Animatronic Dinosaur

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.

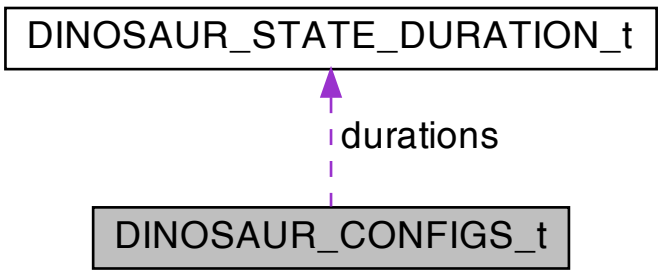
The documentation for this struct was generated from the following file:

- code/src/cuteOS.c

3.2 DINOSAUR_CONFIGS_t Struct Reference

`#include <dinosaur_cfg.h>`

Collaboration diagram for DINOSAUR_CONFIGS_t:



Data Fields

- DINOSAUR_STATE_t state
- DINOSAUR_STATE_DURATION_t durations

3.2.1 Detailed Description

Definition at line 32 of file dinosaur_cfg.h.

3.2.2 Field Documentation

3.2.2.1 durations `DINOSAUR_STATE_DURATION_t durations`

Definition at line 34 of file dinosaur_cfg.h.

3.2.2.2 state `DINOSAUR_STATE_t state`

Definition at line 33 of file dinosaur_cfg.h.

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

- code/include/dinosaur_cfg.h

3.3 DINOSAUR_STATE_DURATION_t Struct Reference

`#include <dinosaur_cfg.h>`

Data Fields

- u16 sleeping
- u16 waking
- u16 growling
- u16 attacking

3.3.1 Detailed Description

Definition at line 24 of file dinosaur_cfg.h.

3.3.2 Field Documentation

3.3.2.1 attacking `u16 attacking`

Definition at line 28 of file dinosaur_cfg.h.

3.3.2.2 growling `u16 growling`

Definition at line 27 of file dinosaur_cfg.h.

3.3.2.3 sleeping `u16 sleeping`

Definition at line 25 of file dinosaur_cfg.h.

3.3.2.4 waking `u16 waking`

Definition at line 26 of file dinosaur_cfg.h.

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

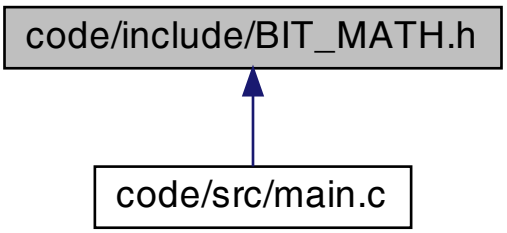
- code/include/dinosaur_cfg.h

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

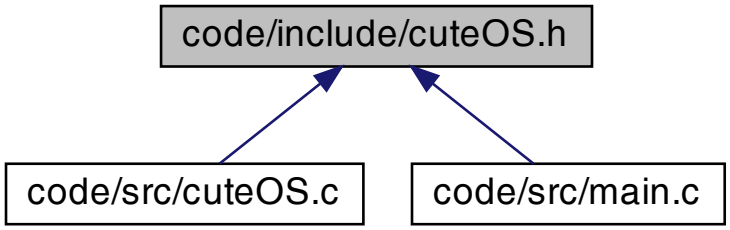
```
00001
00008 #ifndef BIT_MATH_H
00009 #define BIT_MATH_H
00010
00011
00019 #define GET_BIT(REGISTER, BIT)      ( 1 & ( (REGISTER) » (BIT) ) )
00020
00021
00028 #define SET_BIT(REGISTER, BIT)      ( (REGISTER) |= (1 « (BIT)) )
00029
00030
00037 #define CLR_BIT(REGISTER, BIT)      ( (REGISTER) &= ~(1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

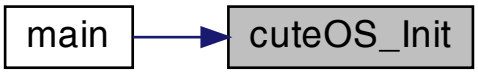
< Enable Timer 2 interrupt

< 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

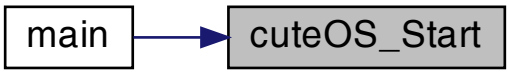
The next clock tick will return the processor to the normal operating state.

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.

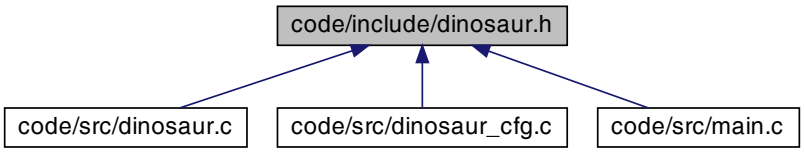
4.4 cuteOS.h

