Washing Machine

1.0.0

Generated by Doxygen 1.9.1

1 Data Structure Index 1

1.1 Data Structures . . . 1

2 File Index 2

2.1 File List . . . 2

3 Data Structure Documentation 2

3.1 cuteOS_TASK_t Struct Reference . . . 2

3.1.1 Detailed Description . . . 2

3.1.2 Field Documentation . . . 3

3.2 DEVICES_ACTIVE_STATE_t Struct Reference . . . 3

3.2.1 Detailed Description . . . 3

3.2.2 Field Documentation . . . 4

3.3 WASHER_CONFIGS_t Struct Reference . . . 5

3.3.1 Detailed Description . . . 6

3.3.2 Field Documentation . . . 6

4 File Documentation 6

4.1 code/include/BIT_MATH.h File Reference . . . 6

4.1.1 Detailed Description . . . 7

4.1.2 Macro Definition Documentation . . . 7

4.2 BIT_MATH.h . . . 11

4.3 code/include/cuteOS.h File Reference . . . 12

4.3.1 Detailed Description . . . 12

4.3.2 Function Documentation . . . 13

4.4 cuteOS.h . . . 18

4.5 code/include/main.h File Reference . . . 19

4.5.1 Detailed Description . . . 19

4.5.2 Macro Definition Documentation . . . 19

4.6 main.h . . . 20

4.7 code/include/port.h File Reference . . . 21

4.7.1 Detailed Description . . . 21

4.7.2 Variable Documentation . . . 22

4.8 port.h . . . 23

4.9 code/include/STD_TYPES.h File Reference . . . 23

4.9.1 Detailed Description . . . 24

4.9.2 Macro Definition Documentation . . . 24

4.9.3 Typedef Documentation . . . 25

4.9.4 Enumeration Type Documentation . . . 26

4.10 STD_TYPES.h . . . 27

4.11 code/include/washer.h File Reference . . . 28

4.11.1 Detailed Description . . . 28

4.11.2 Function Documentation . . . 29

4.12 washer.h . . . 30
4.13 code/include/washer_cfg.h File Reference . . . 30
4.13.1 Detailed Description . . . 31
4.13.2 Enumeration Type Documentation . . . 31
4.13.3 Variable Documentation . . . 32
4.14 washer_cfg.h . . . 32
4.15 code/src/cuteOS.c File Reference . . . 33
4.15.1 Detailed Description . . . 34
4.15.2 Macro Definition Documentation . . . 34
4.15.3 Function Documentation . . . 35
4.15.4 Variable Documentation . . . 38
4.16 cuteOS.c . . . 39
4.17 code/src/main.c File Reference . . . 41
4.17.1 Detailed Description . . . 41
4.17.2 Function Documentation . . . 42
4.18 main.c . . . 43
4.19 code/src/washer.c File Reference . . . 43
4.19.1 Detailed Description . . . 43
4.19.2 Function Documentation . . . 44
4.20 washer.c . . . 45
4.21 code/src/washer_cfg.c File Reference . . . 51
4.21.1 Detailed Description . . . 51
4.21.2 Variable Documentation . . . 52
4.22 washer_cfg.c . . . 52
4.23 code/src/washer_private.h File Reference . . . 53
4.23.1 Detailed Description . . . 53
4.23.2 Macro Definition Documentation . . . 54
4.24 washer_private.h . . . 56

Index **57**

1 Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

cuteOS_TASK_t **2**

DEVICES_ACTIVE_STATE_t **3**

WASHER_CONFIGS_t **5**

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

code/include/BIT_MATH.h
Common bit manipulation operations 6

code/include/cuteOS.h
Simple EOS interfaces header file. See cuteOS.c for more details 12

code/include/main.h
Project Header for main.c 19

code/include/port.h
Port Header file for 8052 microcontroller 21

code/include/STD_TYPES.h
Standard data types For 8051 Microcontrollers 23

code/include/washer.h
Dinosaur Animation System interfaces header file. See washer.c for more details 28

code/include/washer_cfg.h
Washer Machine System interfaces header file. See washer.c for more details 30

code/src/cuteOS.c
Main file for Cute Embedded Operating System (cuteOS) for 8051 33

code/src/main.c
Washing machine system 41

code/src/washer.c
This is a washer animation system 43

code/src/washer_cfg.c
Configurations of Washer MAchine System 51

code/src/washer_private.h
This is a private header file for the washer class 53

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 DEVICES_ACTIVE_STATE_t Struct Reference

`#include <washer_cfg.h>`

Data Fields

- ACTIVATION_STATUS_t selectorCotton
- ACTIVATION_STATUS_t start
- ACTIVATION_STATUS_t level
- ACTIVATION_STATUS_t detergent
- ACTIVATION_STATUS_t doorLock
- ACTIVATION_STATUS_t washer
- ACTIVATION_STATUS_t drain
- ACTIVATION_STATUS_t heater
- ACTIVATION_STATUS_t valve

3.2.1 Detailed Description

Definition at line 30 of file washer_cfg.h.

3.2.2 Field Documentation

3.2.2.1 detergent ACTIVATION_STATUS_t detergent

Which state activates the detergent

Definition at line 35 of file washer_cfg.h.

3.2.2.2 doorLock ACTIVATION_STATUS_t doorLock

Which state activates the door lock

Definition at line 36 of file washer_cfg.h.

3.2.2.3 drain ACTIVATION_STATUS_t drain

Which state activates the drain pump

Definition at line 38 of file washer_cfg.h.

3.2.2.4 heater ACTIVATION_STATUS_t heater

Which state activates the heater

Definition at line 39 of file washer_cfg.h.

3.2.2.5 level ACTIVATION_STATUS_t level

Which state the level sensors detect water

Definition at line 33 of file washer_cfg.h.

3.2.2.6 selectorCotton ACTIVATION_STATUS_t selectorCotton

Which state of the selector dial to consider the cotton program

Definition at line 31 of file washer_cfg.h.

3.2.2.7 start ACTIVATION_STATUS_t start

Which state of start button to start the washing machine

Definition at line 32 of file washer_cfg.h.

3.2.2.8 valve ACTIVATION_STATUS_t valve

Which state activates the valve

Definition at line 40 of file washer_cfg.h.

3.2.2.9 washer ACTIVATION_STATUS_t washer

Which state activates the washer

Definition at line 37 of file washer_cfg.h.

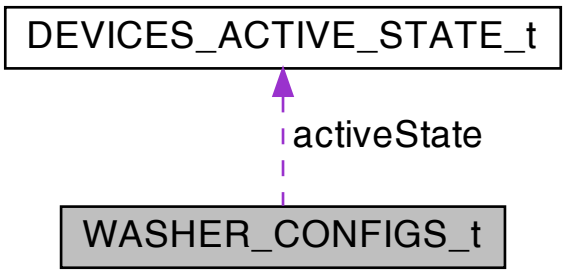
The documentation for this struct was generated from the following file:

- code/include/washer_cfg.h

3.3 WASHER_CONFIGS_t Struct Reference

`#include <washer_cfg.h>`

Collaboration diagram for WASHER_CONFIGS_t:



Data Fields

- WASHER_STATE_t state
- const u8 thresholdTemperature
- const DEVICES_ACTIVE_STATE_t activeState

3.3.1 Detailed Description

Definition at line 44 of file washer_cfg.h.

3.3.2 Field Documentation

3.3.2.1 activeState `const DEVICES_ACTIVE_STATE_t activeState`

Definition at line 47 of file washer_cfg.h.

3.3.2.2 state `WASHER_STATE_t state`

Definition at line 45 of file washer_cfg.h.

3.3.2.3 thresholdTemperature `const u8 thresholdTemperature`

Threshold temperature in degrees Celsius

Definition at line 46 of file washer_cfg.h.

