

OpenSwarm

0.16.1.2

Generated by Doxygen 1.8.9.1

Wed Jan 27 2016 16:03:18

Contents

1	OpenSwarm documentation	1
1.1	Introduction	1
1.2	Links	1
1.3	License	1
1.4	Thanks	2
2	Todo List	2
3	Module Index	2
3.1	Modules	2
4	Data Structure Index	3
4.1	Data Structures	3
5	File Index	4
5.1	File List	4
6	Module Documentation	6
6.1	Kernel	6
6.1.1	Detailed Description	7
6.1.2	Introduction	7
6.2	Event Management	8
6.2.1	Detailed Description	8
6.2.2	Usage	8
6.2.3	Example	8
6.2.4	License	9
6.3	I/O Management	10
6.3.1	Detailed Description	11
6.3.2	Introduction	11
6.3.3	Usage	11
6.3.4	License	11
6.4	Camera Module	12
6.4.1	Detailed Description	12
6.4.2	Usage	12
6.4.3	License	13
6.5	Shefpuck	14
6.5.1	Detailed Description	14
6.5.2	License	14
6.6	e-puck specific modules	15
6.6.1	Detailed Description	15

6.6.2	Features	16
6.6.3	License	16
6.7	I2C interface	18
6.7.1	Detailed Description	18
6.7.2	Usage	18
6.7.3	License	19
6.8	Motor Control	20
6.8.1	Detailed Description	20
6.8.2	Usage	20
6.8.3	License	20
6.9	Remote Control	21
6.9.1	Detailed Description	21
6.9.2	Usage	21
6.9.3	License	21
6.10	UART 1&2	22
6.10.1	Detailed Description	22
6.10.2	Usage	22
6.10.3	License	22
6.11	Process Management	23
6.11.1	Detailed Description	23
6.11.2	Usage	24
6.11.3	Example	25
6.11.4	License	25
7	Data Structure Documentation	26
7.1	sys_event_data Struct Reference	26
7.1.1	Detailed Description	26
7.1.2	Field Documentation	26
7.2	sys_i2c_msg Struct Reference	26
7.2.1	Detailed Description	27
7.2.2	Field Documentation	27
7.3	sys_motors Struct Reference	27
7.3.1	Detailed Description	27
7.3.2	Field Documentation	28
7.4	sys_occurred_event Struct Reference	28
7.4.1	Detailed Description	28
7.4.2	Field Documentation	28
7.5	sys_pcb Struct Reference	28
7.5.1	Detailed Description	29
7.5.2	Field Documentation	29

7.6	sys_pcb_list_element Struct Reference	30
7.6.1	Detailed Description	30
7.6.2	Field Documentation	30
7.7	sys_peh Struct Reference	31
7.7.1	Detailed Description	31
7.7.2	Field Documentation	31
7.8	sys_pIOHandler Struct Reference	32
7.8.1	Detailed Description	32
7.8.2	Field Documentation	32
7.9	sys_registered_event Struct Reference	33
7.9.1	Detailed Description	33
7.9.2	Field Documentation	33
7.10	sys_rgb_pixel Struct Reference	34
7.10.1	Detailed Description	34
7.10.2	Field Documentation	34
7.11	sys_scheduler_info Struct Reference	34
7.11.1	Detailed Description	34
7.11.2	Field Documentation	35
7.12	sys_subscribed_process Struct Reference	35
7.12.1	Detailed Description	35
7.12.2	Field Documentation	35
7.13	sys_uart_txdata Struct Reference	35
7.13.1	Detailed Description	36
7.13.2	Field Documentation	36
8	File Documentation	36
8.1	definitions.h File Reference	36
8.1.1	Detailed Description	38
8.1.2	Macro Definition Documentation	38
8.1.3	Typedef Documentation	39
8.1.4	Enumeration Type Documentation	40
8.2	events/events.c File Reference	41
8.2.1	Detailed Description	42
8.2.2	Function Documentation	42
8.2.3	Variable Documentation	44
8.3	events/events.h File Reference	45
8.3.1	Detailed Description	46
8.3.2	Typedef Documentation	46
8.3.3	Function Documentation	46
8.4	interrupts.c File Reference	48

8.4.1	Detailed Description	48
8.4.2	Function Documentation	49
8.5	interrupts.h File Reference	50
8.5.1	Detailed Description	50
8.5.2	Macro Definition Documentation	50
8.5.3	Function Documentation	51
8.6	io/io.c File Reference	52
8.6.1	Detailed Description	53
8.6.2	Function Documentation	53
8.7	io/io.h File Reference	54
8.7.1	Detailed Description	55
8.7.2	Function Documentation	56
8.8	io/io_clock.c File Reference	57
8.8.1	Detailed Description	58
8.8.2	Function Documentation	58
8.9	io/io_clock.h File Reference	59
8.9.1	Detailed Description	59
8.9.2	Function Documentation	60
8.10	memory.c File Reference	60
8.10.1	Detailed Description	61
8.10.2	Function Documentation	61
8.11	memory.h File Reference	62
8.11.1	Detailed Description	63
8.11.2	Function Documentation	63
8.12	platform/e-puck/camera.c File Reference	64
8.12.1	Detailed Description	65
8.12.2	Macro Definition Documentation	65
8.12.3	Function Documentation	67
8.13	platform/e-puck/camera.h File Reference	68
8.13.1	Detailed Description	69
8.13.2	Macro Definition Documentation	70
8.13.3	Typedef Documentation	70
8.13.4	Function Documentation	70
8.14	platform/e-puck/camera_processing.c File Reference	72
8.14.1	Detailed Description	73
8.14.2	Macro Definition Documentation	73
8.14.3	Function Documentation	74
8.14.4	Variable Documentation	74
8.15	platform/e-puck/camera_processing.h File Reference	75
8.15.1	Detailed Description	75

8.15.2	Function Documentation	75
8.16	platform/e-puck/DSPIC30F6014A_HDI.h File Reference	76
8.16.1	Detailed Description	77
8.16.2	Macro Definition Documentation	77
8.17	platform/e-puck/i2c.c File Reference	78
8.17.1	Detailed Description	79
8.17.2	Function Documentation	79
8.18	platform/e-puck/i2c.h File Reference	81
8.18.1	Detailed Description	82
8.18.2	Function Documentation	82
8.19	platform/e-puck/i2c_data.c File Reference	83
8.19.1	Detailed Description	84
8.19.2	Function Documentation	84
8.19.3	Variable Documentation	85
8.20	platform/e-puck/i2c_data.h File Reference	85
8.20.1	Detailed Description	86
8.20.2	Enumeration Type Documentation	86
8.20.3	Function Documentation	87
8.20.4	Variable Documentation	87
8.21	platform/e-puck/i2c_HDI.c File Reference	87
8.21.1	Detailed Description	88
8.21.2	Function Documentation	89
8.22	platform/e-puck/i2c_HDI.h File Reference	90
8.22.1	Detailed Description	91
8.22.2	Function Documentation	92
8.23	platform/e-puck/io_HDI.c File Reference	93
8.23.1	Detailed Description	94
8.23.2	Function Documentation	94
8.23.3	Variable Documentation	96
8.24	platform/e-puck/io_HDI.h File Reference	96
8.24.1	Detailed Description	97
8.24.2	Macro Definition Documentation	97
8.24.3	Function Documentation	98
8.24.4	Variable Documentation	98
8.25	platform/e-puck/motors.c File Reference	99
8.25.1	Detailed Description	100
8.25.2	Macro Definition Documentation	100
8.25.3	Function Documentation	100
8.26	platform/e-puck/motors.h File Reference	102
8.26.1	Detailed Description	103