```
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/dinosaur.h File Reference

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

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



Functions

- ERROR_t DINOSAUR_Init (void)

  Initialize the dinosaur system according to the configurations in the DINOSAUR_CONFIGS_t structure.
- ERROR_t DINOSAUR_Update (void)

4.5.1 Detailed Description

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

4.5.2 Function Documentation

4.5.2.1 DINOSAUR_Init() ERROR_t DINOSAUR_Init (void)

Initialize the dinosaur system according to the configurations in the DINOSAUR_CONFIGS_t structure.

Returns

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

< Enter the default state

Definition at line 33 of file dinosaur.c.

Here is the caller graph for this function:

4.5.2.2 DINOSAUR_Update() ERROR_t DINOSAUR_Update (void)

Definition at line 59 of file dinosaur.c.

Here is the caller graph for this function:

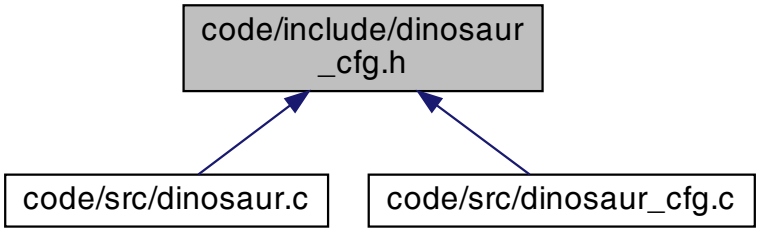
4.6 dinosaur.h

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

4.7 code/include/dinosaur_cfg.h File Reference

Traffic Light System interfaces header file. See dinosaur.c for more details.

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



Data Structures

- struct DINOSAUR_STATE_DURATION_t
- struct DINOSAUR_CONFIGS_t

Enumerations

- enum DINOSAUR_STATE_t { DINOSAUR_SLEEPING , DINOSAUR_WAKING , DINOSAUR_GROWLING , DINOSAUR_ATTACKING }

Variables

- DINOSAUR_CONFIGS_t DINOSAUR_Configs

  Initial configuration of the dinosaur system.

4.7.1 Detailed Description

Traffic Light System interfaces header file. See dinosaur.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 dinosaur_cfg.h.

4.7.2 Enumeration Type Documentation

4.7.2.1 DINOSAUR_STATE_t enum DINOSAUR_STATE_t

< Dinosaur states

Enumerator

DINOSAUR_SLEEPING	
DINOSAUR_WAKING	
DINOSAUR_GROWLING	
DINOSAUR_ATTACKING	

Definition at line 16 of file dinosaur_cfg.h.

4.7.3 Variable Documentation

4.7.3.1 DINOSAUR_Configs DINOSAUR_CONFIGS_t DINOSAUR_Configs [extern]

Initial configuration of the dinosaur system.

Definition at line 32 of file dinosaur_cfg.c.

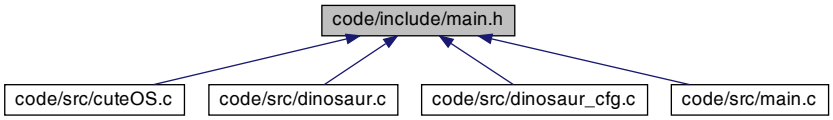
4.8 dinosaur_cfg.h

```
00001
00009 #ifndef DINOSAUR_CFG_H
00010 #define DINOSAUR_CFG_H
00011
00012 /*-----------------------------------------------------------------------*/
00013 /*  PRIVATE TYPE DEFINITIONS                                             */
00014 /*-----------------------------------------------------------------------*/
00016 typedef enum{
00017     DINOSAUR_SLEEPING,
00018     DINOSAUR_WAKING,
00019     DINOSAUR_GROWLING,
00020     DINOSAUR_ATTACKING,
00021 } DINOSAUR_STATE_t;
00022
00024 typedef struct {
00025     u16 sleeping;
00026     u16 waking;
00027     u16 growling;
00028     u16 attacking;
00029 } DINOSAUR_STATE_DURATION_t;
00030
00032 typedef struct {
00033     DINOSAUR_STATE_t state;
00034     DINOSAUR_STATE_DURATION_t durations;
00035 }DINOSAUR_CONFIGS_t;
00036
00037
00038
00039
00040
00041
00042 /*-----------------------------------------------------------------------*/
00043 /*                YOU MUST «<NOT»> CHANGE THE FOLLOWING PARAMETERS       */
00044 /*-----------------------------------------------------------------------*/
00045 extern DINOSAUR_CONFIGS_t DINOSAUR_Configs;
00046
00047
00048 #endif /* DINOSAUR_CFG_H */
```

