBER = 2,
00018     TRAFFIC_DURATION_GREEN = 4,
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 8

Functions

- ERROR_t cuteOS_TaskCreate (ERROR_t(∗const callback)(void), const u16_t TICK_TIME_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)

  The OS enters 'idle mode' between clock ticks to save power.
- ERROR_t cuteOS_GetTickTime (u8_t ∗const ptr_tick_time_ms)
- 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)

cuteOS schedules the tasks in a cooperative manner. It invokes te 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

1.1.0

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 8`

Number of tasks created by the user.

Definition at line 61 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).

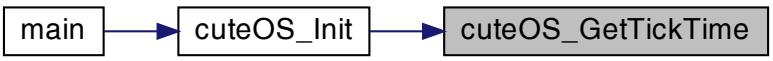
Definition at line 53 of file cuteOS.c.

4.15.3 Function Documentation

4.15.3.1 cuteOS_GetTickTime() ERROR_t cuteOS_GetTickTime (u8_t *const *ptr_tick_time_ms*)

Definition at line 156 of file cuteOS.c.

Here is the caller graph for this function:



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)

< increments = (Number of mSec) * (Number of Instructions per mSec)

< Number of mSec = tick_time_ms

< 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

< Load T2 and reload capt. reg. high bytes

< Load T2 and reload capt. reg. low bytes

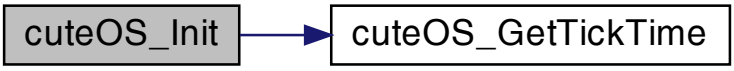
< Enable Timer 2 interrupt

< Start Timer 2

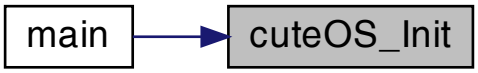
< Globally enable interrupts

Definition at line 179 of file cuteOS.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.15.3.3 cuteOS_Start() `void cuteOS_Start ( void )`

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

Go to idle mode for some time = tickTimeInMs by disabling all interrupts and setting the sleep mode to Idle.

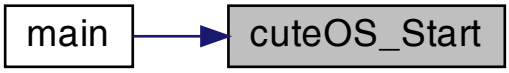
Note

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

< Enter idle mode to save power

Definition at line 149 of file cuteOS.c.

Here is the caller graph for this function:



4.15.3.4 cuteOS_TaskCreate()

```
ERROR_t cuteOS_TaskCreate (
        ERROR_t(*)(void) callback,
        const u16_t TICK_TIME_MS )
```

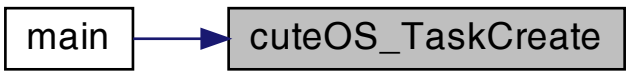
Create a task with the given task function and the given tick time.

This function does the following:

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

Definition at line 91 of file cuteOS.c.

Here is the caller graph for this function:



4.15.3.5 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

in	*callback*	Pointer to the task function.

