Cute OS

1.0.0

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)
- 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:

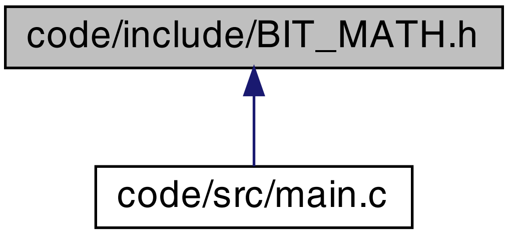
- code/include/traffic_cfg.h

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)

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.

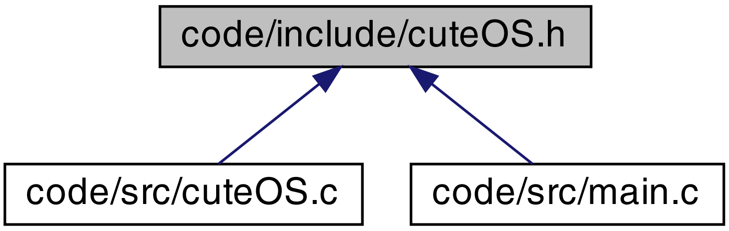
4.2 BIT_MATH.h

```
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_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`)

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

TICK_MS	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

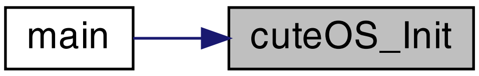
< Enable Timer 2 interrupt

< 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

taskPtr	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

TICK_MS	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

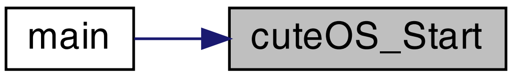
The next clock tick will return the processor to the normal operating state.

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 u16 TICK_MS )
```

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

Parameters

in	*taskPtr*	Pointer to the task function.
in	*TICK_MS*	the frequency of task execution in milliseconds.

Returns

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

Example

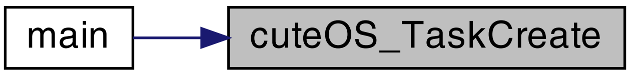
```
cuteOS_TaskCreate(task1, 1000); // task1 will run every 1 second
cuteOS_TaskCreate(task2, 2000); // task2 will run every 2 seconds
```

This function does the following:

- Increment the task counter.
- Set the task ID.
- Set the pointer to the task function.
- Set the number of schedular 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	*taskPtr*	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	*callback*	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:

main
cuteOS_TaskRemove

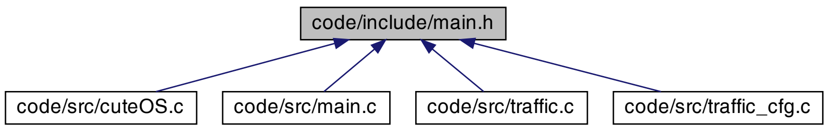
4.4 cuteOS.h

```

#ifndef CUTE_OS_H
#define CUTE_OS_H


ERROR_t cuteOS_SetCallback( ERROR_t (* const taskPtr)(void) );


ERROR_t cuteOS_Init(void);


ERROR_t cuteOS_TaskCreate(ERROR_t (* const taskPtr)(void), const u16 TICK_MS);


ERROR_t cuteOS_TaskRemove(ERROR_t (* const taskPtr)(void));



void cuteOS_Start(void);


ERROR_t cuteOS_SetTickTime(const u8 TICK_MS);


ERROR_t cuteOS_GetTickTime(u8 * const TICK_MS);

#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`)

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.

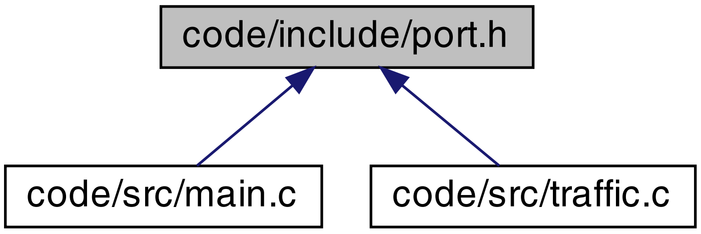
4.6 main.h

```
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.