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

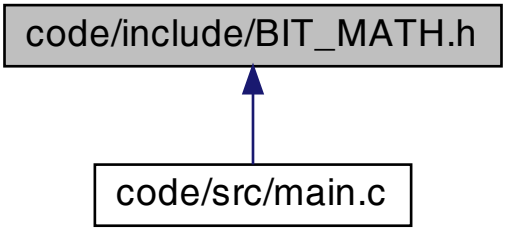
- code/include/washer_cfg.h

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

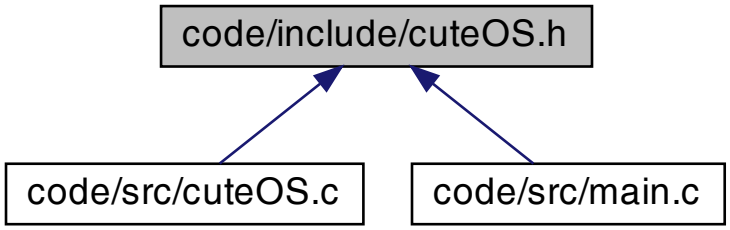
```
00001
00008 #ifndef BIT_MATH_H
00009 #define BIT_MATH_H
00010
00011
00019 #define GET_BIT(REGISTER, BIT)      ( 1 & ( (REGISTER) » (BIT) ) )
00020
00021
00028 #define SET_BIT(REGISTER, BIT)      ( (REGISTER) |= (1 « (BIT)) )
00029
00030
00037 #define CLR_BIT(REGISTER, BIT)      ( (REGISTER) &= ~(1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

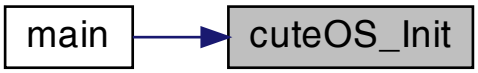
< Enable Timer 2 interrupt

< 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

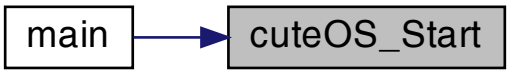
The next clock tick will return the processor to the normal operating state.

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.

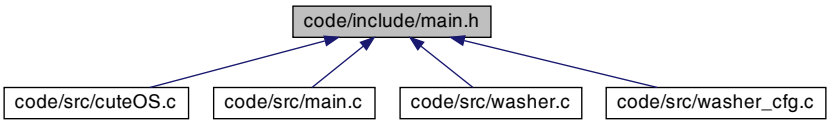
4.4 cuteOS.h

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

4.5 code/include/main.h File Reference

Project Header for main.c.

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



Macros

- #define OSC_FREQ (12000000UL)
- #define OSC_PER_INST (12)

  Number of oscillations per instruction (12, etc)
- #define INTERRUPT_Timer_0_Overflow 1
- #define INTERRUPT_Timer_1_Overflow 3
- #define INTERRUPT_Timer_2_Overflow 5

4.5.1 Detailed Description

Project Header for main.c.

Author

Mahmoud Karam (ma.karam272@gmail.com)

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.

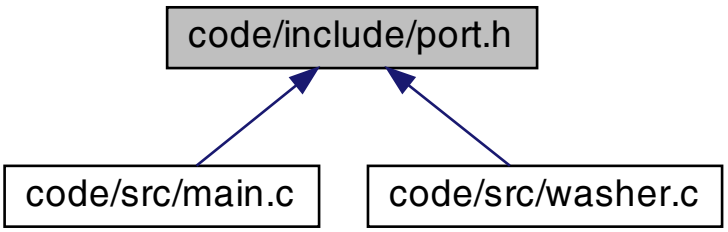
4.6 main.h

```
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 8052 microcontroller.

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



Variables

- sbit selectorPin = P1^0
- sbit startPin = P1^1
- sbit lowLevelPin = P1^2
- sbit highLevelPin = P1^3
- sbit temperaturePin = P1^4
- sbit detergentPin = P2^0
- sbit doorLockPin = P2^1
- sbit washerPin = P2^2
- sbit drainPin = P2^3
- sbit heaterPin = P2^4
- sbit valvePin = P2^5

4.7.1 Detailed Description

Port Header file for 8052 microcontroller.

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 detergentPin `sbit detergentPin = P2^0`

< Output pins

Definition at line 23 of file port.h.

4.7.2.2 doorLockPin `sbit doorLockPin = P2^1`

Definition at line 24 of file port.h.

4.7.2.3 drainPin `sbit drainPin = P2^3`

Definition at line 26 of file port.h.

4.7.2.4 heaterPin `sbit heaterPin = P2^4`

Definition at line 27 of file port.h.

4.7.2.5 highLevelPin `sbit highLevelPin = P1^3`

Definition at line 19 of file port.h.

4.7.2.6 lowLevelPin `sbit lowLevelPin = P1^2`

Definition at line 18 of file port.h.

4.7.2.7 selectorPin `sbit selectorPin = P1^0`

In file washer.C

< Input pins

Definition at line 16 of file port.h.

4.7.2.8 startPin `sbit startPin = P1^1`

Definition at line 17 of file port.h.

4.7.2.9 temperaturePin `sbit temperaturePin = P1^4`

Definition at line 20 of file port.h.

4.7.2.10 valvePin `sbit valvePin = P2^5`

Definition at line 28 of file port.h.

4.7.2.11 washerPin `sbit washerPin = P2^2`

Definition at line 25 of file port.h.

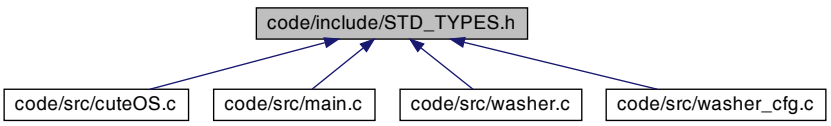
4.8 port.h

```
00001
00009 #ifndef PORT_H
00010 #define PORT_H
00011
00012
00016 sbit selectorPin    = P1^0;   /* Port 1 pin 0 */
00017 sbit startPin       = P1^1;   /* Port 1 pin 1 */
00018 sbit lowLevelPin    = P1^2;   /* Port 1 pin 2 */
00019 sbit highLevelPin   = P1^3;   /* Port 1 pin 3 */
00020 sbit temperaturePin = P1^4;   /* Port 1 pin 3 */
00021
00023 sbit detergentPin   = P2^0;   /* Port 2 pin 0 */
00024 sbit doorLockPin    = P2^1;   /* Port 2 pin 1 */
00025 sbit washerPin      = P2^2;   /* Port 2 pin 2 */
00026 sbit drainPin       = P2^3;   /* Port 2 pin 3 */
00027 sbit heaterPin      = P2^4;   /* Port 2 pin 4 */
00028 sbit valvePin       = P2^5;   /* Port 2 pin 5 */
00029
00030
00031 #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 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 ,
  ERROR_INIT_FAIL = 0x80 }

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

NULL pointer

Definition at line 62 of file STD_TYPES.h.

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

Definition at line 65 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
ERROR_INIT_FAIL	Initialization failed 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

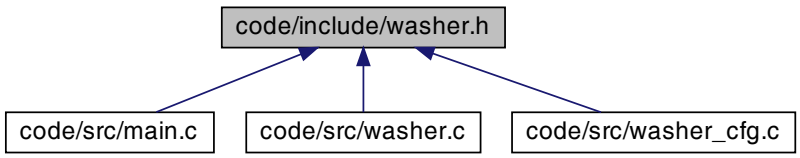
```
00001
00008 #ifndef STD_TYPES_H
00009 #define STD_TYPES_H
00010
00011 /* Signed integers  */
00012 typedef     signed long int             s32;
00013 typedef     signed short int            s16;
00014 typedef     signed char                 s8;
00015
00016 /* Unsigned integers    */
00017 typedef     unsigned long int           u32;
00018 typedef     unsigned short int          u16;
00019 typedef     unsigned char               u8;
00020
00021 /* Float numbers    */
00022 typedef     float                       f32;
00023 typedef     double                      f64;
00024
00025 /* Special types    */
00026  #undef __SIZE_TYPE__
00027 typedef     u16   size_t;
00029 #undef HIGH
00030 #undef LOW
00031 typedef enum{
00032   LOW,
```