8.26.2	Macro Definition Documentation	103
8.26.3	Function Documentation	104
8.27	platform/e-puck/motors_HDI.c File Reference	105
8.27.1	Detailed Description	105
8.27.2	Function Documentation	105
8.28	platform/e-puck/motors_HDI.h File Reference	106
8.28.1	Detailed Description	107
8.28.2	Macro Definition Documentation	107
8.28.3	Function Documentation	107
8.29	platform/e-puck/process_Management_HDI.c File Reference	108
8.29.1	Detailed Description	108
8.29.2	Function Documentation	109
8.30	platform/e-puck/process_Management_HDI.h File Reference	110
8.30.1	Detailed Description	111
8.30.2	Function Documentation	111
8.31	platform/e-puck/remoteControl.c File Reference	112
8.31.1	Detailed Description	112
8.31.2	Function Documentation	113
8.32	platform/e-puck/remoteControl.h File Reference	114
8.32.1	Detailed Description	116
8.32.2	Macro Definition Documentation	116
8.32.3	Function Documentation	120
8.33	platform/e-puck/remoteControl_HDI.c File Reference	121
8.33.1	Detailed Description	122
8.33.2	Function Documentation	123
8.33.3	Variable Documentation	123
8.34	platform/e-puck/remoteControl_HDI.h File Reference	124
8.34.1	Detailed Description	125
8.34.2	Macro Definition Documentation	125
8.34.3	Function Documentation	126
8.34.4	Variable Documentation	126
8.35	platform/e-puck/system_Timer_HDI.c File Reference	126
8.35.1	Detailed Description	127
8.35.2	Function Documentation	128
8.35.3	Variable Documentation	128
8.36	platform/e-puck/system_Timer_HDI.h File Reference	129
8.36.1	Detailed Description	130
8.36.2	Function Documentation	130
8.36.3	Variable Documentation	131
8.37	platform/e-puck/traps.c File Reference	131

8.37.1 Detailed Description	131
8.37.2 Function Documentation	132
8.38 platform/e-puck/uart.c File Reference	132
8.38.1 Detailed Description	133
8.38.2 Macro Definition Documentation	134
8.38.3 Function Documentation	134
8.39 platform/e-puck/uart.h File Reference	135
8.39.1 Detailed Description	136
8.39.2 Function Documentation	137
8.40 platform/e-puck/uart_HDI.c File Reference	138
8.40.1 Detailed Description	139
8.40.2 Function Documentation	140
8.40.3 Variable Documentation	141
8.41 platform/e-puck/uart_HDI.h File Reference	142
8.41.1 Detailed Description	143
8.41.2 Macro Definition Documentation	143
8.41.3 Function Documentation	144
8.41.4 Variable Documentation	145
8.42 processes/data.c File Reference	146
8.42.1 Detailed Description	147
8.42.2 Function Documentation	147
8.42.3 Variable Documentation	150
8.43 processes/data.h File Reference	150
8.43.1 Detailed Description	152
8.43.2 Function Documentation	152
8.43.3 Variable Documentation	154
8.44 processes/process_Management.c File Reference	155
8.44.1 Detailed Description	156
8.44.2 Function Documentation	156
8.45 processes/process_Management.h File Reference	161
8.45.1 Detailed Description	163
8.45.2 Macro Definition Documentation	163
8.45.3 Function Documentation	163
8.46 processes/scheduler.c File Reference	168
8.46.1 Detailed Description	168
8.46.2 Function Documentation	169
8.47 processes/scheduler.h File Reference	169
8.47.1 Detailed Description	170
8.47.2 Macro Definition Documentation	171
8.47.3 Function Documentation	171

8.48	processes/system_Timer.c File Reference	172
8.48.1	Detailed Description	172
8.48.2	Function Documentation	173
8.49	processes/system_Timer.h File Reference	174
8.49.1	Detailed Description	175
8.49.2	Function Documentation	175
8.50	system.c File Reference	176
8.50.1	Detailed Description	176
8.50.2	Function Documentation	177
8.51	system.h File Reference	178
8.51.1	Detailed Description	178
8.51.2	Macro Definition Documentation	179
8.51.3	Function Documentation	179
	Index	181

1 OpenSwarm documentation

1.1 Introduction

OpenSwarm is an easy-to-use event-driven preemptive operating system for miniature robots. It offers abstract hardware-independent functions to make user code more extendible, maintainable, and portable. The hybrid kernel provides preemptive and cooperative scheduling, asynchronous and synchronous programming models with events, and inter-process communication functions.

OpenSwarm was created during the PhD of Stefan M Trenkwalder (<http://trenkwalder.tech>) at the University of Sheffield (<http://www.sheffield.ac.uk/>) under the Supervision of Dr. Roderich Gross and Dr. Andreas Kolling.

The code of OpenSwarm can be basically divided into 3 different modules:

- [Process Management](#)
- [Event Management](#)
- [I/O Management](#) (This includes device specific sensors and actuators)

All modules are, then, combined in OpenSwarm's [Kernel](#).

1.2 Links

- <http://www.openswarm.org/> The official OpenSwarm website
- <http://trenkwalder.tech/> The academic webpage of Stefan Trenkwalder
- <http://naturalrobotics.group.shef.ac.uk/> The website of the research group
- <http://openswarm.org/license/> The link to the newest license (in case it changed)

1.3 License

LICENSE: adapted FreeBSD License (see <http://openswarm.org/license>)
 Copyright (c) 2015, Stefan M. Trenkwalder
 All rights reserved.

1.4 Thanks

OpenSwarm is part of the PhD of Stefan M. Trenkwalder (<http://trenkwalder.tech>) who is recipient of a DOC Fellowship of the Austrian Academy of Sciences (<http://www.oeaw.ac.at/>).

2 Todo List

Module **camera**

The used functions from the e-puck library are very time and computational intensive. These function need to be rewritten to decrease the processing load.