Returns

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

< Find the task in the task array

< Task found

< Rearrange the tasks array

Definition at line 118 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 73 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 /* PRIVATE FUNCTIONS DECLARATION                                     */
00030 /*-------------------------------------------------------------------*/
00031 static ERROR_t cuteOS_SetTickTime(const u8_t TICK_TIME_MS);
00032 static ERROR_t cuteOS_GCD(u32_t *gcd);
00033 static ERROR_t cuteOS_UpdateTicks(void);
00034 static void cuteOS_Sleep(void);
00035 static void cuteOS_ISR();
00036
00037
00038 /*-------------------------------------------------------------------*/
00039 /*                          PRIVATE DATA                             */
00040 /*-------------------------------------------------------------------*/
00041
00051 #define MAX_TICK_TIME_MS  65
00052
00054 static u8_t cuteOS_tick_time_ms = 0;
00055
00057 static u16_t cuteOS_tick_count = 0;
00058
00059 #define MAX_TASKS_NUM   8
00060
00062 static u8_t cuteOS_task_counter = 0;
00065 typedef struct {
00066     ERROR_t (*callback)(void);
00067     u32_t delay_ms;
00068     u16_t ticks;
00069     u8_t  id;
00070 }cuteOS_TASK_t;
00071
00073 cuteOS_TASK_t tasks[MAX_TASKS_NUM] = {0};
00074
00075
00076
00077
00078
00079
00080
00081 /*-------------------------------------------------------------------*/
00082 /*                          PUBLIC FUNCTIONS                         */
00083 /*-------------------------------------------------------------------*/
00084
00091 ERROR_t cuteOS_TaskCreate(ERROR_t (* const callback)(void), const u16_t TICK_TIME_MS) {
00092     ERROR_t error = ERROR_NO;
00093
00094     if(cuteOS_task_counter < MAX_TASKS_NUM) {
00095         ++cuteOS_task_counter;
00096         tasks[cuteOS_task_counter - 1].id = cuteOS_task_counter - 1;
00097         tasks[cuteOS_task_counter - 1].delay_ms = TICK_TIME_MS;
00098         tasks[cuteOS_task_counter - 1].callback = callback;
00099
00100         // error |= cuteOS_UpdateTicks();
00101     } else {
00102         error |= ERROR_OUT_OF_RANGE;
00103     }
00104
00105     return error;
00106 }
00107
00108
00118 ERROR_t cuteOS_TaskRemove(ERROR_t (* const callback)(void)) {
00119     ERROR_t error = ERROR_YES;
00120     u8_t i;
00121
00123     for(i = 0; i < cuteOS_task_counter; ++i) {
00124         if(tasks[i].callback == callback) {
00125             error |= ERROR_NO;
00128             for(; i < cuteOS_task_counter - 1; ++i) {
00129                 tasks[i] = tasks[i + 1];
00130             }
00131
00132             --cuteOS_task_counter;
00133             tasks[cuteOS_task_counter].callback = NULL;
00134
00135             error |= cuteOS_UpdateTicks();
00136
00137             break;
00138         }
00139     }
```

```
00140
00141     return error;
00142 }
00143
00144
00149 void cuteOS_Start(void) {
00150     cuteOS_UpdateTicks();
00151     while(1) {
00152         PCON |= 0x01;
00153     }
00154 }
00155
00156 ERROR_t cuteOS_GetTickTime(u8_t * const ptr_tick_time_ms){
00157     ERROR_t error = ERROR_NO;
00158
00159     if(ptr_tick_time_ms != NULL) {
00160         if(0 == cuteOS_tick_time_ms) {
00161             cuteOS_tick_time_ms = MAX_TICK_TIME_MS;
00162         }
00163
00164         *ptr_tick_time_ms = cuteOS_tick_time_ms;
00165     } else {
00166         error |= ERROR_NULL_POINTER;
00167     }
00168
00169     return error;
00170 }
00171
00172
00179 ERROR_t cuteOS_Init(void) {
00180     ERROR_t error = ERROR_NO;
00181     u16_t increments, reload_16;
00182     u8_t tick_time_ms;
00183
00184     TR2 = 0;
00186
00190     T2CON = 0x04;
00198     error |= cuteOS_GetTickTime(&tick_time_ms);
00199     increments = (u16_t) ( ((u32_t)tick_time_ms * (OSC_FREQ/1000)) / (u32_t)OSC_PER_INST );
00200
00201     reload_16 = (u16_t)(65536UL - increments);
00202     RCAP2H = TH2 = (u8_t)(reload_16 / 256);
00203     RCAP2L = TL2 = (u8_t)(reload_16 % 256);
00205     ET2 = 1;
00206     TR2 = 1;
00207     EA = 1;
00209     return error;
00210 }
00211
00212
00213
00214
00215 /*---------------------------------------------------------------------*/
00216 /*  PRIVATE FUNCTIONS DEFINITIONS                                      */
00217 /*---------------------------------------------------------------------*/
00218
00222 static void cuteOS_ISR() interrupt INTERRUPT_Timer_2_Overflow {
00223     u8_t i;
00224
00226     TF2 = 0;
00227
00229     ++cuteOS_tick_count;
00230
00232     for(i = 0; i < cuteOS_task_counter; ++i) {
00233         if( (cuteOS_tick_count % tasks[i].ticks) == 0) {
00234             if(tasks[i].callback != NULL) {
00235                 tasks[i].callback();
00236             }
00237         }
00238     }
00239
00241     // cuteOS_tick_count = 0;
00242 }
00243
00244 static ERROR_t cuteOS_UpdateTicks(void) {
00245     ERROR_t error = ERROR_NO;
00246     u32_t gcd_delay_ms;
00247     u8_t i;
00248
00250     error |= cuteOS_GCD(&gcd_delay_ms);
00251
00252     error |= cuteOS_SetTickTime(gcd_delay_ms);
00253
00255     for(i = 0; i < cuteOS_task_counter; ++i) {
00256         tasks[i].ticks = tasks[i].delay_ms / gcd_delay_ms;
00257     }
00258
```

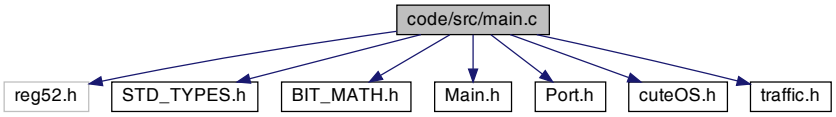
```
00259     return error;
00260 }
00261
00262 static ERROR_t cuteOS_GCD(u32_t *gcd) {
00263     ERROR_t error = ERROR_NO;
00264     u32_t remainder;
00265     u32_t y;
00266     u8_t i;
00267
00268     *gcd = tasks[0].delay_ms;
00269     for(i = 1; i < cuteOS_task_counter; ++i) {
00270         y = tasks[i].delay_ms;
00271         while(y != 0) {
00272             remainder = *gcd % y;
00273             *gcd = y;
00274             y = remainder;
00275         }
00276     }
00277
00279     i = 2;
00280     while(*gcd > MAX_TICK_TIME_MS) {
00281         while( (*gcd % i) != 0) {
00282             ++i;
00283         }
00284         *gcd /= i;
00285     }
00286
00287     return error;
00288 }
00289
00290
00295 static ERROR_t cuteOS_SetTickTime(const u8_t TICK_TIME_MS){
00296     ERROR_t error = ERROR_NO;
00297
00298     if(TICK_TIME_MS <= MAX_TICK_TIME_MS) {
00299         if(TICK_TIME_MS <= 1) {
00300             cuteOS_tick_time_ms = 1;
00301         } else {
00302             cuteOS_tick_time_ms = TICK_TIME_MS;
00303         }
00304         error |= cuteOS_Init();
00305     } else {
00306         error |= ERROR_OUT_OF_RANGE;
00307     }
00308
00309     return error;
00310 }
```

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)

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();
```