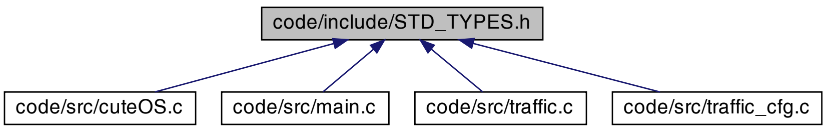
4.8 port.h

```
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 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`)

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 occured
ERROR_YES	Error occured
ERROR_TIMEOUT	Timeout occured
ERROR_NULL_POINTER	Null pointer occured
ERROR_BUSY	Busy state occured
ERROR_NOT_INITIALIZED	Not initialized state occured
ERROR_ILLEGAL_PARAM	Invalid input state occured
ERROR_OUT_OF_RANGE	Out of range state occured

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.

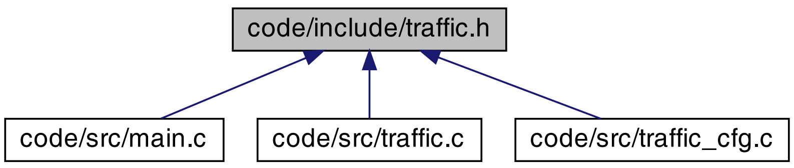
4.10 STD_TYPES.h

```

#ifndef STD_TYPES_H
#define STD_TYPES_H

/* Signed integers  */
typedef     signed long int             s32;
typedef     signed short int            s16;
typedef     signed char                 s8;

/* Unsigned integers    */
typedef     unsigned long int           u32;
typedef     unsigned short int          u16;
typedef     unsigned char               u8;

/* Float numbers    */
typedef     float                       f32;
typedef     double                      f64;

/* Special types    */
 #undef __SIZE_TYPE__
typedef     u16   size_t;
#undef HIGH
#undef LOW
typedef enum{
  LOW,
  HIGH,
  NORMAL,              /* Used for any normal state */
}STATE_t;

typedef enum{
    ACTIVE_LOW,
    ACTIVE_HIGH,
}ACTIVATION_STATUS_t;

/* Boolean type */
typedef enum{
    FALSE,
    TRUE
}BOOL_t;

typedef enum{
    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,
}ERROR_t;

/* Pointers */
#undef NULL
#define NULL ((void *)0)
#undef NULL_BYTE
#define NULL_BYTE  ('\0')

 #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 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	*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 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:

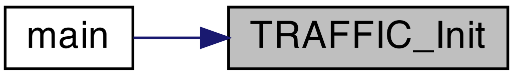
- Initialize the traffic light system to RED sequence.
- Set the callback function of the OS delay to the function TRAFFIC_Update.

< 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: • RED • RED_AMBER • GREEN • AMBER Those colors are members of the global enumeration TRAFFIC_SEQUENCE_t.

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:

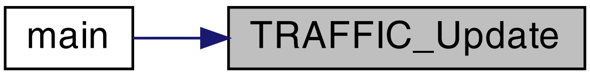
- Setting the traffic light color sequence according to the current color sequence.
- Update the OS delay for the current color sequence.
- Update the color sequence value to the next color sequence. So, when calling this function again, the color sequence will be changed.

< 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:



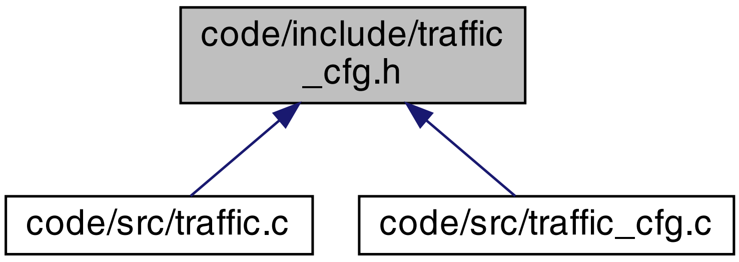
4.12 traffic.h

```
#ifndef TRAFFIC_H
#define TRAFFIC_H

/*---------------------------------------------------------------------*/
/*                           TYPE DEFINITIONS                          */
/*---------------------------------------------------------------------*/
typedef enum {
    RED,
    RED_AMBER,
    GREEN,
    AMBER
}TRAFFIC_SEQUENCE_t;




/*---------------------------------------------------------------------*/
/*                           API FUNCTIONS                             */
/*---------------------------------------------------------------------*/


ERROR_t TRAFFIC_Init(void);


ERROR_t TRAFFIC_DeInit(void);

ERROR_t TRAFFIC_Update(void);


ERROR_t TRAFFIC_SetColor(const TRAFFIC_SEQUENCE_t Copy_color);


ERROR_t TRAFFIC_GetColor(TRAFFIC_SEQUENCE_t * const Copy_color);


#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`)

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

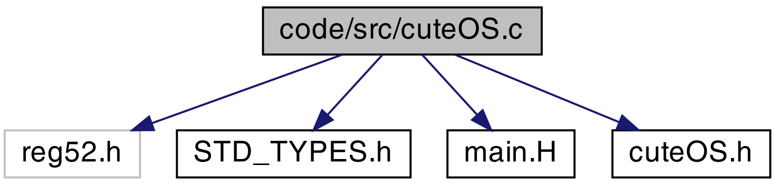
```
#ifndef TRAFFIC_CFG_H
#define TRAFFIC_CFG_H

/*-------------------------------------------------------------------------*/
/*                 YOU CAN CHANGE THE FOLLOWING PARAMETERS                 */
/*-------------------------------------------------------------------------*/
typedef enum {
    TRAFFIC_DURATION_RED = 10,
    TRAFFIC_DURATION_RED_AMBER = 2,
    TRAFFIC_DURATION_GREEN = 10,
    TRAFFIC_DURATION_AMBER = 2,
}TRAFFIC_SEQUENCE_DURATION_t;





/*-------------------------------------------------------------------------*/
/*                 YOU MUST «<NOT»> CHANGE THE FOLLOWING PARAMETERS       */
/*-------------------------------------------------------------------------*/
typedef struct {
    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;
}TRAFFIC_CONFIGS_t;

extern TRAFFIC_CONFIGS_t TRAFFIC_Configs;

#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`)

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

1.0.0

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

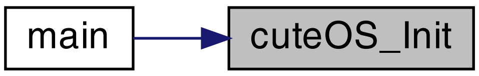
< Enable Timer 2 interrupt

< 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 )
```

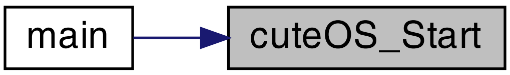
Start the Cute Embedded Operating System (cuteOS)

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 )
```

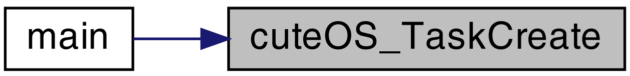
Create a task with the given task function and the given tick time.

This function does the following:

- Increment the task counter.
- Set the task ID.
- Set the pointer to the task function.
- Set the number of schedular 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

in	*callback*	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:

main
cuteOS_TaskRemove

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

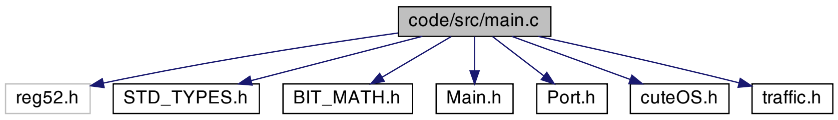
```
#include <reg52.h>
#include "STD_TYPES.h"
#include "main.H"
#include "cuteOS.h"


/*-------------------------------------------------------------------*/
/*                          PRIVATE DATA                             */
/*-------------------------------------------------------------------*/

#define MAX_TICK_TIME_MS  65

static u8 cuteOS_TickTimeMs = 50;

static u16 cuteOS_TickCount = 0;


#define MAX_TASKS_NUM   10

static u8 cuteOS_TaskCounter = 0;
typedef struct {
    ERROR_t (*callback)(void);
    u16 ticks;
    u8  id;
}cuteOS_TASK_t;

cuteOS_TASK_t tasks[MAX_TASKS_NUM] = {0};






/*-------------------------------------------------------------------*/
/*                          PRIVATE FUNCTIONS                        */
/*-------------------------------------------------------------------*/

static void cuteOS_ISR() interrupt INTERRUPT_Timer_2_Overflow {
    u8 i = 0;

    TF2 = 0;

    ++cuteOS_TickCount;

    for(i = 0; i < cuteOS_TaskCounter; ++i) {
        if( (cuteOS_TickCount % tasks[i].ticks) == 0) {
            // cuteOS_TickCount = 0;

            if(tasks[i].callback != NULL) {
                tasks[i].callback();
            }
        }
    }
}


static void cuteOS_Sleep(void) {
    PCON |= 0x01;
}


/*-------------------------------------------------------------------*/
/*                          PUBLIC FUNCTIONS                         */
/*-------------------------------------------------------------------*/

ERROR_t cuteOS_TaskCreate(ERROR_t (* const callback)(void), const u16 TICK_MS) {
    ERROR_t error = ERROR_NO;

    ++cuteOS_TaskCounter;

    if(cuteOS_TaskCounter > MAX_TASKS_NUM) {
        error = ERROR_OUT_OF_RANGE;
    } else {
        tasks[cuteOS_TaskCounter - 1].id = cuteOS_TaskCounter - 1;
        tasks[cuteOS_TaskCounter - 1].ticks = TICK_MS / cuteOS_TickTimeMs;
        tasks[cuteOS_TaskCounter - 1].callback = callback;
    }

    return error;
}


ERROR_t cuteOS_TaskRemove(ERROR_t (* const callback)(void)) {
    ERROR_t error = ERROR_YES;
    u8 i = 0;

    for(i = 0; i < cuteOS_TaskCounter; ++i) {
        if(tasks[i].callback == callback) {
            error = ERROR_NO;
            for(; i < cuteOS_TaskCounter - 1; ++i) {
                tasks[i] = tasks[i + 1];
            }
            tasks[cuteOS_TaskCounter - 1].callback = NULL;

            --cuteOS_TaskCounter;
            break;
        }
    }

    return error;
}


void cuteOS_Start(void) {
    while(1) {
        cuteOS_Sleep();
    }
}


ERROR_t cuteOS_SetTickTime(const u8 TICK_MS){
    ERROR_t error = ERROR_NO;

    cuteOS_TickTimeMs = TICK_MS;

    if(cuteOS_TickTimeMs > MAX_TICK_TIME_MS) {
        error = ERROR_OUT_OF_RANGE;
    } else {
        cuteOS_Init();
    }

    return ERROR_NO;
}


ERROR_t cuteOS_GetTickTime(u8 * const tickTimeInMsPtr){
    ERROR_t error = ERROR_NO;

    if(tickTimeInMsPtr != NULL) {
        *tickTimeInMsPtr = cuteOS_TickTimeMs;
    } else {
        error |= ERROR_NULL_POINTER;
    }

    return error;
}


ERROR_t cuteOS_Init(void) {
    ERROR_t error = ERROR_NO;
    u32 Inc;
    u16 Reload_16;
    u8 Reload_08H, Reload_08L;

    TR2 = 0;

    T2CON = 0x04;
    Inc = ((u32)cuteOS_TickTimeMs * (OSC_FREQ/1000)) / (u32)OSC_PER_INST;

    Reload_16 = (u16) (65536UL - Inc);

    Reload_08H = (u8)(Reload_16 / 256);
    Reload_08L = (u8)(Reload_16 % 256);
    // Used for manually checking timing (in simulator)
    //P2 = Reload_08H;
    //P3 = Reload_08L;
    RCAP2H = TH2 = Reload_08H;
    RCAP2L = TL2 = Reload_08L;
    ET2 = 1;
    TR2 = 1;
    EA = 1;
    return error;
}

```

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.

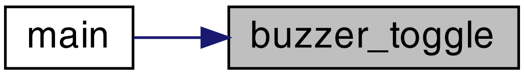
4.17.2 Function Documentation

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:

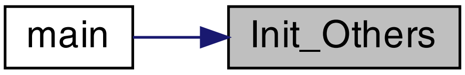


4.17.2.2 Init_Others()

```
void Init_Others (
        void )
```

Definition at line 67 of file main.c.