File **camera.c**

The used functions from the e-puck library are very time and computational intensive. These function can be rewritten to decrease the processing load.

Module **i2c**

testing and debugging of this module.

globalScope> Global Sys_Camera_PreProcessor (void)

rewrite the camera to computational less intensive functions

globalScope> Global Sys_Init_Camera (void)

rewrite the camera to computational less intensive functions

globalScope> Global Sys_Init_Camera (void)

rewrite the camera to computational less intensive functions

globalScope> Global Sys_Start_Camera (void)

rewrite the camera to computational less intensive functions

globalScope> Global Sys_Start_Camera (void)

rewrite the camera to computational less intensive functions

3 Module Index

3.1 Modules

Here is a list of all modules:

Kernel	6
Event Management	8
I/O Management	10
Camera Module	12
I2C interface	18
Motor Control	20

Remote Control	21
UART 1&2	22
Shfepuck	14
e-puck specific modules	15
Camera Module	12
I2C interface	18
Motor Control	20
Remote Control	21
UART 1&2	22
Process Management	23

4 Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

sys_event_data	It is a single linked list element and contains data of an occurred event	??
sys_i2c_msg	It is a single linked list element containing messages that need to be sent via I2C. This list acts as a message buffer	??
sys_motors	This struct contains the speed for a motor	??
sys_occurred_event	Linked list element containing an occurred events	??
sys_pcb	Process Control Block contains all data for a single process	??
sys_pcb_list_element	Double linked list element containing sys_process_control_block	??
sys_peh	Double linked list element of process event-handlers	??
sys_piOHandler	Linked list element containing I/O Handler function pointers	??
sys_registered_event	A single linked element containing a registered event and its subscribers	??
sys_rgb_pixel	This bitfield contains the structure of a received camera pixel	??
sys_scheduler_info	The scheduling information for each process	??

[sys_subscribed_process](#)

A single linked list element containing the ID of a process that is subscribed to a specific event ??

[sys_uart_txdata](#)

Linked list element to store transmission data ??

5 File Index

5.1 File List

Here is a list of all files with brief descriptions:

[definitions.h](#)

It declares general preprocessor variables and types ??

[interrupts.c](#)

It defines the functions to create atomic sections ??

[interrupts.h](#)

It declares interrupt priority levels and functions to create atomic sections ??

[memory.c](#)

It defines functions to allocate, free, and copy memory ??

[memory.h](#)

It declares functions to allocate, free, and copy memory ??

[system.c](#)

Defines functions to initialise and start OpenSwarm ??

[system.h](#)

It declares functions to initialise and start OpenSwarm ??

[events/events.c](#)

Defines functions to create, (un)subscribe, (un)register, and delete events and related handler ??

[events/events.h](#)

Declares functions to create, (un)subscribe, (un)register, and delete events and related handler ??

[io/io.c](#)

It defines functions to control the IO timer and to (un)register IO Handler ??

[io/io.h](#)

It declares functions to control the IO timer and to (un)register IO Handler ??

[io/io_clock.c](#)

It defines the system clock that provides a continuous time value (granulation of 1 ms) ??

[io/io_clock.h](#)

It declares the system clock that provides a continuous time value (granulation of 1 ms) ??

[platform/e-puck/camera.c](#)

It defines functions to process data retrieved by a camera ??

[platform/e-puck/camera.h](#)

It declares functions to process data retrieved by a camera ??

[platform/e-puck/camera_processing.c](#)

??

platform/e-puck/ camera_processing.h	External set of functions to assist the use of the camera. (provided by Shefpuck)	??
platform/e-puck/ DSPIC30F6014A_HDI.h	Declares e-puck specific types and preprocessor variables	??
platform/e-puck/ i2c.c	It defines functions to read and write on the I2C interface	??
platform/e-puck/ i2c.h	It declares functions to read and write on the I2C interface	??
platform/e-puck/ i2c_data.c	It defines functions to manage the I2C queue	??
platform/e-puck/ i2c_data.h	It declares functions to manage the I2C queue	??
platform/e-puck/ i2c_HDI.c	Hardware dependent implementations to read and write on the I2C interface	??
platform/e-puck/ i2c_HDI.h	Hardware dependent implementations to read and write on the I2C interface	??
platform/e-puck/ io_HDI.c	Hardware dependent implementations to control the IO timer and to (un)register IO Handler	??
platform/e-puck/ io_HDI.h	Hardware dependent implementations to control the IO timer and to (un)register IO Handler	??
platform/e-puck/ motors.c	It defines function to drive motors	??
platform/e-puck/ motors.h	It declares functions to drive motors	??
platform/e-puck/ motors_HDI.c	Hardware dependent implementations to drive motors	??
platform/e-puck/ motors_HDI.h	Hardware dependent implementations to drive motors	??
platform/e-puck/ process_Management_HDI.c	Hardware dependent implementations to manage processes (e.g. task swichting)	??
platform/e-puck/ process_Management_HDI.h	Hardware dependent implementations to manage processes (e.g. task swichting)	??
platform/e-puck/ remoteControl.c	It defines functions to receive and decode messages from a remote control	??
platform/e-puck/ remoteControl.h	It declares functions to receive and decode messages from a remote control	??
platform/e-puck/ remoteControl_HDI.c	Hardware dependent implementations to receive and decode messages from a remote control	??
platform/e-puck/ remoteControl_HDI.h	Hardware dependent implementations to receive and decode messages from a remote control	??
platform/e-puck/ system_Timer_HDI.c	Hardware dependent implementations to initialise, configure and run the system Timer	??

platform/e-puck/system_Timer_HDI.h	Hardware dependent implementations to initialise, configure and run the system Timer	??
platform/e-puck/traps.c	Hardware dependent implementations to catch hardware traps	??
platform/e-puck/uart.c	It declares functions to transmit bytes via UART	??
platform/e-puck/uart.h	It declares functions to transmit bytes via UART	??
platform/e-puck/uart_HDI.c	Hardware dependent implementations to transmit bytes via UART	??
platform/e-puck/uart_HDI.h	Hardware dependent implementations to transmit bytes via UART	??
processes/data.c	It defines functions to manage process lists and related structs	??
processes/data.h	It declares functions to manage process lists and related structs	??
processes/process_Management.c	It defines functions to manage processes (e.g. task creation, switching, termination)	??
processes/process_Management.h	It declares functions to manage processes (e.g. task creation, switching, termination)	??
processes/scheduler.c	It defines functions to specify a scheduling algorithm	??
processes/scheduler.h	It declares functions to specify a scheduling algorithm	??
processes/system_Timer.c	It defines functions to initialise, configure and run the system Timer	??
processes/system_Timer.h	It declares functions to initialise, configure and run the system Timer	??