4.9 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

4.9.1 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.

4.9.2 Macro Definition Documentation

4.9.2.1 INTERRUPT_Timer_0_Overflow `#define INTERRUPT_Timer_0_Overflow 1`

Definition at line 36 of file main.h.

4.9.2.2 INTERRUPT_Timer_1_Overflow `#define INTERRUPT_Timer_1_Overflow 3`

Definition at line 37 of file main.h.

4.9.2.3 INTERRUPT_Timer_2_Overflow `#define INTERRUPT_Timer_2_Overflow 5`

Definition at line 38 of file main.h.

4.9.2.4 OSC_FREQ `#define OSC_FREQ (12000000UL)`

Definition at line 16 of file main.h.

4.9.2.5 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.

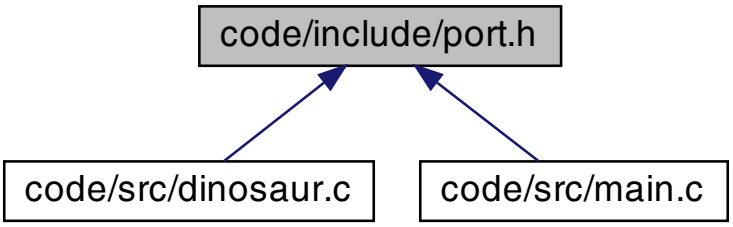
4.10 main.h

```
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 */
```

4.11 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 redPin = P1^0
- sbit amberPin = P1^1
- sbit greenPin = P1^2

4.11.1 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.

4.11.2 Variable Documentation

4.11.2.1 amberPin `sbit amberPin = P1^1`

Definition at line 16 of file port.h.

4.11.2.2 greenPin `sbit greenPin = P1^2`

Definition at line 17 of file port.h.

4.11.2.3 redPin `sbit redPin = P1^0`

In file dinosaur.C

Definition at line 15 of file port.h.

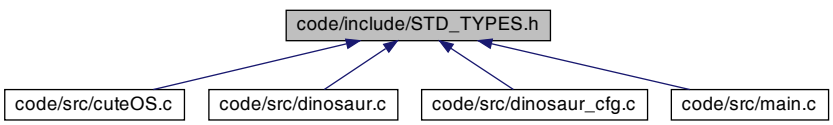
4.12 port.h

```
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 #endif  /* PORT_H */
```

4.13 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 }

4.13.1 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.

4.13.2 Macro Definition Documentation

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

NULL pointer

Definition at line 61 of file STD_TYPES.h.

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

Definition at line 64 of file STD_TYPES.h.

4.13.3 Typedef Documentation

4.13.3.1 f32 `typedef float f32`

Definition at line 22 of file STD_TYPES.h.

4.13.3.2 f64 `typedef double f64`

Definition at line 23 of file STD_TYPES.h.

4.13.3.3 s16 `typedef signed short int s16`

Definition at line 13 of file STD_TYPES.h.

4.13.3.4 s32 `typedef signed long int s32`

Definition at line 12 of file STD_TYPES.h.

4.13.3.5 s8 `typedef signed char s8`

Definition at line 14 of file STD_TYPES.h.

4.13.3.6 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.

4.13.3.7 u16 `typedef unsigned short int u16`

Definition at line 18 of file STD_TYPES.h.

4.13.3.8 u32 `typedef unsigned long int u32`

Definition at line 17 of file STD_TYPES.h.

4.13.3.9 u8 `typedef unsigned char u8`

Definition at line 19 of file STD_TYPES.h.