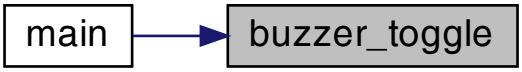
Definition in file main.c.

4.17.2 Function Documentation

4.17.2.1 buzzer_toggle() `void buzzer_toggle ( void )`

Definition at line 63 of file main.c.

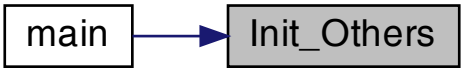
Here is the caller graph for this function:



4.17.2.2 Init_Others() `void Init_Others ( void )`

Definition at line 67 of file main.c.

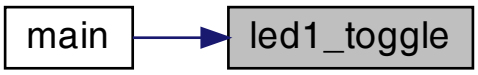
Here is the caller graph for this function:



4.17.2.3 led1_toggle() `void led1_toggle ( void )`

Definition at line 47 of file main.c.

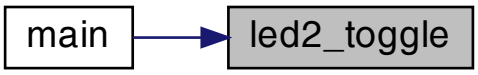
Here is the caller graph for this function:



4.17.2.4 led2_toggle() `void led2_toggle ( void )`

Definition at line 51 of file main.c.

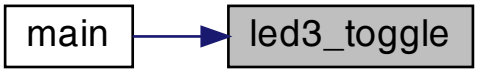
Here is the caller graph for this function:



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

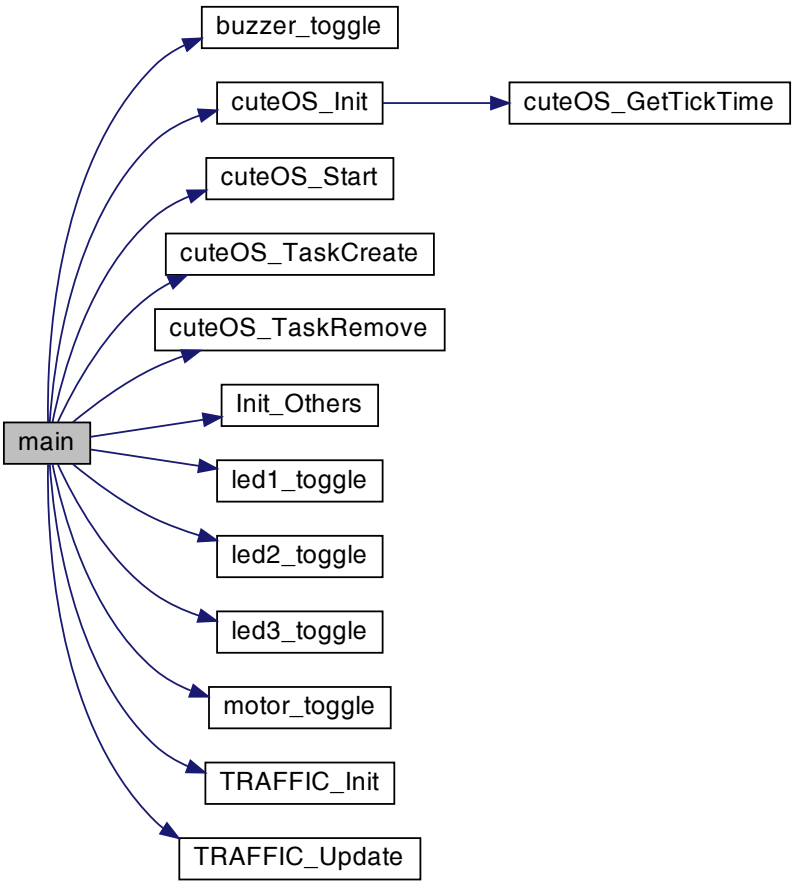
< Initialize the traffic light system

< 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

Definition at line 83 of file main.c.

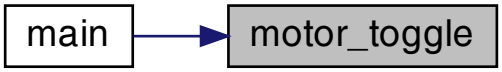
Here is the call graph for this function:



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);
```

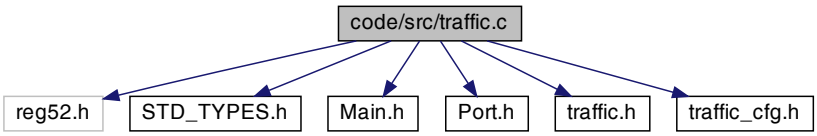
```
00093     cuteOS_TaskCreate(led3_toggle   , 4000);
00094     cuteOS_TaskCreate(buzzer_toggle   , 2000);
00095     cuteOS_TaskCreate(motor_toggle , 5000);
00097     cuteOS_TaskRemove(buzzer_toggle);
00099     cuteOS_Start();
00100 }
```

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)

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 142 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	*Copy_color*	pointer to the variable to store the color sequence. Expected values: • RED • RED_AMBER • GREEN • AMBER Those colors are members of the global enumeration TRAFFIC_SEQUENCE_t.

Returns

ERROR_t : Check the options in the global enum ERROR_t.

Definition at line 165 of file traffic.c.

4.19.2.3 TRAFFIC_Init() `ERROR_t TRAFFIC_Init ( void )`

Initialize the traffic light system to RED state.

Returns

ERROR_t : Check the options in the global enum ERROR_t.

< Reset the time counter

< Initialize the colorSequence

Definition at line 122 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: • RED • RED_AMBER • GREEN • AMBER Those colors are members of the global enumeration TRAFFIC_SEQUENCE_t.

Returns

ERROR_t : Check the options in the global enum ERROR_t.

Definition at line 156 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_t                  timeInState = 0;
00028 /*-------------------------------------------------------------------*/
00029 /*                    PRIVATE FUNCTIONS PROTOTYPES                   */
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 ERROR_t TRAFFIC_Init(void) {
00123     ERROR_t error = ERROR_NO;
00124
00126     timeInState = 0;
00127
00129     colorSequence = RED;
00130     redPin    = HIGH;
00131     amberPin = LOW;
00132     greenPin = LOW;
00133
```

```
00134     return error;
00135 }
00136
00137
00142 ERROR_t TRAFFIC_DeInit(void) {
00143     ERROR_t error = ERROR_NO;
00144
00146     redPin   = LOW;
00147     amberPin = LOW;
00148     greenPin = LOW;
00149
00151     //cuteOS_(NULL);
00152
00153     return error;
00154 }
00155
00156 ERROR_t TRAFFIC_SetColor(const TRAFFIC_SEQUENCE_t Copy_color) {
00157     ERROR_t error = ERROR_NO;
00158     colorSequence = Copy_color;
00159
00160     error |= TRAFFIC_Update();
00161
00162     return error;
00163 }
00164
00165 ERROR_t TRAFFIC_GetColor(TRAFFIC_SEQUENCE_t * const Copy_color) {
00166     ERROR_t error = ERROR_NO;
00167
00168     *Copy_color = colorSequence;
00169
00170     return error;
00171 }
00172
00173
00174
00175
00176
00177
00178 /*-------------------------------------------------------------------------*/
00179 /*                 PRIVATE FUNCTIONS DEFINITIONS                           */
00180 /*-------------------------------------------------------------------------*/
00181
00189 ERROR_t TRAFFIC_Update(void) {
00190     ERROR_t error = ERROR_NO;
00191
00193     switch(colorSequence) {
00194         case RED:;
00195             error |= TRAFFIC_RedSequence();
00196             break;
00197         case RED_AMBER:;
00198             error |= TRAFFIC_RedAmberSequence();
00199             break;
00200         case GREEN:;
00201             error |= TRAFFIC_GreenSequence();
00202             break;
00203         case AMBER:;
00204             error |= TRAFFIC_AmberSequence();
00205             break;
00206         default:;
00207             error |= ERROR_ILLEGAL_PARAM;
00208             break;
00209     }
00210
00211     return error;
00212 }
00213
00214 static ERROR_t TRAFFIC_RedSequence(void) {
00215     ERROR_t error = ERROR_NO;
00216
00217     error |= TRAFFIC_GenericSequence(HIGH, LOW, LOW, TRAFFIC_Configs.red_duration);
00218
00219     return error;
00220 }
00221
00222 static ERROR_t TRAFFIC_RedAmberSequence(void) {
00223     ERROR_t error = ERROR_NO;
00224
00225     error |= TRAFFIC_GenericSequence(HIGH, HIGH, LOW, TRAFFIC_Configs.red_amber_duration);
00226
00227     return error;
00228 }
00229
00230 static ERROR_t TRAFFIC_GreenSequence(void) {
00231     ERROR_t error = ERROR_NO;
00232
00233     error |= TRAFFIC_GenericSequence(LOW, LOW, HIGH, TRAFFIC_Configs.green_duration);
00234
```