```
00033   HIGH,
00034   NORMAL,             /* Used for any normal state */
00035 }STATE_t;
00036
00037 typedef enum{
00038     ACTIVE_LOW,
00039     ACTIVE_HIGH,
00040 }ACTIVATION_STATUS_t;
00041
00042 /* Boolean type */
00043 typedef enum{
00044     FALSE,
00045     TRUE
00046 }BOOL_t;
00047
00048 typedef enum{
00049     ERROR_NO                = 0,
00050     ERROR_YES               = 0x1,
00051     ERROR_TIMEOUT           = 0x2,
00052     ERROR_NULL_POINTER      = 0x4,
00053     ERROR_BUSY              = 0x8,
00054     ERROR_NOT_INITIALIZED   = 0x10,
00055     ERROR_ILLEGAL_PARAM     = 0x20,
00056     ERROR_OUT_OF_RANGE      = 0x40,
00057     ERROR_INIT_FAIL         = 0x80,
00058 }ERROR_t;
00059
00060 /* Pointers */
00061 #undef NULL
00062 #define NULL ((void *)0)
00064 #undef NULL_BYTE
00065 #define NULL_BYTE  ('\0')
00066
00067  #endif   /* STD_TYPES_H */
```

4.11 code/include/washer.h File Reference

Dinosaur Animation System interfaces header file. See washer.c for more details.

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



Functions

- ERROR_t WASHER_Init (void)

  Initialize the washer system according to the configurations in the WASHER_CONFIGS_t structure.
- ERROR_t WASHER_Update (void)

4.11.1 Detailed Description

Dinosaur Animation System interfaces header file. See washer.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 washer.h.

4.11.2 Function Documentation

4.11.2.1 WASHER_Init() ERROR_t WASHER_Init (void)

Initialize the washer system according to the configurations in the WASHER_CONFIGS_t structure.

Returns

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

< Setting input pins

< Turning off output pins

< Error handling

< No error

Definition at line 26 of file washer.c.

Here is the caller graph for this function:

4.11.2.2 WASHER_Update() ERROR_t WASHER_Update (void)

Definition at line 49 of file washer.c.

Here is the caller graph for this function:

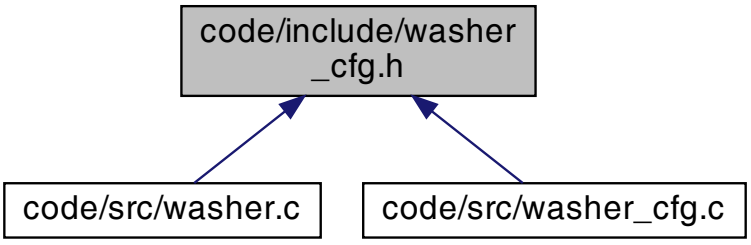
4.12 washer.h

```
00001
00010 #ifndef WASHER_H
00011 #define WASHER_H
00012
00013
00014 /*--------------------------------------------------------------------------*/
00015 /*                          API FUNCTIONS                                   */
00016 /*--------------------------------------------------------------------------*/
00017
00018
00023 ERROR_t WASHER_Init(void);
00024
00025 ERROR_t WASHER_Update(void);
00026
00027
00028
00029 #endif          /* WASHER_H */
```

4.13 code/include/washer_cfg.h File Reference

Washer Machine System interfaces header file. See washer.c for more details.

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



Data Structures

- struct DEVICES_ACTIVE_STATE_t
- struct WASHER_CONFIGS_t

Enumerations

- enum WASHER_STATE_t {
  WASHER_STATE_INIT , WASHER_STATE_FILL_DRUM , WASHER_STATE_HEATING_WATER ,
  WASHER_STATE_WASHING_WATER ,
  WASHER_STATE_DRAIN_DRUM , WASHER_STATE_END }

Variables

- WASHER_CONFIGS_t WASHER_Configs
  Initial configuration of the washer system.

4.13.1 Detailed Description

Washer Machine System interfaces header file. See washer.c for more details.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-22

Copyright

Definition in file washer_cfg.h.

4.13.2 Enumeration Type Documentation

4.13.2.1 WASHER_STATE_t enum WASHER_STATE_t

< Washing machine states

Enumerator

WASHER_STATE_INIT	
WASHER_STATE_FILL_DRUM	
WASHER_STATE_HEATING_WATER	
WASHER_STATE_WASHING_WATER	
WASHER_STATE_DRAIN_DRUM	
WASHER_STATE_END	

Definition at line 20 of file washer_cfg.h.

4.13.3 Variable Documentation

4.13.3.1 WASHER_Configs `WASHER_CONFIGS_t WASHER_Configs [extern]`

Initial configuration of the washer system.

Definition at line 21 of file washer_cfg.c.

4.14 washer_cfg.h

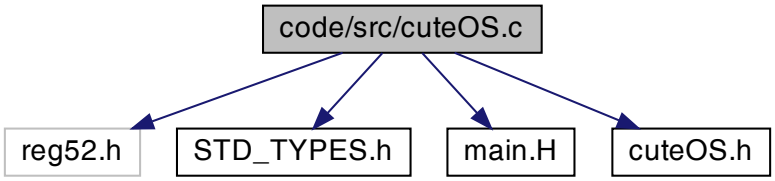
```
00001
00009 #ifndef WASHER_CFG_H
00010 #define WASHER_CFG_H
00011
00012 /*-------------------------------------------------------------------------*/
00013 /*                YOU MUST «<NOT»> CHANGE THE FOLLOWING                    */
00014 /*-------------------------------------------------------------------------*/
00015
00016 /*-----------------------------------------------------------------*/
00017 /*  PRIVATE TYPE DEFINITIONS                                       */
00018 /*-----------------------------------------------------------------*/
00020 typedef enum{
00021     WASHER_STATE_INIT,
00022     WASHER_STATE_FILL_DRUM,
00023     WASHER_STATE_HEATING_WATER,
00024     WASHER_STATE_WASHING_WATER,
00025     WASHER_STATE_DRAIN_DRUM,
00026     WASHER_STATE_END,
00027 } WASHER_STATE_t;
00028
00030 typedef struct {
00031     ACTIVATION_STATUS_t selectorCotton;
00032     ACTIVATION_STATUS_t start;
00033     ACTIVATION_STATUS_t level;
00035     ACTIVATION_STATUS_t detergent;
00036     ACTIVATION_STATUS_t doorLock;
00037     ACTIVATION_STATUS_t washer;
00038     ACTIVATION_STATUS_t drain;
00039     ACTIVATION_STATUS_t heater;
00040     ACTIVATION_STATUS_t valve;
00041 } DEVICES_ACTIVE_STATE_t;
00042
00044 typedef struct {
00045     WASHER_STATE_t state;
00046     const u8 thresholdTemperature;
00047     const DEVICES_ACTIVE_STATE_t activeState;
00048 }WASHER_CONFIGS_t;
00049
00050
00051 extern WASHER_CONFIGS_t WASHER_Configs;
00052
00053
00054 #endif /* WASHER_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

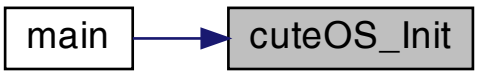
< Enable Timer 2 interrupt

< 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)

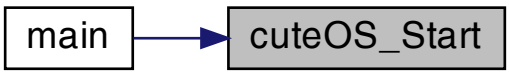
Start the Cute Embedded Operating System (cuteOS)

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.

4.15.4 Variable Documentation

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

Definition at line 66 of file cuteOS.c.

4.16 cuteOS.c

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

```
00136         tasks[cuteOS_TaskCounter - 1].callback = callback;
00137     }
00138
00139     return error;
00140 }
00141
00142
00152 ERROR_t cuteOS_TaskRemove(ERROR_t (* const callback)(void)) {
00153     ERROR_t error = ERROR_YES;
00154     u8 i = 0;
00155
00157     for(i = 0; i < cuteOS_TaskCounter; ++i) {
00158         if(tasks[i].callback == callback) {
00159             error = ERROR_NO;
00161             for(; i < cuteOS_TaskCounter - 1; ++i) {
00162                 tasks[i] = tasks[i + 1];
00163             }
00165             tasks[cuteOS_TaskCounter - 1].callback = NULL;
00166
00168             --cuteOS_TaskCounter;
00169             break;
00170         }
00171     }
00172
00173     return error;
00174 }
00175
00176
00179 void cuteOS_Start(void) {
00181     while(1) {
00182         cuteOS_Sleep();
00183     }
00184 }
00185
00186
00191 ERROR_t cuteOS_SetTickTime(const u8 TICK_MS){
00192     ERROR_t error = ERROR_NO;
00193
00194     cuteOS_TickTimeMs = TICK_MS;
00195
00196     if(cuteOS_TickTimeMs > MAX_TICK_TIME_MS) {
00197         error = ERROR_OUT_OF_RANGE;
00198     } else {
00200         cuteOS_Init();
00201     }
00202
00203     return ERROR_NO;
00204 }
00205
00206
00209 ERROR_t cuteOS_GetTickTime(u8 * const tickTimeInMsPtr){
00210     ERROR_t error = ERROR_NO;
00211
00212     if(tickTimeInMsPtr != NULL) {
00213         *tickTimeInMsPtr = cuteOS_TickTimeMs;
00214     } else {
00215         error |= ERROR_NULL_POINTER;
00216     }
00217
00218     return error;
00219 }
00220
00221
00228 ERROR_t cuteOS_Init(void) {
00229     ERROR_t error = ERROR_NO;
00230     u32 Inc;
00231     u16 Reload_16;
00232     u8 Reload_08H, Reload_08L;
00233
00234     TR2 = 0;
00236
00240     T2CON = 0x04;
00248     Inc = ((u32)cuteOS_TickTimeMs * (OSC_FREQ/1000)) / (u32)OSC_PER_INST;
00249
00251     Reload_16 = (u16) (65536UL - Inc);
00252
00254     Reload_08H = (u8)(Reload_16 / 256);
00255     Reload_08L = (u8)(Reload_16 % 256);
00257     // Used for manually checking timing (in simulator)
00258     //P2 = Reload_08H;
00259     //P3 = Reload_08L;
00260     RCAP2H = TH2 = Reload_08H;
00261     RCAP2L = TL2 = Reload_08L;
00263     ET2 = 1;
00264     TR2 = 1;
00265     EA = 1;
00267     return error;
```