Here is the caller graph for this function:

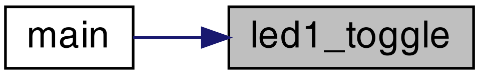


4.17.2.3 led1_toggle()

```
void led1_toggle (
        void )
```

Definition at line 47 of file main.c.

Here is the caller graph for this function:

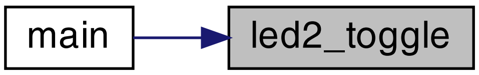


4.17.2.4 led2_toggle()

```
void led2_toggle (
        void )
```

Definition at line 51 of file main.c.

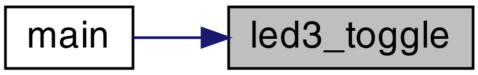
Here is the caller graph for this function:



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

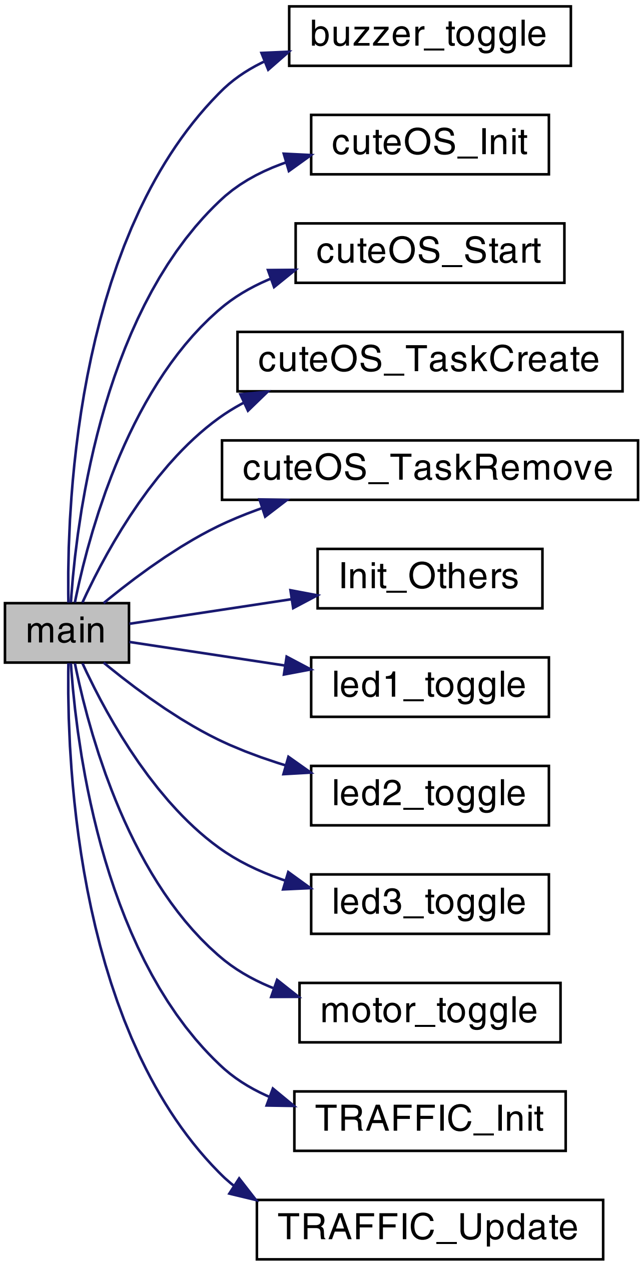
< Remove the task to toggle the buzzer

< 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:

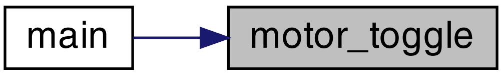


4.17.2.7 motor_toggle()

```
void motor_toggle (
        void )
```

Definition at line 59 of file main.c.