6 Module Documentation

6.1 Kernel

OpenSwarm provides functions to start and initialise the operating system; allocate, free and copy memory; and create and end atomic sections.

Files

- file [interrupts.c](#)
It defines the functions to create atomic sections.
- file [interrupts.h](#)
It declares interrupt priority levels and functions to create atomic sections.
- file [memory.c](#)
It defines functions to allocate, free, and copy memory.

- file [memory.h](#)
It declares functions to allocate, free, and copy memory.
- file [system.c](#)
defines functions to initialise and start OpenSwarm.
- file [system.h](#)
It declares functions to initialise and start OpenSwarm.

6.1.1 Detailed Description

OpenSwarm provides functions to start and initialise the operating system; allocate, free and copy memory; and create and end atomic sections.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

6.1.2 Introduction

The Kernel contains basic functions to initialise and start all modules of OpenSwarm. This part of OpenSwarm executes all three modules

1. [Process Management](#)
2. [Event Management](#)
3. [I/O Management](#)

First, the system has to be configured by setting preprocessor values and variables. Then, OpenSwarm initialises the system and I/O according to its configuration (preprocessor definitions) and with an additional command the system can be started. In addition, functions to define atomic sections (sections that cannot be interrupted by anything), allocate and free memory are also provided.

6.1.2.1 Definitions

definition.h provides standardised ports, configures the used platform, and defines general preprocessor/type definitions that are needed in OpenSwarm.

6.1.2.2 Memory Management

OpenSwarm is designed for processing units that lack a MMU (Memory Management Unit). As a consequence, advance memory management functions as virtual memory cannot be implemented without a significant reduction of efficiency.

However, OpenSwarm provides atomic functions to allocate, free and copy memory in [memory.h](#).

6.1.2.3 Interrupt Management

OpenSwarm provides defined interrupt priorities and functions to create atomic sections in [interrupts.h](#). A atomic section is a section of code that cannot be interrupted.

6.2 Event Management

Events are the main information exchange method in OpenSwarm. Events can be emitted, created, and (un)registered. Functions to handle events can also (un)subscribe to certain events. It can be used to synchronise and communicate with processes, to implement asynchronous programming model, and process incoming data/signals.

Files

- file [events.c](#)
defines functions to create, (un)subscribe, (un)register, and delete events and related handler.
- file [events.h](#)
declares functions to create, (un)subscribe, (un)register, and delete events and related handler.

6.2.1 Detailed Description

Events are the main information exchange method in OpenSwarm. Events can be emitted, created, and (un)registered. Functions to handle events can also (un)subscribe to certain events. It can be used to synchronise and communicate with processes, to implement asynchronous programming model, and process incoming data/signals.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

In general, events are data structures passed from one process/part of OpenSwarm to another.

For example, an event might be triggered by detecting another robot within range. If a function is subscribed to this event, the robot can react on the detection of another robot.

Events can also carry additional information—such as the distance and angle to the detected robot.

6.2.2 Usage

The event system is ready to be used after the start of OpenSwarm. Each event is identified by an integer **eventID**. To use an event the following steps have to be taken:

1. An event (**eventID**) can be (un)registered by [Sys_Register_Event\(uint16 eventID\)](#) and [Sys_Unregister_Event\(uint16 eventID\)](#). When an event is registered, it means that an event (**eventID**) can occur and be handled by OpenSwarm.
2. After the event was registered, processes can be subscribed to it with [Sys_Subscribe_to_Event\(uint16 eventID, uint16 pid, pEventHandlerFunction handler, pConditionFunction condition\)](#) and [Sys_Unsubscribe_from_Event\(uint16 eventID, uint16 pid\)](#). During the subscription, an event handler (i.e. a function to process the event) is subscribed to a specific event (**eventID**) and a process. Each event handler of a process for a specific event is unique. As a result, the same handler function can be used by multiple processes.
3. After an event is registered, events can be sent with [Sys_Send_Event\(uint16 eventID, void *data, uint16 data_size\)](#) and [Sys_Send_IntEvent\(uint16 eventID, uint16 data\)](#).

6.2.3 Example

```
#include "os/system.h"
#include "os/events/events.h"

#define USER_EVENT_ID 0xCC

bool pConditionFunction(void *data){//only executes the eventHandler every 5th time.
    static int counter = 0;
```



```
        if(++counter >= 4){ //if the event occurred the 5th time
            counter = 0;
            return true; //execute eventHandler
        }

        return false; //don't execute eventHandler
    }

bool eventHandler(uint16 pid, uint16 eventID, sys_event_data *data){
    //do something with the data
}

int main(void){
    //initialise some global or local variables

    int variable;

    Sys_Init_Kernel();

    Sys_Register_Event(USER_EVENT_ID);

    Sys_Start_Kernel(); //OpenSwarm is running now.

    while(true){
        if( ... ){ //under a certain condition
            Sys_Send_Event(USER_EVENT_ID, &variable, sizeof(int));
            //alternatively, you could use Sys_Send_IntEvent(USER_EVENT_ID, variable);
        }
        //do something
    }
}
```

6.2.4 License

LICENSE: adapted FreeBSD License (see <http://openswarm.org/license>)

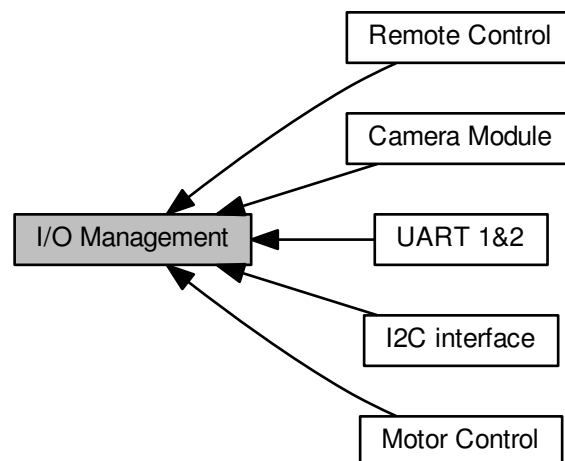
Copyright (c) 2015, Stefan M. Trenkwalder

All rights reserved.

6.3 I/O Management

I/O Management controls the input (e.g. sensors or camera), output (e.g. motors), and communication (e.g. UART, CAN, Bluetooth, ...) devices. For robots, I/O devices are important to interact with its environment, which is the main purpose in robotics. Consequently, this module provides functions and mechanisms to use these I/O devices.

Collaboration diagram for I/O Management:



Modules

- [Camera Module](#)
The camera module is used to retrieve raw camera data, process the incoming frames, and emits the result as [events](#).
- [I2C interface](#)
Functions to read from and write on the I2C interface.
- [Motor Control](#)
Functions to control the two stepper motors of the e-puck.
- [Remote Control](#)
Functions to receive data from a remote control.
- [UART 1&2](#)
Functions to control the message flow of the UART interface.