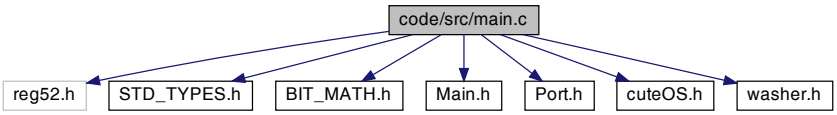
```
00268 }
00269
```

4.17 code/src/main.c File Reference

Washing machine system.

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

Include dependency graph for main.c:



Functions

- void main (void)

4.17.1 Detailed Description

Washing machine system.

Author

Mahmoud Karam (ma.karam272@gmail.com)

This project is a washing machine system. It works as the following: 1- The user selects a wash program (e.g. 'Wool', 'Cotton') on the selector dial. 2- The user presses the 'Start' switch. 3- The door lock is engaged. 4- The water valve is opened to allow water into the wash drum. 5- If the wash program involves detergent, the detergent hatch is opened. When the detergent has been released, the detergent hatch is closed. 6- When the 'full water level' is sensed, the water valve is closed. 7- If the wash program involves warm water, the water heater is switched on. When the water reaches the correct temperature, the water heater is switched off. 8- The washer motor is turned on to rotate the drum. The motor then goes through a series of movements, both forward and reverse (at various speeds) to wash the clothes. (The precise set of movements carried out depends on the wash program that the user has selected.) At the end of the wash cycle, the motor is stopped. 9- The pump is switched on to drain the drum. When the drum is empty, the pump is switched off.

Application usage: 1- In main() function, Initialize the OS and the washer.

```
cuteOS_Init();
WASHER_Init();
```

2- In main() function, create the washer task with 1 second period.

```
cuteOS_TaskCreate(WASHER_Update, 1000);
```

3- In main() function, start the Cute OS scheduler.

```
cuteOS_Start();
```

4- Everything is done. The scheduler will call the washer task every second to animate the washer (WASHER_Update()). 5- The washer task will either remain in the previous state or will change to the next state.

Version

1.0.0

Date

2022-03-24

Copyright

Copyright (c) 2022

Application usage:

Definition in file main.c.

4.17.2 Function Documentation

4.17.2.1 main() `void main ( void )`

< Initialize Cute OS

< Initialize the washer light system

< Create the tasks

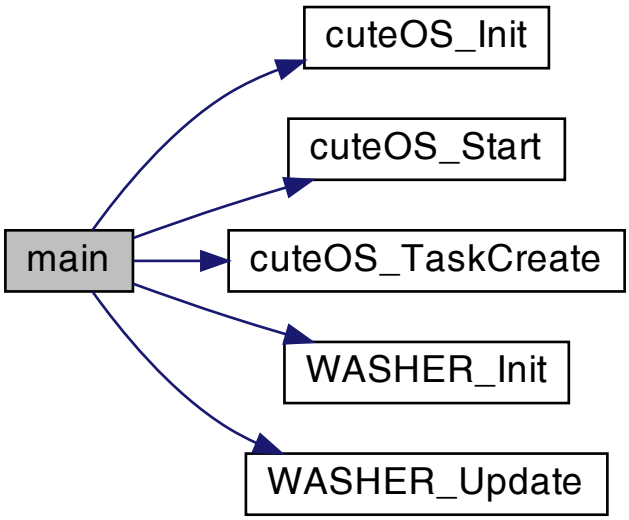
< Create a task to run the washer light system

< Start the Cute OS scheduler

< The scheduler will never return from here

Definition at line 55 of file main.c.

Here is the call graph for this function:



4.18 main.c

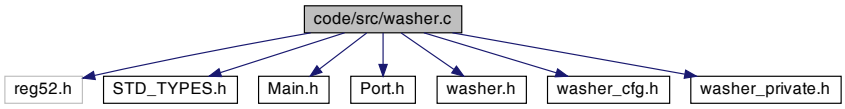
```
00001
00047 #include <reg52.h>
00048 #include "STD_TYPES.h"
00049 #include "BIT_MATH.h"
00050 #include "Main.h"
00051 #include "Port.h"
00052 #include "cuteOS.h"
00053 #include "washer.h"
00054
00055 void main(void) {
00056     /* Initialize the system */
00057     cuteOS_Init();
00058     WASHER_Init();
00061     cuteOS_TaskCreate(WASHER_Update, 1000);
00064     cuteOS_Start();
00065
00067     while(1);
00068 }
```

4.19 code/src/washer.c File Reference

This is a washer animation system.

```
#include <reg52.h>
#include "STD_TYPES.h"
#include "Main.h"
#include "Port.h"
#include "washer.h"
#include "washer_cfg.h"
#include "washer_private.h"
```

Include dependency graph for washer.c:



Functions

- ERROR_t WASHER_Init (void)

  Initialize the washer system according to the configurations in the WASHER_CONFIGS_t structure.
- ERROR_t WASHER_Update (void)

4.19.1 Detailed Description

This is a washer animation system.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-22

Copyright

Copyright (c) 2022

Definition in file washer.c.

4.19.2 Function Documentation

4.19.2.1 WASHER_Init() `ERROR_t WASHER_Init ( void )`

Initialize the washer system according to the configurations in the WASHER_CONFIGS_t structure.

Returns

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

< Setting input pins

< Turning off output pins

< Error handling

< No error

Definition at line 26 of file washer.c.

Here is the caller graph for this function:

4.19.2.2 WASHER_Update() ERROR_t WASHER_Update (void)

Definition at line 49 of file washer.c.

Here is the caller graph for this function:

4.20 washer.c

```
00001
00009 #include <reg52.h>
00010 #include "STD_TYPES.h"
00011 #include "Main.h"
00012 #include "Port.h"
00013 #include "washer.h"
00014 #include "washer_cfg.h"
00015 #include "washer_private.h"
00016
00017 /*-----------------------------------------------------------------------*/
00018 /* PRIVATE DATA                                                          */
00019 /*-----------------------------------------------------------------------*/
00021 static u16 timeInState = 0;
00022
00023 /*-----------------------------------------------------------------------*/
00024 /*  PUBLIC FUNCTIONS DEFINITIONS                                         */
00025 /*-----------------------------------------------------------------------*/
00026 ERROR_t WASHER_Init(void) {
00027     ERROR_t error = ERROR_NO;
00028
00030     selectorPin     = 0;
00031     startPin        = 0;
00032     lowLevelPin     = 0;
00033     highLevelPin    = 0;
00034     temperaturePin  = 0;
00035
00037     error |= WASHER_InitState();
00038
00040     if(ERROR_NO != error) {
00041         WASHER_HandleError(error);
00042     } else {
00044     }
00045
00046     return error;
00047 }
00048
00049 ERROR_t WASHER_Update(void) {
00050     ERROR_t error = ERROR_NO;
00051
00052     switch(WASHER_Configs.state) {
00053         case WASHER_STATE_INIT:
00054             WASHER_InitState();
00055             break;
00056         case WASHER_STATE_FILL_DRUM:
00057             WASHER_FillDrumState();
00058             break;
00059         case WASHER_STATE_HEATING_WATER:
00060             WASHER_HeatingWaterState();
00061             break;
00062         case WASHER_STATE_WASHING_WATER:
00063             WASHER_WashingWaterState();
00064             break;
00065         case WASHER_STATE_DRAIN_DRUM:
00066             WASHER_DrainState();
00067             break;
00068         case WASHER_STATE_END:
00069             WASHER_EndState();
```

```
00070             break;
00071         default:
00072             error |= ERROR_ILLEGAL_PARAM;
00073             break;
00074     }
00075
00076     return error;
00077 }
00078
00079
00080
00081
00082
00083
00084
00085 /*--------------------------------------------------------------------*/
00086 /*                 PRIVATE FUNCTIONS DEFINITIONS                      */
00087 /*--------------------------------------------------------------------*/
00088 static ERROR_t WASHER_ControlAll(const u8 states) {
00089     ERROR_t error = ERROR_NO;
00090
00091     error |= WASHER_ControlDrain(           (STATE_t)( 1 && (states & MASK_DRAIN     ) ) );
00092     error |= WASHER_ControlHeater(          (STATE_t)( 1 && (states & MASK_HEATER    ) ) );
00093     error |= WASHER_ControlValve(           (STATE_t)( 1 && (states & MASK_VALVE     ) ) );
00094     error |= WASHER_ControlDoorLock(        (STATE_t)( 1 && (states & MASK_DOOR_LOCK ) ) );
00095     error |= WASHER_ControlDetergentHatch(  (STATE_t)( 1 && (states & MASK_DETERGENT ) ) );
00096     error |= WASHER_ControlWasher(          (STATE_t)( 1 && (states & MASK_WASHER    ) ) );
00097
00098     return error;
00099 }
00100
00101 static ERROR_t WASHER_EndState(void) {
00102     ERROR_t error = ERROR_NO;
00103
00104     WASHER_Configs.state = WASHER_STATE_END;
00105
00107     if(0 == timeInState) {
00109         error |= WASHER_ControlAll(ALL_OFF);
00110
00111         ++timeInState;
00112     } else {
00114     }
00115
00116     return error;
00117 }
00118
00119 static ERROR_t WASHER_InitState(void) {
00120     ERROR_t error = ERROR_NO;
00121     STATE_t startSwitchState = LOW;
00122
00123     WASHER_Configs.state = WASHER_STATE_INIT;
00124
00126     if(0 == timeInState) {
00128         error |= WASHER_ControlAll(ALL_OFF);
00129     } else {
00131     }
00132     ++timeInState;
00133
00137     error |= WASHER_ReadStartSwitch(&startSwitchState);
00138     if(HIGH == startSwitchState) {
00139         timeInState = 0;
00140         error |= WASHER_FillDrumState();
00141     } else {
00143     }
00144
00146     if(ERROR_NO != error) {
00147         WASHER_HandleError(error);
00148     } else {
00150     }
00151
00152     return error;
00153 }
00154
00155 static ERROR_t WASHER_FillDrumState(void) {
00156     ERROR_t error = ERROR_NO;
00157     STATE_t waterLevel = LOW;
00158
00159     WASHER_Configs.state = WASHER_STATE_FILL_DRUM;
00160
00162     if(0 == timeInState) {
00164         error |= WASHER_ControlAll(MASK_DOOR_LOCK | MASK_VALVE);
00165     } else {
00167     }
00168     ++timeInState;
00169
00171     if(timeInState >= MAX_FILL_DRUM_DURATION) {
00172        error |= ERROR_TIMEOUT;
```

```
00173     } else {
00175     }
00176
00178     error |= WASHER_ReadWaterLevel(&waterLevel);
00179     if(HIGH == waterLevel) {
00180         timeInState = 0;
00181         error |= WASHER_HeatingWaterState();
00182     } else {
00184     }
00185
00187     if(ERROR_NO != error) {
00188         WASHER_HandleError(error);
00189     } else {
00191     }
00192
00193     return error;
00194 }
00195
00196 static ERROR_t WASHER_HeatingWaterState(void) {
00197     ERROR_t error = ERROR_NO;
00198     STATE_t heatState = LOW;
00199
00200     WASHER_Configs.state = WASHER_STATE_HEATING_WATER;
00201
00203     if(0 == timeInState) {
00205         error |= WASHER_ControlAll(MASK_HEATER);
00206     } else {
00208     }
00209     ++timeInState;
00210
00212     error |= WASHER_ReadTemperature(&heatState);
00213     if(HIGH == heatState) {
00214         timeInState = 0;
00215         error |= WASHER_WashingWaterState();
00216     } else {
00218     }
00219
00221     if(ERROR_NO != error) {
00222         WASHER_HandleError(error);
00223     } else {
00225     }
00226
00227     return error;
00228 }
00229
00230 static ERROR_t WASHER_WashingWaterState(void) {
00231     ERROR_t error = ERROR_NO;
00232
00233     WASHER_Configs.state = WASHER_STATE_WASHING_WATER;
00234
00236     if(0 == timeInState) {
00238         error |= WASHER_ControlAll(MASK_WASHER);
00239     } else {
00241     }
00242     ++timeInState;
00243
00245     if(timeInState >= MAX_WASHING_WATER_DURATION) {
00246         timeInState = 0;
00247         error |= WASHER_DrainState();
00248     } else {
00250     }
00251
00253     if(ERROR_NO != error) {
00254         WASHER_HandleError(error);
00255     } else {
00257     }
00258
00259     return error;
00260 }
00261
00262 static ERROR_t WASHER_DrainState(void) {
00263     ERROR_t error = ERROR_NO;
00264     STATE_t waterLevel = LOW;
00265
00266     WASHER_Configs.state = WASHER_STATE_DRAIN_DRUM;
00267
00269     if(0 == timeInState) {
00271         error |= WASHER_ControlAll(MASK_DRAIN);
00272     } else {
00274     }
00275     ++timeInState;
00276
00278     error |= WASHER_ReadWaterLevel(&waterLevel);
00279     if(LOW == waterLevel) {
00280         timeInState = 0;
00281         error |= WASHER_EndState();
00282     } else {
```

```
00284     }
00285
00287     if(ERROR_NO != error) {
00288         WASHER_HandleError(error);
00289     } else {
00291     }
00292
00293     return error;
00294 }
00295
00296
00303 static ERROR_t WASHER_ReadSelectoreDial(STATE_t * const state) {
00304     ERROR_t error = ERROR_NO;
00305
00307     *state = (STATE_t)selectorPin;
00308
00310     if(WASHER_Configs.activeState.selectorCotton == ACTIVE_LOW) {
00312         *state == HIGH? (*state = LOW): (*state = HIGH);
00313     } else {
00315     }
00316
00318     if(ERROR_NO != error) {
00319         WASHER_HandleError(error);
00320     } else {
00322     }
00323
00324     return error;
00325 }
00326
00327
00334 static ERROR_t WASHER_ReadStartSwitch(STATE_t * const state) {
00335     ERROR_t error = ERROR_NO;
00336
00338     *state = (STATE_t)startPin;
00339
00341     if(WASHER_Configs.activeState.start == ACTIVE_LOW) {
00343         *state == HIGH? (*state = LOW): (*state = HIGH);
00344     } else {
00346     }
00347
00349     if(ERROR_NO != error) {
00350         WASHER_HandleError(error);
00351     } else {
00353     }
00354
00355     return error;
00356 }
00357
00358
00366 static ERROR_t WASHER_ReadWaterLevel(STATE_t * const state) {
00367     ERROR_t error = ERROR_NO;
00368     bit highLevel = 0, lowLevel = 0;
00369
00371     highLevel = highLevelPin;
00372     lowLevel  = lowLevelPin;
00373
00375     if(WASHER_Configs.activeState.level == ACTIVE_LOW) {
00377         highLevel = ! highLevel;
00378         lowLevel  = ! lowLevel;
00379     } else {
00381     }
00382
00383     if(highLevel && lowLevel) {
00384         *state = HIGH;
00385     } else if( (!highLevel) && (!lowLevel) ) {
00386         *state = LOW;
00387     } else if( (!highLevel) && lowLevel) {
00388         *state = NORMAL;
00389     } else {
00390         error |= ERROR_OUT_OF_RANGE;
00391     }
00392
00393     return error;
00394 }
00395
00396
00398 static ERROR_t WASHER_ReadTemperature(STATE_t * const state) {
00399     ERROR_t error = ERROR_NO;
00400     u8 temperature = 0;
00401
00403     temperature = temperaturePin;
00404
00408     if(temperature == HIGH) {
00409         *state = HIGH;
00410     } else {
00411         *state = LOW;
00412     }
```

```
00413
00415     if(ERROR_NO != error) {
00416         WASHER_HandleError(error);
00417     } else {
00419     }
00420
00421     return error;
00422 }
00423
00424
00431 static ERROR_t WASHER_ControlDetergentHatch(const STATE_t state) {
00432     ERROR_t error = ERROR_NO;
00433
00434     #if 1
00435     switch(state) {
00436         case LOW:
00437             detergentPin = ! WASHER_Configs.activeState.detergent;
00438             break;
00439         case HIGH:
00440             detergentPin = WASHER_Configs.activeState.detergent;
00441             break;
00442         default:
00443             error |= ERROR_ILLEGAL_PARAM;
00444             break;
00445     }
00446     #elif 1
00449     detergentPin = ( (!state)   && (!WASHER_Configs.activeState.detergent) ) ||
00450                    (  state     &&   WASHER_Configs.activeState.detergent  )    ;
00451     #endif
00452
00454     if(ERROR_NO != error) {
00455         WASHER_HandleError(error);
00456     } else {
00458     }
00459
00460     return error;
00461 }
00462
00463
00470 static ERROR_t WASHER_ControlDoorLock(const STATE_t state) {
00471     ERROR_t error = ERROR_NO;
00472
00473     #if 1
00474     switch(state) {
00475         case LOW:
00476             doorLockPin = ! WASHER_Configs.activeState.doorLock;
00477             break;
00478         case HIGH:
00479             doorLockPin = WASHER_Configs.activeState.doorLock;
00480             break;
00481         default:
00482             error |= ERROR_ILLEGAL_PARAM;
00483             break;
00484     }
00485     #elif 1
00488     doorLockPin = ( (!state)   && (!WASHER_Configs.activeState.doorLock) ) ||
00489                   (  state     &&   WASHER_Configs.activeState.doorLock  )    ;
00490     #endif
00491
00493     if(ERROR_NO != error) {
00494         WASHER_HandleError(error);
00495     } else {
00497     }
00498
00499     return error;
00500 }
00501
00502
00509 static ERROR_t WASHER_ControlWasher(const STATE_t state) {
00510     ERROR_t error = ERROR_NO;
00511
00512     #if 1
00513     switch(state) {
00514         case LOW:
00515             washerPin = ! WASHER_Configs.activeState.washer;
00516             break;
00517         case HIGH:
00518             washerPin = WASHER_Configs.activeState.washer;
00519             break;
00520         default:
00521             error |= ERROR_ILLEGAL_PARAM;
00522             break;
00523     }
00524     #elif 1
00527     washerPin = ( (!state)   && (!WASHER_Configs.activeState.washer) ) ||
00528                   (  state     &&   WASHER_Configs.activeState.washer  )    ;
00529     #endif
```

```
00530
00532     if(ERROR_NO != error) {
00533         WASHER_HandleError(error);
00534     } else {
00536     }
00537
00538     return error;
00539 }
00540
00541
00548 static ERROR_t WASHER_ControlDrain(const STATE_t state) {
00549     ERROR_t error = ERROR_NO;
00550
00551     #if 1
00552     switch(state) {
00553         case LOW:
00554             drainPin = ! WASHER_Configs.activeState.drain;
00555             break;
00556         case HIGH:
00557             drainPin = WASHER_Configs.activeState.drain;
00558             break;
00559         default:
00560             error |= ERROR_ILLEGAL_PARAM;
00561             break;
00562     }
00563     #elif 1
00566     drainPin = ( (!state)   && (!WASHER_Configs.activeState.drain) ) ||
00567                (  state     &&   WASHER_Configs.activeState.drain  )    ;
00568     #endif
00569
00571     if(ERROR_NO != error) {
00572         WASHER_HandleError(error);
00573     } else {
00575     }
00576
00577     return error;
00578 }
00579
00580
00587 static ERROR_t WASHER_ControlHeater(const STATE_t state) {
00588     ERROR_t error = ERROR_NO;
00589
00590     #if 1
00591     switch(state) {
00592         case LOW:
00593             heaterPin = ! WASHER_Configs.activeState.heater;
00594             break;
00595         case HIGH:
00596             heaterPin = WASHER_Configs.activeState.heater;
00597             break;
00598         default:
00599             error |= ERROR_ILLEGAL_PARAM;
00600             break;
00601     }
00602     #elif 1
00605     heaterPin = ( (!state)   && (!WASHER_Configs.activeState.heater) ) ||
00606                 (  state     &&   WASHER_Configs.activeState.heater  )    ;
00607     #endif
00608
00610     if(ERROR_NO != error) {
00611         WASHER_HandleError(error);
00612     } else {
00614     }
00615
00616     return error;
00617 }
00618
00619
00626 static ERROR_t WASHER_ControlValve(const STATE_t state) {
00627     ERROR_t error = ERROR_NO;
00628
00629     #if 1
00630     switch(state) {
00631         case LOW:
00632             valvePin = ! WASHER_Configs.activeState.valve;
00633             break;
00634         case HIGH:
00635             valvePin = WASHER_Configs.activeState.valve;
00636             break;
00637         default:
00638             error |= ERROR_ILLEGAL_PARAM;
00639             break;
00640     }
00641     #elif 1
00644     valvePin = ( (!state)   && (!WASHER_Configs.activeState.valve) ) ||
00645                (  state     &&   WASHER_Configs.activeState.valve  )    ;
00646     #endif
```