Here is the caller graph for this function:



4.18 main.c

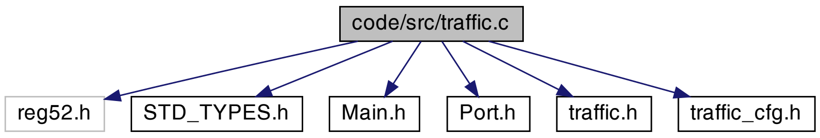
```
#include <reg52.h>
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "Main.h"
#include "Port.h"
#include "cuteOS.h"
#include "traffic.h"

/*---------------------------------------------------------------------------------*/
/*  THE FOLLOWING ARE ONLY FOR TESTING THE SIMPLE OS.                              */
/*  DO NOT USE THEM IN YOUR APPLICATION.                                           */
/*---------------------------------------------------------------------------------*/
void led1_toggle(void){
    led1Pin = !led1Pin;
}

void led2_toggle(void){
    led2Pin = !led2Pin;
}

void led3_toggle(void){
    led3Pin = !led3Pin;
}

void motor_toggle(void){
    motorPin = !motorPin;
}

void buzzer_toggle(void){
    buzzerPin = !buzzerPin;
}

void Init_Others(void) {
    led1Pin = 1;
    led2Pin = 0;
    led3Pin = 0;
    motorPin = 1;
    buzzerPin = 0;
}






/*---------------------------------------------------------------------------------*/
/*                  APPLICATION MAIN FUNCTION                                      */
/*---------------------------------------------------------------------------------*/
void main(void) {
    /* Initialize the system */
    cuteOS_Init();
    TRAFFIC_Init();
    Init_Others();
    cuteOS_TaskCreate(TRAFFIC_Update, 1000);
    cuteOS_TaskCreate(led1_toggle    , 1000);
    cuteOS_TaskCreate(led2_toggle    , 2000);
    cuteOS_TaskCreate(led3_toggle    , 4000);
    cuteOS_TaskCreate(buzzer_toggle    , 1000);
    cuteOS_TaskCreate(motor_toggle , 5000);
    cuteOS_TaskRemove(buzzer_toggle);
    cuteOS_Start();

    while(1);
}
```

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 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	*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 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:

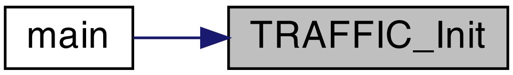
- Initialize the traffic light system to RED sequence.
- Set the callback function of the OS delay to the function TRAFFIC_Update.

< 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: • RED • RED_AMBER • GREEN • AMBER Those colors are members of the global enumeration TRAFFIC_SEQUENCE_t.

Returns

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

Definition at line 162 of file traffic.c.