Files

- file [io.c](#)
It defines functions to control the IO timer and to (un)register IO Handler.
- file [io.h](#)
It declares functions to control the IO timer and to (un)register IO Handler.
- file [io_clock.c](#)
It defines the system clock that provides a continuous time value (granulation of 1 ms).
- file [io_clock.h](#)
It declares the system clock that provides a continuous time value (granulation of 1 ms).
- file [io_HDI.c](#)

Hardware dependent implementations to control the IO timer and to (un)register IO Handler.

- file [io_HDL.h](#)

Hardware dependent implementations to control the IO timer and to (un)register IO Handler.

6.3.1 Detailed Description

I/O Management controls the input (e.g. sensors or camera), output (e.g. motors), and communication (e.g. UART, CAN, Bluetooth, ...) devices. For robots, I/O devices are important to interact with its environment, which is the main purpose in robotics. Consequently, this module provides functions and mechanisms to use these I/O devices.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

6.3.2 Introduction

The I/O Management is started by initialising & starting of the [Kernel](#)

I/O devices build the interface between environment and robot. In general, I/O devices can be divided into two sets:

- interrupt-driven I/O devices are dedicated hardware modules on the microcontroller that signal by emitting hardware interrupts at defined changes of the device. As a result, these devices work concurrently and independently of the main OpenSwarm processes. For instance, the UART signals a received byte; sent byte; empty sending/receiving buffer; and full buffer. Interrupt-driven I/O devices on a microcontroller are usually UART, I2C, ADC, External Interrupt pins, ...
- non-interrupt-driven I/O devices has to be checked regularly if its state has changed. This is done by the I/O handler that are executed by occurrence of the I/O timer. non-interrupt-driven I/O devices on a microcontroller are usually General purpose IO (GPIO) pins and any device that is connected to them.

Commonly used on robots, I/O devices might be a camera, motors, or gripper.

6.3.3 Usage

The I/O management is initialised with [Sys_Init_IOManagement\(\)](#), which initialises the System Timer (100us) and a list of I/O handlers that need to be executed periodically. After starting the timer with [Sys_Start_IOManagement\(void\)](#), it can be stopped with [Sys_Stop_IOManagement\(void\)](#).

The I/O Timer can be manipulated as follows

- Stop: [Sys_Stop_IOTimer\(void\)](#)
- Continue: [Sys_Continue_IOTimer\(void\)](#)
- Reset: [Sys_Reset_IOTimer\(void\)](#)
- Disable: [Sys_Disable_IOTimerInterrupt\(void\)](#)
- Enable: [Sys_Enable_IOTimerInterrupt\(void\)](#)
- Force an I/O Timer interrupt: [Sys_Force_IOTimerInterrupt\(void\)](#)

New I/O devices can be added and removed during run-time by (un)registering with [Sys_Register_IOHandler\(pFunction func\)](#) and [Sys_Unregister_IOHandler\(pFunction func\)](#).

6.3.4 License

LICENSE: adapted FreeBSD License (see <http://openswarm.org/license>)

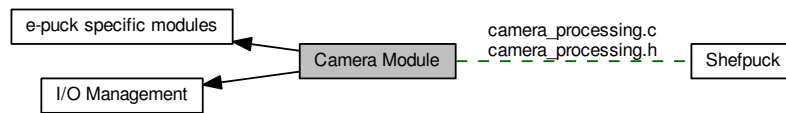
Copyright (c) 2015, Stefan M. Trenkwalder

All rights reserved.

6.4 Camera Module

The camera module is used to retrieve raw camera data, process the incoming frames, and emits the result as [events](#).

Collaboration diagram for Camera Module:



Files

- file [camera.c](#)
It defines functions to process data retrieved by a camera.
- file [camera.h](#)
It declares functions to process data retrieved by a camera.
- file [camera_processing.c](#)
- file [camera_processing.h](#)
External set of functions to assist the use of the camera. (provided by [Shefpuck](#))

6.4.1 Detailed Description

The camera module is used to retrieve raw camera data, process the incoming frames, and emits the result as [events](#).

This module currently is under development and is using functions of the e-puck library provided using Subversion at [svn://svn.gna.org/svn/e-puck/trunk](http://svn.gna.org/svn/e-puck/trunk) (01/01/2016).

Todo The used functions from the e-puck library are very time and computational intensive. These function need to be rewritten to decrease the processing load.

6.4.2 Usage

The camera is initialised and started by [Sys_Init_Camera\(\)](#) and [Sys_Start_Camera\(\)](#) respectively.

The camera uses a preprocessor to process a frame and generate the required events. This preprocessor can be defined by [Sys_Set_Preprocessing\(pCameraPreProcessor\)](#).

A received frame, if available ([isNewFrameAvailable\(\)](#)) can be obtained with [getFinishedFrame\(\)](#).

Note

An e-puck cannot store a entire frame with full resolution (640x480). Consequently, only sub-frames can be taken.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

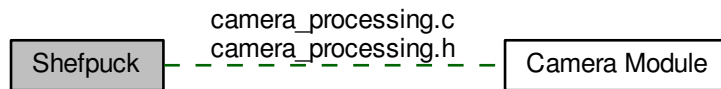
6.4.3 License

LICENSE: adapted FreeBSD License (see <http://openswarm.org/license>)
Copyright (c) 2015, Stefan M. Trenkwalder
All rights reserved.

6.5 Shefpuck

External set of functions to assist the programming of the e-Puck.

Collaboration diagram for Shefpuck:



Files

- file [camera_processing.c](#)
- file [camera_processing.h](#)

External set of functions to assist the use of the camera. (provided by [Shefpuck](#))

6.5.1 Detailed Description

External set of functions to assist the programming of the e-Puck.

Author

Yuri Kaszubowski Lopes yurikazuba@gmail.com

This file is part of shefpuck.

This library is in development.

6.5.2 License

shefpuck is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

shefpuck is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with shefpuck. If not, see <http://www.gnu.org/licenses/>. Copyright (C) 2014-2015 Yuri Kaszubowski Lopes - yurikazuba@gmail.com

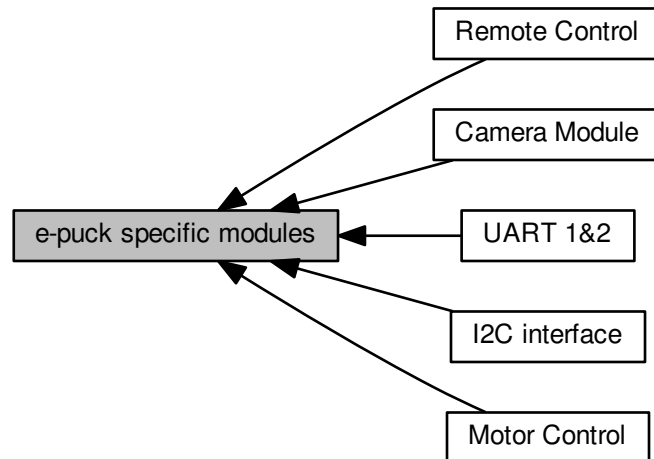
Note

Due to the use of the e-puck library while processing the camera input, this module is used to process a camera frame into a virtual simple line of sight sensor value. This module, as well as the functions used from the e-puck library, will be replaced.