4.13.4 Enumeration Type Documentation

4.13.4.1 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.

4.13.4.2 BOOL_t `enum BOOL_t`

Enumerator

FALSE	
TRUE	

Definition at line 43 of file STD_TYPES.h.

4.13.4.3 ERROR_t `enum ERROR_t`

Enumerator

ERROR_NO	No error occured

Enumerator

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.

4.13.4.4 STATE_t enum STATE_t

Enumerator

LOW	
HIGH	
NORMAL	

Definition at line 31 of file STD_TYPES.h.

4.14 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;
```

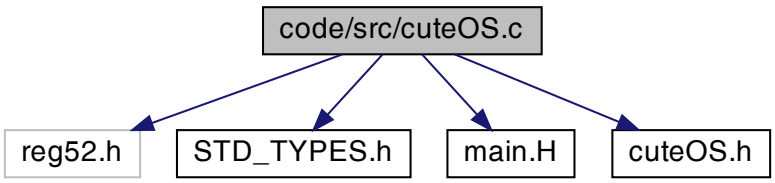
```
00047
00048 typedef enum{
00049     ERROR_NO                = 0,
00050     ERROR_YES               = 0x1,
00051     ERROR_TIMEOUT           = 0x2,
00052     ERROR_NULL_POINTER      = 0x4,
00053     ERROR_BUSY              = 0x8,
00054     ERROR_NOT_INITIALIZED   = 0x10,
00055     ERROR_ILLEGAL_PARAM     = 0x20,
00056     ERROR_OUT_OF_RANGE      = 0x40,
00057 }ERROR_t;
00058
00059 /* Pointers */
00060 #undef NULL
00061 #define NULL ((void *)0)
00063 #undef NULL_BYTE
00064 #define NULL_BYTE  ('\0')
00065
00066  #endif   /* STD_TYPES_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

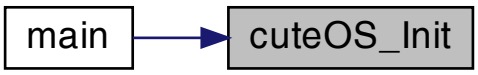
< Enable Timer 2 interrupt

< 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 )`

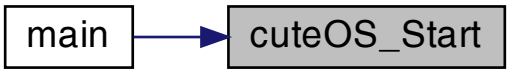
Start the Cute Embedded Operating System (cuteOS)

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
```

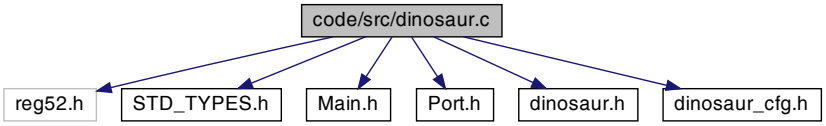
```
00218     return error;
00219 }
00220
00221
00228 ERROR_t cuteOS_Init(void) {
00229     ERROR_t error = ERROR_NO;
00230     u32 Inc;
00231     u16 Reload_16;
00232     u8 Reload_08H, Reload_08L;
00233
00234     TR2 = 0;
00236
00240     T2CON = 0x04;
00248     Inc = ((u32)cuteOS_TickTimeMs * (OSC_FREQ/1000)) / (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/dinosaur.c File Reference

This is the source file for functions used in dinosaur animation system. For more details, see main.c file description.

```
#include <reg52.h>
#include "STD_TYPES.h"
#include "Main.h"
#include "Port.h"
#include "dinosaur.h"
#include "dinosaur_cfg.h"
```

Include dependency graph for dinosaur.c:



Functions

- ERROR_t DINOSAUR_Init (void)

  Initialize the dinosaur system according to the configurations in the DINOSAUR_CONFIGS_t structure.
- ERROR_t DINOSAUR_Update (void)

Variables

- u16 timeInState = 0

4.17.1 Detailed Description

This is the source file for functions used in dinosaur animation system. For more details, see main.c file description.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-22

Copyright

Copyright (c) 2022

Definition in file dinosaur.c.

4.17.2 Function Documentation

4.17.2.1 DINOSAUR_Init() `ERROR_t DINOSAUR_Init ( void )`

Initialize the dinosaur system according to the configurations in the DINOSAUR_CONFIGS_t structure.

Returns

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

< Enter the default state

Definition at line 33 of file dinosaur.c.

Here is the caller graph for this function:

4.17.2.2 DINOSAUR_Update() ERROR_t DINOSAUR_Update (void)

Definition at line 59 of file dinosaur.c.