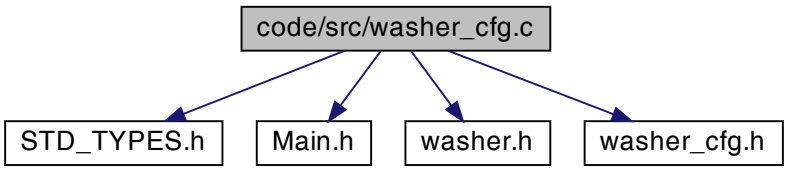
```
00647
00649     if(ERROR_NO != error) {
00650         WASHER_HandleError(error);
00651     } else {
00653     }
00654
00655     return error;
00656 }
00657
00658
00662 static void WASHER_HandleError(ERROR_t error) {
00663     switch(error) {
00664         case ERROR_NO:
00666             break;
00667         case ERROR_ILLEGAL_PARAM:
00669             break;
00670         case ERROR_INIT_FAIL:
00672             break;
00673         case ERROR_NULL_POINTER:
00675             break;
00676         case ERROR_TIMEOUT:
00678             break;
00679         case ERROR_BUSY:
00681             break;
00682         case ERROR_NOT_INITIALIZED:
00684             break;
00685         case ERROR_OUT_OF_RANGE:
00687             break;
00688         default:
00690             break;
00691     }
00692 }
```

4.21 code/src/washer_cfg.c File Reference

Configurations of Washer MAchine System.

```
#include "STD_TYPES.h"
#include "Main.h"
#include "washer.h"
#include "washer_cfg.h"
```

Include dependency graph for washer_cfg.c:



Variables

- WASHER_CONFIGS_t WASHER_Configs

  Initial configuration of the washer system.

4.21.1 Detailed Description

Configurations of Washer MAchine System.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-22

Copyright

Definition in file washer_cfg.c.

4.21.2 Variable Documentation

4.21.2.1 WASHER_Configs WASHER_CONFIGS_t WASHER_Configs

Initial value:

```
= {
    WASHER_STATE_INIT,
    25,
    {
        ACTIVE_HIGH,
        ACTIVE_HIGH,
        ACTIVE_HIGH,
        ACTIVE_HIGH,
        ACTIVE_HIGH,
        ACTIVE_HIGH,
        ACTIVE_HIGH,
        ACTIVE_HIGH,
        ACTIVE_HIGH,
    }
}
```

Initial configuration of the washer system.

Definition at line 21 of file washer_cfg.c.

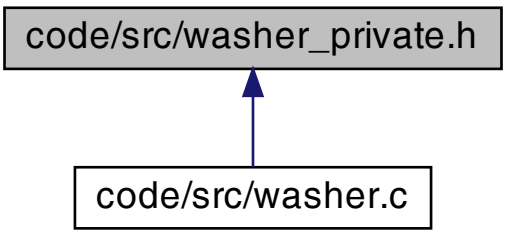
4.22 washer_cfg.c

```
00001
00009 #include "STD_TYPES.h"
00010 #include "Main.h"
00011 #include "washer.h"
00012 #include "washer_cfg.h"
00013
00014
00015 /*-------------------------------------------------------------------------*/
00016 /*                YOU CAN CHANGE THE FOLLOWING PARAMETERS                  */
00017 /*-------------------------------------------------------------------------*/
00018
00021 WASHER_CONFIGS_t WASHER_Configs = {
00022     WASHER_STATE_INIT,
00023     25,
00024     {
00025         ACTIVE_HIGH,
00026         ACTIVE_HIGH,
00027         ACTIVE_HIGH,
00029         ACTIVE_HIGH,
00030         ACTIVE_HIGH,
00031         ACTIVE_HIGH,
00032         ACTIVE_HIGH,
00033         ACTIVE_HIGH,
00034         ACTIVE_HIGH,
00035     }
00036 };
```

4.23 code/src/washer_private.h File Reference

This is a private header file for the washer class.