6.6 e-puck specific modules

Modules and functions that are needed to use the e-puck platform (<http://www.gctronic.com/doc/index.php/E-Puck>)

Collaboration diagram for e-puck specific modules:



Modules

- [Camera Module](#)
The camera module is used to retrieve raw camera data, process the incoming frames, and emits the result as [events](#).
- [I2C interface](#)
Functions to read from and write on the I2C interface.
- [Motor Control](#)
Functions to control the two stepper motors of the e-puck.
- [Remote Control](#)
Functions to receive data from a remote control.
- [UART 1&2](#)
Functions to control the message flow of the UART interface.

Files

- file [DSPIC30F6014A_HDI.h](#)
declares e-puck specific types and preprocessor variables
- file [traps.c](#)
Hardware dependent implementations to catch hardware traps.

6.6.1 Detailed Description

Modules and functions that are needed to use the e-puck platform (<http://www.gctronic.com/doc/index.php/E-Puck>)

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

6.6.2 Features

The e-puck provides the following features:

6.6.2.1 Sensors:**6.6.2.1.1 8 infra-red proximity sensors**

The infra-red proximity sensors are currently under implementation. Therefore not ready yet.

6.6.2.1.2 accelerometer

The accelerometer weren't needed for many applications and, therefore, the priority to implement the accelerometer is small.

6.6.2.1.3 3 microphones

The microphones weren't needed for many applications and, therefore, the priority to implement the microphones is small.

6.6.2.1.4 camera:

The camera functions can be found at [Camera Module](#)

6.6.2.1.5 remote control receiver:

This function is fully implemented ([Remote Control](#)).

6.6.2.2 Actuators:**6.6.2.2.1 differential drive (ref motors).****6.6.2.2.2 leds:**

Hardware independent functions to control the LEDs are not yet implemented, due to it's simple nature. Currently you can use the MACROs LED0, LED1, ..., LED7, BODYLED, FRONTLED to set and unset these LEDs.

6.6.2.2.3 speaker:

The speakers weren't needed for many applications and, therefore, the priority to implement the speakers is small.

6.6.2.3 communication:**6.6.2.3.1 Bluetooth:**

The Bluetooth can be used by sending and receiving bytes via UART1 ([UART 1&2](#))

6.6.2.3.2 Infra-red communication

The infra-red proximity sensors can be used to transmit and receive data. This function leads to a local broadcasting. However, this function is still under development.

6.6.3 License

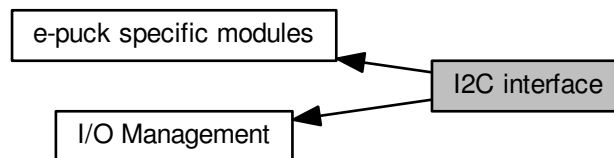
LICENSE: adapted FreeBSD License (see <http://openswarm.org/license>)
Copyright (c) 2015, Stefan M. Trenkwalder

All rights reserved.

6.7 I2C interface

Functions to read from and write on the I2C interface.

Collaboration diagram for I2C interface:



Files

- file [i2c.c](#)
It defines functions to read and write on the I2C interface.
- file [i2c.h](#)
It declares functions to read and write on the I2C interface.
- file [i2c_data.c](#)
It defines functions to manage the I2C queue.
- file [i2c_data.h](#)
It declares functions to manage the I2C queue.
- file [i2c_HDI.c](#)
Hardware dependent implementations to read and write on the I2C interface.
- file [i2c_HDI.h](#)
Hardware dependent implementations to read and write on the I2C interface.

6.7.1 Detailed Description

Functions to read from and write on the I2C interface.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Inter-Integrated Circuit bus is a multi-master, multi-slave, serial bus (see also <https://en.wikipedia.org/wiki/I%C2%B2C>)

6.7.2 Usage

The I2C interface can be initialised and started with [Sys_Init_I2C\(\)](#) and [Sys_Start_I2C\(\)](#) respectively. Similarly, it can be paused, continued, or stopped by [Sys_Pause_I2C\(\)](#), [Sys_Continue_I2C\(\)](#), or [Sys_Stop_I2C\(\)](#) respectively. While the interface is running, data can be written with [Sys_I2C_SentBytes\(uint8, uint8 *, uint16\)](#). Values can be read with [Sys_I2C_Read\(uint8, uint8 *, uint16, pByteFunction\)](#) where the request message has also to be specified.

Todo testing and debugging of this module.

Note

This module is currently untested. It might not work or includes some bugs. The interrupt handler `_MI2↔CInterrupt()` is also out commented, because it might interfere with the e-Puck library used in the camera module.

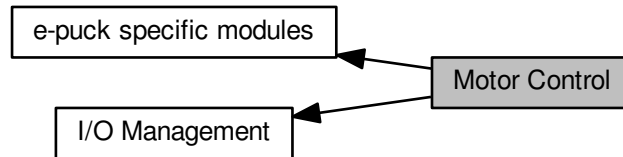
6.7.3 License

LICENSE: adapted FreeBSD License (see <http://openswarm.org/license>)
Copyright (c) 2015, Stefan M. Trenkwalder
All rights reserved.

6.8 Motor Control

Functions to control the two stepper motors of the e-puck.

Collaboration diagram for Motor Control:



Files

- file [motors.c](#)
It defines function to drive motors.
- file [motors.h](#)
It declares functions to drive motors.
- file [motors_HDI.c](#)
Hardware dependent implementations to drive motors.
- file [motors_HDI.h](#)
Hardware dependent implementations to drive motors.

6.8.1 Detailed Description

Functions to control the two stepper motors of the e-puck.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

The motor control module controls the speed and direction of the two stepper motors

6.8.2 Usage

After the initialisation with [Sys_Init_Motors\(\)](#), the motors can be used by setting the motor speed. This can be done by sending the motor velocities via events to `SYS_EVENT_IO_MOTOR_LEFT` and `SYS_EVENT_IO_MOTOR_RIGHT` or by setting the speed directly by calling [Sys_Set_LeftWheelSpeed\(sint16\)](#) and [Sys_Set_RightWheelSpeed\(sint16\)](#). The current speed can be obtained [Sys_Get_LeftWheelSpeed\(\)](#) and [Sys_Get_RightWheelSpeed\(\)](#).