Here is the caller graph for this function:

4.17.3 Variable Documentation

4.17.3.1 timeInState u16 timeInState = 0

Time in the current state in seconds

Definition at line 28 of file dinosaur.c.

4.18 dinosaur.c

```
00001
00010 #include <reg52.h>
00011 #include "STD_TYPES.h"
00012 #include "Main.h"
00013 #include "Port.h"
00014 #include "dinosaur.h"
00015 #include "dinosaur_cfg.h"
00016
00017 /*-------------------------------------------------------------------*/
00018 /*  PRIVATE FUNCTIONS PROTOTYPES                                     */
00019 /*-------------------------------------------------------------------*/
00020 static ERROR_t DINOSAUR_Sleep(void);
00021 static ERROR_t DINOSAUR_Wake(void);
00022 static ERROR_t DINOSAUR_Growl(void);
00023 static ERROR_t DINOSAUR_Attack(void);
00024
00025 /*-------------------------------------------------------------------*/
00026 /* PUBLIC DATA                                                       */
00027 /*-------------------------------------------------------------------*/
00028 u16 timeInState = 0;
00030 /*-------------------------------------------------------------------*/
00031 /*                          PUBLIC FUNCTIONS                         */
00032 /*-------------------------------------------------------------------*/
00033 ERROR_t DINOSAUR_Init(void) {
00034     ERROR_t error = ERROR_NO;
00035
00037     switch(DINOSAUR_Configs.state) {
00038         case DINOSAUR_SLEEPING:
00039             error |= DINOSAUR_Sleep();
00040             break;
00041         case DINOSAUR_WAKING:
00042             error |= DINOSAUR_Wake();
00043             break;
00044         case DINOSAUR_GROWLING:
00045             error |= DINOSAUR_Growl();
00046             break;
00047         case DINOSAUR_ATTACKING:
```

```
00048             error |= DINOSAUR_Attack();
00049             break;
00050         default:
00051             error |= ERROR_ILLEGAL_PARAM;
00052             break;
00053     }
00054
00055     return error;
00056 }
00057
00058
00059 ERROR_t DINOSAUR_Update(void) {
00060     ERROR_t error = ERROR_NO;
00061
00062     switch(DINOSAUR_Configs.state) {
00063         case DINOSAUR_SLEEPING:
00064             if(timeInState < DINOSAUR_Configs.durations.sleeping - 1){
00065                 if(0 == timeInState){
00066                     error |= DINOSAUR_Sleep();
00067                 }
00068                 ++timeInState;
00069             } else if(timeInState >= DINOSAUR_Configs.durations.sleeping - 1) {
00070                 timeInState = 0;
00071                 error |= DINOSAUR_Wake();
00072             } else {
00073                 ++timeInState;
00074                 error |= ERROR_OUT_OF_RANGE;
00075             }
00076             break;
00077         case DINOSAUR_WAKING:
00078             if(timeInState < DINOSAUR_Configs.durations.waking - 1){
00079                 if(0 == timeInState){
00080                     error |= DINOSAUR_Wake();
00081                 }
00082                 ++timeInState;
00083             } else if(timeInState >= DINOSAUR_Configs.durations.waking - 1) {
00084                 timeInState = 0;
00085                 error |= DINOSAUR_Growl();
00086             } else {
00087                 ++timeInState;
00088                 error |= ERROR_OUT_OF_RANGE;
00089             }
00090             break;
00091         case DINOSAUR_GROWLING:
00092             if(timeInState < DINOSAUR_Configs.durations.growling - 1){
00093                 if(0 == timeInState){
00094                     error |= DINOSAUR_Growl();
00095                 }
00096                 ++timeInState;
00097             } else if(timeInState >= DINOSAUR_Configs.durations.growling - 1) {
00098                 timeInState = 0;
00099                 error |= DINOSAUR_Attack();
00100             } else {
00101                 ++timeInState;
00102                 error |= ERROR_OUT_OF_RANGE;
00103             }
00104             break;
00105         case DINOSAUR_ATTACKING:
00106             if(timeInState < DINOSAUR_Configs.durations.attacking - 1){
00107                 if(0 == timeInState){
00108                     error |= DINOSAUR_Attack();
00109                 }
00110                 ++timeInState;
00111             } else if(timeInState >= DINOSAUR_Configs.durations.attacking - 1) {
00112                 timeInState = 0;
00113                 error |= DINOSAUR_Sleep();
00114             } else {
00115                 ++timeInState;
00116                 error |= ERROR_OUT_OF_RANGE;
00117             }
00118             break;
00119         default:
00120             error |= ERROR_ILLEGAL_PARAM;
00121             break;
00122     }
00123
00124     return error;
00125 }
00126
00127
00128
00129
00130
00131
00132 /*-----------------------------------------------------------------------*/
00133 /*                PRIVATE FUNCTIONS DEFINITIONS                          */
00134 /*-----------------------------------------------------------------------*/
```