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



Macros

- #define ALL_OFF ((u8)0x00)
- #define MASK_DRAIN ((u8)0x01)
- #define MASK_HEATER ((u8)0x02)
- #define MASK_VALVE ((u8)0x04)
- #define MASK_DOOR_LOCK ((u8)0x08)
- #define MASK_DETERGENT ((u8)0x10)
- #define MASK_WASHER ((u8)0x20)
- #define MAX_FILL_DRUM_DURATION ((u16)1000)
- #define MAX_FILL_DETERGENT_DURATION ((u16)1000)
- #define MAX_HEATING_WATER_DURATION ((u16)1000)
- #define MAX_WASHING_WATER_DURATION ((u16)5)
- #define MAX_DRAIN_DRUM_DURATION ((u16)1000)

4.23.1 Detailed Description

This is a private header file for the washer class.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-22

Copyright

Copyright (c) 2022

Definition in file washer_private.h.

4.23.2 Macro Definition Documentation

4.23.2.1 ALL_OFF `#define ALL_OFF ( (u8)0x00 )`

Definition at line 17 of file washer_private.h.

4.23.2.2 MASK_DETERGENT `#define MASK_DETERGENT ( (u8)0x10 )`

Definition at line 22 of file washer_private.h.

4.23.2.3 MASK_DOOR_LOCK `#define MASK_DOOR_LOCK ( (u8)0x08 )`

Definition at line 21 of file washer_private.h.

4.23.2.4 MASK_DRAIN `#define MASK_DRAIN ( (u8)0x01 )`

Definition at line 18 of file washer_private.h.

4.23.2.5 MASK_HEATER `#define MASK_HEATER ( (u8)0x02 )`

Definition at line 19 of file washer_private.h.

4.23.2.6 MASK_VALVE `#define MASK_VALVE ( (u8)0x04 )`

Definition at line 20 of file washer_private.h.

4.23.2.7 MASK_WASHER `#define MASK_WASHER ( (u8)0x20 )`

Maximum duration of filling drum state in Seconds

Definition at line 25 of file washer_private.h.

4.23.2.8 MAX_DRAIN_DRUM_DURATION `#define MAX_DRAIN_DRUM_DURATION ((u16)1000)`

Definition at line 34 of file washer_private.h.

4.23.2.9 MAX_FILL_DETERGENT_DURATION `#define MAX_FILL_DETERGENT_DURATION ((u16)1000)`

Maximum duration of heating water state in Seconds

Definition at line 29 of file washer_private.h.

4.23.2.10 MAX_FILL_DRUM_DURATION `#define MAX_FILL_DRUM_DURATION ((u16)1000)`

Maximum duration of filling detergent state in Seconds

Definition at line 27 of file washer_private.h.

4.23.2.11 MAX_HEATING_WATER_DURATION `#define MAX_HEATING_WATER_DURATION ((u16)1000)`

Maximum duration of washing water state in Seconds

Definition at line 31 of file washer_private.h.

4.23.2.12 MAX_WASHING_WATER_DURATION `#define MAX_WASHING_WATER_DURATION ((u16)5)`

5 seconds, for testing Maximum duration of draining drum state in Seconds

Definition at line 33 of file washer_private.h.

4.24 washer_private.h

```
00001
00009 #ifndef WASHER_PRIVATE_H
00010 #define WASHER_PRIVATE_H
00011
00012 /*--------------------------------------------------------------------*/
00013 /* PRIVATE DATA                                                       */
00014 /*--------------------------------------------------------------------*/
00015
00016 /* Constants used by WASHER_ControlAll() */
00017 #define ALL_OFF             ( (u8)0x00 )
00018 #define MASK_DRAIN          ( (u8)0x01 )
00019 #define MASK_HEATER         ( (u8)0x02 )
00020 #define MASK_VALVE          ( (u8)0x04 )
00021 #define MASK_DOOR_LOCK      ( (u8)0x08 )
00022 #define MASK_DETERGENT      ( (u8)0x10 )
00023 #define MASK_WASHER         ( (u8)0x20 )
00024
00026 #define MAX_FILL_DRUM_DURATION      ((u16)1000)
00028 #define MAX_FILL_DETERGENT_DURATION ((u16)1000)
00030 #define MAX_HEATING_WATER_DURATION  ((u16)1000)
00032 #define MAX_WASHING_WATER_DURATION  ((u16)5)
00034 #define MAX_DRAIN_DRUM_DURATION     ((u16)1000)
00035
00036
00037 /*--------------------------------------------------------------------*/
00038 /*  PRIVATE FUNCTIONS PROTOTYPES                                      */
00039 /*--------------------------------------------------------------------*/
00040 /* The following functions represent the states of the washer system */
00041 static ERROR_t WASHER_EndState(void);
00042 static ERROR_t WASHER_InitState(void);
00043 static ERROR_t WASHER_FillDrumState(void);
00044 static ERROR_t WASHER_HeatingWaterState(void);
00045 static ERROR_t WASHER_WashingWaterState(void);
00046 static ERROR_t WASHER_DrainState(void);
00047
00048 /* The following functions handle the input events of the washer system */
00049 static ERROR_t WASHER_ReadSelectoreDial(STATE_t * const);
00050 static ERROR_t WASHER_ReadStartSwitch(STATE_t * const);
00051 static ERROR_t WASHER_ReadWaterLevel(STATE_t * const);
00052 static ERROR_t WASHER_ReadTemperature(STATE_t * const);
00053
00054 /* The following functions represent the actions of the washer system */
00055 static ERROR_t WASHER_ControlDetergentHatch(const STATE_t);
00056 static ERROR_t WASHER_ControlDoorLock(const STATE_t);
00057 static ERROR_t WASHER_ControlWasher(const STATE_t);
00058 static ERROR_t WASHER_ControlDrain(const STATE_t);
00059 static ERROR_t WASHER_ControlHeater(const STATE_t);
00060 static ERROR_t WASHER_ControlValve(const STATE_t);
00061
00062 /* The following functions handle the error events of the washer system */
00063 static void WASHER_HandleError(ERROR_t error);
00064
00065
00066
00067 #endif
```

Index

ACTIVATION_STATUS_t
    STD_TYPES.h, 26
ACTIVE_HIGH
    STD_TYPES.h, 26
ACTIVE_LOW
    STD_TYPES.h, 26
activeState
    WASHER_CONFIGS_t, 6
ALL_OFF
    washer_private.h, 54

BIT_IS_CLEAR
    BIT_MATH.h, 7
BIT_IS_SET
    BIT_MATH.h, 8
BIT_MATH.h
    BIT_IS_CLEAR, 7
    BIT_IS_SET, 8
    CLR_BIT, 8
    CONCAT_2BITS, 9
    CONCAT_3BITS, 9
    CONCAT_4BITS, 9
    CONCAT_5BITS, 9
    CONCAT_6BITS, 9
    CONCAT_7BITS, 9
    CONCAT_8BITS, 10
    GET_BIT, 10
    SET_BIT, 10
    TOG_BIT, 11
BOOL_t
    STD_TYPES.h, 26