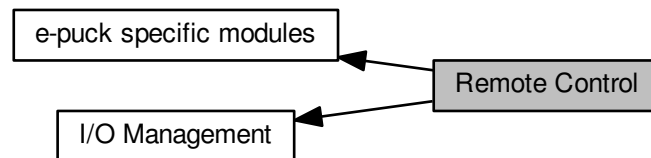
6.8.3 License

LICENSE: adapted FreeBSD License (see <http://openswarm.org/license>)
Copyright (c) 2015, Stefan M. Trenkwalder
All rights reserved.

6.9 Remote Control

Functions to receive data from a remote control.

Collaboration diagram for Remote Control:



Files

- file [remoteControl.c](#)
It defines functions to receive and decode messages from a remote control.
- file [remoteControl.h](#)
It declares functions to receive and decode messages from a remote control.
- file [remoteControl_HDI.c](#)
Hardware dependent implementations to receive and decode messages from a remote control.
- file [remoteControl_HDI.h](#)
Hardware dependent implementations to receive and decode messages from a remote control.

6.9.1 Detailed Description

Functions to receive data from a remote control.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

This module is based on the RC-5 coding for the Toshiba RC-3910 (<https://en.wikipedia.org/wiki/RC-5>)

6.9.2 Usage

After the initialisation with [Sys_Init_RemoteControl\(\)](#), the interface needs to be started to be able to receive or transmit bytes with [Sys_Start_RemoteControl\(\)](#).

After this, any pressed button on the remote control is received as an event (SYS_EVENT_IO_REMOECONTROL).

6.9.3 License

LICENSE: adapted FreeBSD License (see <http://openswarm.org/license>)

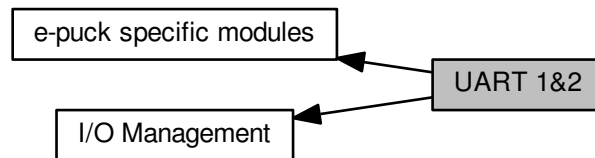
Copyright (c) 2015, Stefan M. Trenkwalder

All rights reserved.

6.10 UART 1&2

Functions to control the message flow of the UART interface.

Collaboration diagram for UART 1&2:



Files

- file [uart.c](#)
It declares functions to transmit bytes via UART.
- file [uart.h](#)
It declares functions to transmit bytes via UART.
- file [uart_HDI.c](#)
Hardware dependent implementations to transmit bytes via UART.
- file [uart_HDI.h](#)
Hardware dependent implementations to transmit bytes via UART.

6.10.1 Detailed Description

Functions to control the message flow of the UART interface.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

A UART (Universal Asynchronous Receiver Transmitter) interface is common on microcontroller to communicate with other devices on a serial bus (https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter) The UART 1 is used on the epuck to communicate with the Bluetooth transceiver.

6.10.2 Usage

After the initialisation with [Sys_Init_UART1\(\)](#) (same applies to UART2), the UART interface needs to be started to be able to receive or transmit bytes. This can be done by sending the bytes via event to `SYS_EVENT_IO_TO_BLUEETOOTH` (UART1) or by handing over the bytes directly by calling [Sys_Writeto_UART1\(void *, uint length\)](#) and [Sys_Writeto_UART2\(void *, uint\)](#). Incoming bytes can be received by defining a reading function with [Sys_SetReadingFunction_UART1\(pUART_reader\)](#) and [Sys_SetReadingFunction_UART2\(pUART_reader\)](#). This function is executed every time a new byte arrives.

6.10.3 License

LICENSE: adapted FreeBSD License (see <http://openswarm.org/license>)

Copyright (c) 2015, Stefan M. Trenkwalder

All rights reserved.

6.11 Process Management

Functions to create, switch, block, yield, and terminate processes and start critical sections.

Files

- file [process_Management_HDI.c](#)
Hardware dependent implementations to manage processes (e.g. task swichting)
- file [process_Management_HDI.h](#)
Hardware dependent implementations to manage processes (e.g. task swichting)
- file [system_Timer_HDI.c](#)
Hardware dependent implementations to initialise, configure and run the system Timer.
- file [system_Timer_HDI.h](#)
Hardware dependent implementations to initialise, configure and run the system Timer.
- file [data.c](#)
It defines functions to manage process lists and related structs.
- file [data.h](#)
It declares functions to manage process lists and related structs.
- file [process_Management.c](#)
It defines functions to manage processes (e.g. task creation, switching, termination)
- file [process_Management.h](#)
It declares functions to manage processes (e.g. task creation, switching, termination)
- file [scheduler.c](#)
It defines functions to specify a scheduling algorithm.
- file [scheduler.h](#)
It declares functions to specify a scheduling algorithm.
- file [system_Timer.c](#)
It defines functions to initialise, configure and run the system Timer.
- file [system_Timer.h](#)
It declares functions to initialise, configure and run the system Timer.

6.11.1 Detailed Description

Functions to create, switch, block, yield, and terminate processes and start critical sections.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

A process is a basic form to execute functions in OpenSwarm. Due to target device architecture, OpenSwarm does not provide functions to separate memory in pages or segments. Therefore, all processes are executed in the same memory area. Consequently, Each process can be seen as a single thread where all threads share the same memory space. A thread is represented by a function with no arguments and no return value. One function can be executed multiple times as individual threads.

OpenSwarm organises processes in four lists (pid sorted):

1. running list: includes only one element—the process that is executed at that time.
2. ready list: includes all processes are ready to be executed and will be scheduled according to the scheduling algorithm.
3. blocked list: includes all processes that are waiting for events to occur.
4. Zombie list: includes all processes that are about to be terminated but not deleted yet.

6.11.2 Usage

The process management is initialised with [Sys_Init_Process_Management\(void\)](#), which generated the System Thread (pid: 0) and initialises all data structures. After initialising, the following functions are available.

6.11.2.1 User code:

1. Processes are started and terminated with [Sys_Start_Process\(pFunction function\)](#) and [Sys_Kill_Process\(uint\)](#) respectively.
2. A Process can be yield with [Sys_Yield\(void\)](#) and remains in the ready list. The process can be rescheduled by the scheduler.
3. A thread/process can be suspended while waiting for arriving events with [Sys_Wait_For_Event\(uint\)](#) and [Sys_Wait_For_Condition\(uint, pConditionFunction\)](#). Processes that are suspended are on the block list and are not rescheduled whilst in it.

6.11.2.2 Internal function (shouldn't be used by the user)

6.11.2.2.1 Scheduling (functions to decide which process is executed at which time)

Scheduling-related functions can be found in [scheduler.h](#) and [process_Management.h](#).