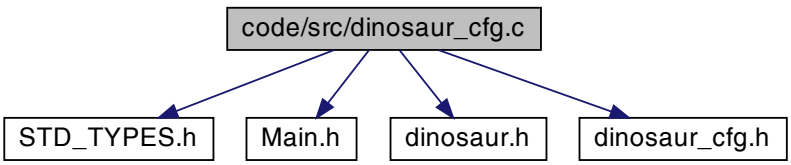
```
00135
00141 static ERROR_t DINOSAUR_Sleep(void) {
00142     ERROR_t error = ERROR_NO;
00143
00144     DINOSAUR_Configs.state = DINOSAUR_SLEEPING;
00145
00146     redPin = 1;
00147     amberPin = 1;
00148     greenPin = 1;
00149
00150
00151     return error;
00152 }
00153
00154
00160 static ERROR_t DINOSAUR_Wake(void) {
00161     ERROR_t error = ERROR_NO;
00162
00163     DINOSAUR_Configs.state = DINOSAUR_WAKING;
00164
00165     redPin = 0;
00166     amberPin = 1;
00167     greenPin = 1;
00168
00169     return error;
00170 }
00171
00172
00178 static ERROR_t DINOSAUR_Growl(void) {
00179     ERROR_t error = ERROR_NO;
00180
00181     DINOSAUR_Configs.state = DINOSAUR_GROWLING;
00182
00183     redPin = 1;
00184     amberPin = 0;
00185     greenPin = 1;
00186
00187     return error;
00188 }
00189
00190
00196 static ERROR_t DINOSAUR_Attack(void) {
00197     ERROR_t error = ERROR_NO;
00198
00199     DINOSAUR_Configs.state = DINOSAUR_ATTACKING;
00200
00201     redPin = 1;
00202     amberPin = 1;
00203     greenPin = 0;
00204
00205     return error;
00206 }
```

4.19 code/src/dinosaur_cfg.c File Reference

Configurations of Traffic Light System.

```
#include "STD_TYPES.h"
#include "Main.h"
#include "dinosaur.h"
#include "dinosaur_cfg.h"
```

Include dependency graph for dinosaur_cfg.c:



Variables

- DINOSAUR_CONFIGS_t DINOSAUR_Configs
  Initial configuration of the dinosaur system.

4.19.1 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 dinosaur_cfg.c.

4.19.2 Variable Documentation

4.19.2.1 DINOSAUR_Configs DINOSAUR_CONFIGS_t DINOSAUR_Configs

Initial value:

```
= {
    DINOSAUR_SLEEPING,
    {
        3,
        3,
        3,
        3,
    }
}
```

Initial configuration of the dinosaur system.

Definition at line 32 of file dinosaur_cfg.c.

4.20 dinosaur_cfg.c

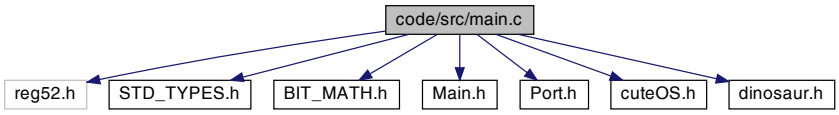
```
00001
00009 #include "STD_TYPES.h"
00010 #include "Main.h"
00011 #include "dinosaur.h"
00012 #include "dinosaur_cfg.h"
00013
00014
00015 /*-----------------------------------------------------------------------------*/
00016 /*                YOU CAN CHANGE THE FOLLOWING PARAMETERS                      */
00017 /*-----------------------------------------------------------------------------*/
00018
00021 #if 0
00022 DINOSAUR_CONFIGS_t DINOSAUR_Configs = {
00023     DINOSAUR_SLEEPING,
00024     {
00025         255,
00026         60,
00027         40,
00028         120,
00029     }
00030 };
00031 #elif 1
00032 DINOSAUR_CONFIGS_t DINOSAUR_Configs = {
00033     DINOSAUR_SLEEPING,
00034     {
00035         3,
00036         3,
00037         3,
00038         3,
00039     }
00040 };
00041 #endif
```

4.21 code/src/main.c File Reference

Animatronic dinosaur such as the one in Natural History Museum in London, UK.