callback
    cuteOS_TASK_t, 3
CLR_BIT
    BIT_MATH.h, 8
code/include/BIT_MATH.h, 6, 11
code/include/cuteOS.h, 12, 18
code/include/main.h, 19, 20
code/include/port.h, 21, 23
code/include/STD_TYPES.h, 23, 27
code/include/washer.h, 28, 30
code/include/washer_cfg.h, 30, 32
code/src/cuteOS.c, 33, 39
code/src/main.c, 41, 43
code/src/washer.c, 43, 45
code/src/washer_cfg.c, 51, 52
code/src/washer_private.h, 53, 56
CONCAT_2BITS
    BIT_MATH.h, 9
CONCAT_3BITS
    BIT_MATH.h, 9
CONCAT_4BITS
    BIT_MATH.h, 9
CONCAT_5BITS
    BIT_MATH.h, 9
CONCAT_6BITS
    BIT_MATH.h, 9
CONCAT_7BITS
    BIT_MATH.h, 9
CONCAT_8BITS
    BIT_MATH.h, 10
cuteOS.c
    cuteOS_GetTickTime, 35
    cuteOS_Init, 35
    cuteOS_SetTickTime, 36
    cuteOS_Start, 37
    cuteOS_TaskCreate, 37
    cuteOS_TaskRemove, 38
    MAX_TASKS_NUM, 34
    MAX_TICK_TIME_MS, 34
    tasks, 38
cuteOS.h
    cuteOS_GetTickTime, 13
    cuteOS_Init, 13
    cuteOS_SetCallback, 15
    cuteOS_SetTickTime, 15
    cuteOS_Start, 16
    cuteOS_TaskCreate, 16
    cuteOS_TaskRemove, 17
cuteOS_GetTickTime
    cuteOS.c, 35
    cuteOS.h, 13
cuteOS_Init
    cuteOS.c, 35
    cuteOS.h, 13
cuteOS_SetCallback
    cuteOS.h, 15
cuteOS_SetTickTime
    cuteOS.c, 36
    cuteOS.h, 15
cuteOS_Start
    cuteOS.c, 37
    cuteOS.h, 16
cuteOS_TASK_t, 2
    callback, 3
    id, 3
    ticks, 3
cuteOS_TaskCreate
    cuteOS.c, 37
    cuteOS.h, 16
cuteOS_TaskRemove
    cuteOS.c, 38
    cuteOS.h, 17

detergent
    DEVICES_ACTIVE_STATE_t, 4
detergentPin
    port.h, 22
DEVICES_ACTIVE_STATE_t, 3
    detergent, 4

doorLock, 4
drain, 4
heater, 4
level, 4
selectorCotton, 4
start, 4
valve, 5
washer, 5
doorLock
DEVICES_ACTIVE_STATE_t, 4
doorLockPin
port.h, 22
drain
DEVICES_ACTIVE_STATE_t, 4
drainPin
port.h, 22

ERROR_BUSY
STD_TYPES.h, 27
ERROR_ILLEGAL_PARAM
STD_TYPES.h, 27
ERROR_INIT_FAIL
STD_TYPES.h, 27
ERROR_NO
STD_TYPES.h, 27
ERROR_NOT_INITIALIZED
STD_TYPES.h, 27
ERROR_NULL_POINTER
STD_TYPES.h, 27
ERROR_OUT_OF_RANGE
STD_TYPES.h, 27
ERROR_t
STD_TYPES.h, 27
ERROR_TIMEOUT
STD_TYPES.h, 27
ERROR_YES
STD_TYPES.h, 27

f32
STD_TYPES.h, 25
f64
STD_TYPES.h, 25
FALSE
STD_TYPES.h, 26

GET_BIT
BIT_MATH.h, 10

heater
DEVICES_ACTIVE_STATE_t, 4
heaterPin
port.h, 22
HIGH
STD_TYPES.h, 27
highLevelPin
port.h, 22

id
cuteOS_TASK_t, 3

INTERRUPT_Timer_0_Overflow
main.h, 19
INTERRUPT_Timer_1_Overflow
main.h, 20
INTERRUPT_Timer_2_Overflow
main.h, 20

level
DEVICES_ACTIVE_STATE_t, 4
LOW
STD_TYPES.h, 27
lowLevelPin
port.h, 22

main
main.c, 42
main.c
main, 42
main.h
INTERRUPT_Timer_0_Overflow, 19
INTERRUPT_Timer_1_Overflow, 20
INTERRUPT_Timer_2_Overflow, 20
OSC_FREQ, 20
OSC_PER_INST, 20
MASK_DETERGENT
washer_private.h, 54
MASK_DOOR_LOCK
washer_private.h, 54
MASK_DRAIN
washer_private.h, 54
MASK_HEATER
washer_private.h, 54
MASK_VALVE
washer_private.h, 54
MASK_WASHER
washer_private.h, 54
MAX_DRAIN_DRUM_DURATION
washer_private.h, 54
MAX_FILL_DETERGENT_DURATION
washer_private.h, 55
MAX_FILL_DRUM_DURATION
washer_private.h, 55
MAX_HEATING_WATER_DURATION
washer_private.h, 55
MAX_TASKS_NUM
cuteOS.c, 34
MAX_TICK_TIME_MS
cuteOS.c, 34
MAX_WASHING_WATER_DURATION
washer_private.h, 55

NORMAL
STD_TYPES.h, 27
NULL
STD_TYPES.h, 24
NULL_BYTE
STD_TYPES.h, 25

OSC_FREQ

main.h, 20
OSC_PER_INST
main.h, 20

port.h
detergentPin, 22
doorLockPin, 22
drainPin, 22
heaterPin, 22
highLevelPin, 22
lowLevelPin, 22
selectorPin, 22
startPin, 22
temperaturePin, 23
valvePin, 23
washerPin, 23

s16
STD_TYPES.h, 25
s32
STD_TYPES.h, 25
s8
STD_TYPES.h, 25
selectorCotton
DEVICES_ACTIVE_STATE_t, 4
selectorPin
port.h, 22
SET_BIT
BIT_MATH.h, 10
size_t
STD_TYPES.h, 25
start
DEVICES_ACTIVE_STATE_t, 4
startPin
port.h, 22
state
WASHER_CONFIGS_t, 6
STATE_t
STD_TYPES.h, 27
STD_TYPES.h
ACTIVATION_STATUS_t, 26
ACTIVE_HIGH, 26
ACTIVE_LOW, 26
BOOL_t, 26
ERROR_BUSY, 27
ERROR_ILLEGAL_PARAM, 27
ERROR_INIT_FAIL, 27
ERROR_NO, 27
ERROR_NOT_INITIALIZED, 27
ERROR_NULL_POINTER, 27
ERROR_OUT_OF_RANGE, 27
ERROR_t, 27
ERROR_TIMEOUT, 27
ERROR_YES, 27
f32, 25
f64, 25
FALSE, 26
HIGH, 27
LOW, 27
NORMAL, 27
NULL, 24
NULL_BYTE, 25
s16, 25
s32, 25
s8, 25
size_t, 25
STATE_t, 27
TRUE, 26
u16, 26
u32, 26
u8, 26

tasks
cuteOS.c, 38
temperaturePin
port.h, 23
thresholdTemperature
WASHER_CONFIGS_t, 6
ticks
cuteOS_TASK_t, 3
TOG_BIT
BIT_MATH.h, 11
TRUE
STD_TYPES.h, 26

u16
STD_TYPES.h, 26
u32
STD_TYPES.h, 26
u8
STD_TYPES.h, 26

valve
DEVICES_ACTIVE_STATE_t, 5
valvePin
port.h, 23

washer
DEVICES_ACTIVE_STATE_t, 5
washer.c
WASHER_Init, 44
WASHER_Update, 44
washer.h
WASHER_Init, 29
WASHER_Update, 29
washer_cfg.c
WASHER_Configs, 52
washer_cfg.h
WASHER_Configs, 32
WASHER_STATE_DRAIN_DRUM, 32
WASHER_STATE_END, 32
WASHER_STATE_FILL_DRUM, 32
WASHER_STATE_HEATING_WATER, 32
WASHER_STATE_INIT, 32
WASHER_STATE_t, 31
WASHER_STATE_WASHING_WATER, 32
WASHER_Configs
washer_cfg.c, 52

washer_cfg.h, 32
WASHER_CONFIGS_t, 5
activeState, 6
state, 6
thresholdTemperature, 6
WASHER_Init
washer.c, 44
washer.h, 29
washer_private.h
ALL_OFF, 54
MASK_DETERGENT, 54
MASK_DOOR_LOCK, 54
MASK_DRAIN, 54
MASK_HEATER, 54
MASK_VALVE, 54
MASK_WASHER, 54
MAX_DRAIN_DRUM_DURATION, 54
MAX_FILL_DETERGENT_DURATION, 55
MAX_FILL_DRUM_DURATION, 55
MAX_HEATING_WATER_DURATION, 55
MAX_WASHING_WATER_DURATION, 55
WASHER_STATE_DRAIN_DRUM
washer_cfg.h, 32
WASHER_STATE_END
washer_cfg.h, 32
WASHER_STATE_FILL_DRUM
washer_cfg.h, 32
WASHER_STATE_HEATING_WATER
washer_cfg.h, 32
WASHER_STATE_INIT
washer_cfg.h, 32
WASHER_STATE_t
washer_cfg.h, 31
WASHER_STATE_WASHING_WATER
washer_cfg.h, 32
WASHER_Update
washer.c, 44
washer.h, 29
washerPin
port.h, 23