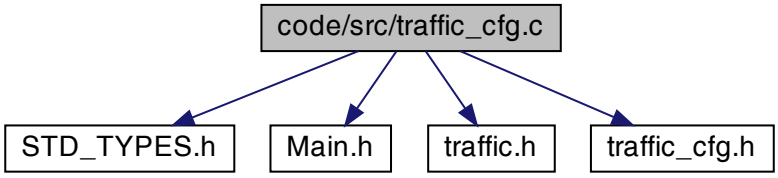
```
00235     return error;
00236 }
00237
00238 static ERROR_t TRAFFIC_AmberSequence(void) {
00239     ERROR_t error = ERROR_NO;
00240
00241     error |= TRAFFIC_GenericSequence(LOW, HIGH, LOW, TRAFFIC_Configs.amber_duration);
00242
00243     return error;
00244 }
00245
00246
00254 static ERROR_t TRAFFIC_GenericSequence(const STATE_t redState, const STATE_t amberState, const STATE_t
       greenState, TRAFFIC_SEQUENCE_DURATION_t duration) {
00255     ERROR_t error = ERROR_NO;
00256     u8_t tickTime = 0;
00257
00259     if(++timeInState >= duration) {
00260         timeInState = 0;
00261         switch(colorSequence) {
00262             case RED:
00263                 colorSequence = RED_AMBER;
00264                 redPin   = HIGH;
00265                 amberPin = HIGH;
00266                 greenPin = LOW;
00267                 break;
00268             case RED_AMBER:
00269                 colorSequence = GREEN;
00270                 redPin   = LOW;
00271                 amberPin = LOW;
00272                 greenPin = HIGH;
00273                 break;
00274             case GREEN:
00275                 colorSequence = AMBER;
00276                 redPin   = LOW;
00277                 amberPin = HIGH;
00278                 greenPin = LOW;
00279                 break;
00280             case AMBER:
00281                 colorSequence = RED;
00282                 redPin   = HIGH;
00283                 amberPin = LOW;
00284                 greenPin = LOW;
00285                 break;
00286             default:
00287                 error |= ERROR_ILLEGAL_PARAM;
00288                 break;
00289         }
00290     } else {
00291         redPin   = redState;
00292         amberPin = amberState;
00293         greenPin = greenState;
00294     }
00295
00296     return error;
00297 }
```

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)

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