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

Include dependency graph for main.c:



Functions

- void main (void)

4.21.1 Detailed Description

Animatronic dinosaur such as the one in Natural History Museum in London, UK.

Author

Mahmoud Karam (ma.karam272@gmail.com)

The Natural History Museum in London recently installed a robotic dinosaur among the fossils in their Dinosaur Gallery. This large exhibit models a tyrannosaurus rex guarding a recent kill: the robot is large, very loud and moves quickly. It has proved to be very popular with visitors.

In this project, we will try to animate some of the dinosaur's maneuvers: 1- Sleeping: The dinosaur will be largely motionless, but will be obviously 'breathing'. Irregular snoring noises, or slight movements during this time will add interest for the audience. Sleeping time is defined in SLEEPING_DURATION macro. 2- Waking: The dinosaur will begin to wake up. Eyelids will begin to flicker. Breathing will become more rapid. Waking time is defined in WAKING_DURATION macro. 3- Growling: Eyes will suddenly open, and the dinosaur will emit a very loud growl. Some further movement and growling will follow. Growling time is defined in GROWLING_DURATION macro. 4- Attacking: Rapid 'random' movements towards the audience. Lots of noise (you should be able to hear this from the next floor in the museum). Attack time is defined in ATTACK_DURATION macro.

Application usage: 1- Initialize the OS and the dinosaur.
```
cuteOS_Init();
DINOSAUR_Init();
```

2- Create the dinosaur task with 1 second period.
```
cuteOS_TaskCreate(DINOSAUR_Update, 1000);
```

3- Start the Cute OS scheduler.
```
cuteOS_Start();
```

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

Version

1.0.0

Date

2022-03-24

Copyright

Application usage:

Definition in file main.c.

4.21.2 Function Documentation

4.21.2.1 main() void main (void)

< Initialize Cute OS

< Initialize the dinosaur light system

< Create the tasks

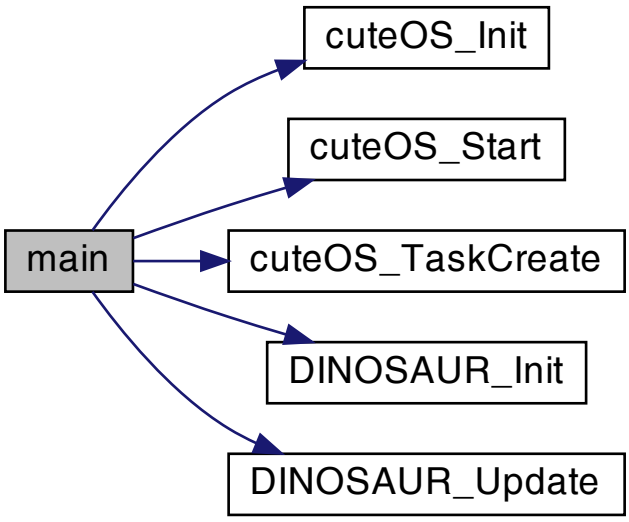
< Create a task to run the dinosaur light system

< Start the Cute OS scheduler

< The scheduler will never return from here

Definition at line 57 of file main.c.

Here is the call graph for this function:



4.22 main.c

```
00001
00049 #include <reg52.h>
00050 #include "STD_TYPES.h"
00051 #include "BIT_MATH.h"
00052 #include "Main.h"
00053 #include "Port.h"
00054 #include "cuteOS.h"
00055 #include "dinosaur.h"
00056
00057 void main(void) {
00058     /* Initialize the system */
00059     cuteOS_Init();
00060     DINOSAUR_Init();
00063     cuteOS_TaskCreate(DINOSAUR_Update, 1000);
00065     cuteOS_Start();
00068     while(1);
00069 }
```

Index