4.20 traffic.c

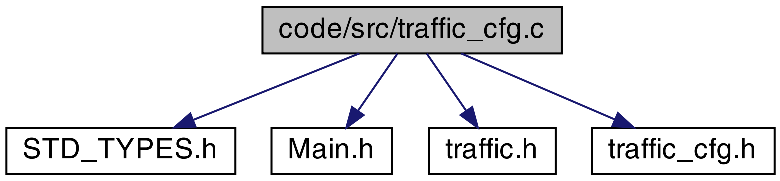
```c

#include <reg52.h>
#include "STD_TYPES.h"
#include "Main.h"
#include "Port.h"
#include "traffic.h"
#include "traffic_cfg.h"

/*--------------------------------------------------------------------*/
/*                          PRIVATE DATA                              */
/*--------------------------------------------------------------------*/
static TRAFFIC_SEQUENCE_t   colorSequence       = RED;
static u16                  timeInState = 0;
/*--------------------------------------------------------------------*/
/*                    PRIVATE FUNCTIONS PROTOTYPES                    */
/*--------------------------------------------------------------------*/

static ERROR_t TRAFFIC_Update(void);


static ERROR_t TRAFFIC_RedSequence(void);


static ERROR_t TRAFFIC_RedAmberSequence(void);


static ERROR_t TRAFFIC_GreenSequence(void);


static ERROR_t TRAFFIC_AmberSequence(void);


static ERROR_t TRAFFIC_GenericSequence(const STATE_t red, const STATE_t amber, const STATE_t green,
 TRAFFIC_SEQUENCE_DURATION_t duration);







/*--------------------------------------------------------------------*/
/*                          PUBLIC FUNCTIONS                          */
/*--------------------------------------------------------------------*/

ERROR_t TRAFFIC_Init(void) {
    ERROR_t error = ERROR_NO;

    timeInState = 0;

    colorSequence = RED;
    redPin    = HIGH;
    amberPin = LOW;
    greenPin = LOW;

    return error;
}


ERROR_t TRAFFIC_DeInit(void) {
    ERROR_t error = ERROR_NO;

    redPin    = LOW;
    amberPin = LOW;
    greenPin = LOW;

    //cuteOS_(NULL);

    return error;
}

ERROR_t TRAFFIC_SetColor(const TRAFFIC_SEQUENCE_t Copy_color) {
    ERROR_t error = ERROR_NO;
    colorSequence = Copy_color;

    error |= TRAFFIC_Update();

    return error;
}

ERROR_t TRAFFIC_GetColor(TRAFFIC_SEQUENCE_t * const Copy_color) {
    ERROR_t error = ERROR_NO;

    *Copy_color = colorSequence;

    return error;
}






/*--------------------------------------------------------------------*/
/*                  PRIVATE FUNCTIONS DEFINITIONS                     */
/*--------------------------------------------------------------------*/

ERROR_t TRAFFIC_Update(void) {
    ERROR_t error = ERROR_NO;

    switch(colorSequence) {
        case RED:;
            error |= TRAFFIC_RedSequence();
            break;
        case RED_AMBER:;
            error |= TRAFFIC_RedAmberSequence();
            break;
        case GREEN:;
            error |= TRAFFIC_GreenSequence();
            break;
        case AMBER:;
            error |= TRAFFIC_AmberSequence();
            break;
        default:;
            error |= ERROR_ILLEGAL_PARAM;
            break;
    }

    return error;
}

static ERROR_t TRAFFIC_RedSequence(void) {
    ERROR_t error = ERROR_NO;

    error |= TRAFFIC_GenericSequence(HIGH, LOW, LOW, TRAFFIC_Configs.red_duration);

    return error;
}

static ERROR_t TRAFFIC_RedAmberSequence(void) {
    ERROR_t error = ERROR_NO;

    error |= TRAFFIC_GenericSequence(HIGH, HIGH, LOW, TRAFFIC_Configs.red_amber_duration);

    return error;
}

static ERROR_t TRAFFIC_GreenSequence(void) {
    ERROR_t error = ERROR_NO;

    error |= TRAFFIC_GenericSequence(LOW, LOW, HIGH, TRAFFIC_Configs.green_duration);

    return error;
}

static ERROR_t TRAFFIC_AmberSequence(void) {
    ERROR_t error = ERROR_NO;

    error |= TRAFFIC_GenericSequence(LOW, HIGH, LOW, TRAFFIC_Configs.amber_duration);

    return error;
}


static ERROR_t TRAFFIC_GenericSequence(const STATE_t redState, const STATE_t amberState, const STATE_t
 greenState, TRAFFIC_SEQUENCE_DURATION_t duration) {
    ERROR_t error = ERROR_NO;
    u8 tickTime = 0;

    if(++timeInState >= duration) {
        timeInState = 0;
        switch(colorSequence) {
            case RED:
                colorSequence = RED_AMBER;
                redPin   = HIGH;
                amberPin = HIGH;
                greenPin = LOW;
                break;
            case RED_AMBER:
                colorSequence = GREEN;
                redPin   = LOW;
                amberPin = LOW;
                greenPin = HIGH;
                break;
            case GREEN:
                colorSequence = AMBER;
                redPin   = LOW;
                amberPin = HIGH;
                greenPin = LOW;
                break;
            case AMBER:
                colorSequence = RED;
                redPin   = HIGH;
                amberPin = LOW;
                greenPin = LOW;
                break;
            default:
                error |= ERROR_ILLEGAL_PARAM;
                break;
        }
    } else {
        redPin   = redState;
        amberPin = amberState;
        greenPin = greenState;
    }

    return error;
}
```

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