- The executing process can be switched by using [Sys_Switch_Process\(uint\)](#) and [Sys_Switch_to_next_Process\(void\)](#).
- To implement a new scheduling algorithm, `sys_scheduler_info_s`, a function to implement the algorithm (`void function(void)`), and a function to set the default values of the struct (`void Sys_Set_Defaults_Info(sys_scheduler_info *)`) needs to be implemented ([scheduler.h](#)).

6.11.2.2.2 System Timer (timer to start the scheduling, found in `system_Timer.h`):

1. The System Timer needs to be initialised and started by [Sys_Init_SystemTimer\(pFunction\)](#) and [Sys_Start_SystemTimer\(void\)](#) respectively (these functions are used when the process Management is initialised and started).
2. It can be stopped, continued, and reset by [Sys_Stop_SystemTimer\(\)](#), [Sys_Continue_SystemTimer\(\)](#), and [Sys_Reset_SystemTimer\(\)](#) respectively.
3. The timer interrupt can be disabled and enabled by [Sys_Disable_TimerInterrupt\(void\)](#) and [Sys_Enable_TimerInterrupt\(void\)](#).
4. To force a system timer and therefore the scheduling process, [Sys_Force_TimerInterrupt\(\)](#) will be executed.

6.11.2.2.3 Process Event handling (functions to store/process events with it's subscribed process and add/remove subscriptions) \ref events

- Event subscription to a process can be added and removed by [Sys_Add_Event_Subscription\(uint, uint, pEventHandlerFunction, pConditionFunction\)](#) and [Sys_Remove_Event_Subscription\(uint, uint, pEventHandlerFunction\)](#).
- Removing all subscription to any process of a single event can be done by [Sys_Remove_All_Event_Subscriptions\(uint eventID\)](#).
- To copy the data of an occurred event to a specific process, [Sys_Add_Event_to_Process\(uint, uint, void *, uint\)](#) can be used.
- All stored data is processed by its registered event handler by [Sys_Execute_All_EventHandler\(\)](#).
- The event data can be cleared with [Sys_Clear_EventData\(sys_event_data **\)](#).

6.11.3 Example

```
#include "os/system.h"
#include "os/events/events.h"
#include "os/processes/process_Management.h"

#define WAIT_FOR_ME 0x0F

void thread(void){//thread definition
    while(true){
        //do something as a thread
        sys_event_data * data = Sys_Wait_For_Event (WAIT_FOR_ME);//
        blocking
        Sys_Clear_EventData (data);
    }
}

int main(void){
    //initialise some global or local variables

    int variable;

    Sys_Init_Kernel();

    Sys_Register_Event (WAIT_FOR_ME);

    Sys_Start_Kernel();//OpenSwarm is running now
    while(1){
        if( ... ){//under a certain condition
            Sys_Send_Event (WAIT_FOR_ME, &variable, sizeof(int));
            //alternatively, you could use Sys_Send_IntEvent(WAIT_FOR_ME, variable);
        }
        //do something
    }
}
```

6.11.4 License

LICENSE: adapted FreeBSD License (see <http://openswarm.org/license>)
Copyright (c) 2015, Stefan M. Trenkwalder
All rights reserved.

7 Data Structure Documentation

7.1 `sys_event_data` Struct Reference

It is a single linked list element and contains data of an occurred event.

```
#include <events.h>
```

Data Fields

- void * [value](#)
- [uint](#) `size`
- struct `sys_event_data_s` * [next](#)

7.1.1 Detailed Description

It is a single linked list element and contains data of an occurred event.

Definition at line 94 of file `events.h`.

7.1.2 Field Documentation

7.1.2.1 `struct sys_event_data_s* sys_event_data::next`

pointer to the next element in the List

Definition at line 98 of file `events.h`.

7.1.2.2 `uint sys_event_data::size`

size of the transferred data (bytes)

Definition at line 96 of file `events.h`.

7.1.2.3 `void* sys_event_data::value`

pointer to data transferred by the event

Definition at line 95 of file `events.h`.

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

- `events/`[events.h](#)

7.2 `sys_i2c_msg` Struct Reference

It is a single linked list element containing messages that need to be sent via I2C. This list acts as a message buffer.

```
#include <i2c_data.h>
```

Data Fields

- [uint8](#) `i2c_device_address`
- [uint8](#) * `data`
- [uint](#) `length`
- bool `write`
- [pByteFunction](#) `handler`
- struct `sys_i2c_message_s` * [next](#)

7.2.1 Detailed Description

It is a single linked list element containing messages that need to be sent via I2C. This list acts as a message buffer.
Definition at line 30 of file i2c_data.h.

7.2.2 Field Documentation

7.2.2.1 uint8* sys_i2c_msg::data

pointer to the bytes that should be sent

Definition at line 32 of file i2c_data.h.

7.2.2.2 pByteFunction sys_i2c_msg::handler

function pointer to handle incoming bytes (unused = 0)

Definition at line 35 of file i2c_data.h.

7.2.2.3 uint8 sys_i2c_msg::i2c_device_address

7-bit I2C address

Definition at line 31 of file i2c_data.h.

7.2.2.4 uint sys_i2c_msg::length

number of bytes contained in data

Definition at line 33 of file i2c_data.h.

7.2.2.5 struct sys_i2c_message_s* sys_i2c_msg::next

pointer to the next element in the linked list

Definition at line 36 of file i2c_data.h.

7.2.2.6 bool sys_i2c_msg::write

is it a write only operating (true) or is the slave transmitting data as a response (false)

Definition at line 34 of file i2c_data.h.

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

- [platform/e-puck/i2c_data.h](#)

7.3 sys_motors Struct Reference

This struct contains the speed for a motor.

Data Fields

- [sint16 speed](#)

7.3.1 Detailed Description

This struct contains the speed for a motor.

Definition at line 31 of file motors.c.

7.3.2 Field Documentation

7.3.2.1 sint16 sys_motors::speed

speed of one motor

Definition at line 32 of file motors.c.

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

- [platform/e-puck/motors.c](#)

7.4 sys_occurred_event Struct Reference

Linked list element containing an occurred events.

```
#include <data.h>
```

Data Fields

- [uint eventID](#)
- [struct sys_occurred_event_s * next](#)

7.4.1 Detailed Description

Linked list element containing an occurred events.

It is a single linked list element that stores the id on an occurred event.

Definition at line 34 of file data.h.

7.4.2 Field Documentation

7.4.2.1 uint sys_occurred_event::eventID

ID of the occurred event

Definition at line 35 of file data.h.

7.4.2.2 struct sys_occurred_event_s* sys_occurred_event::next

pointer to the next element in the linked list.

Definition at line 37 of file data.h.

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

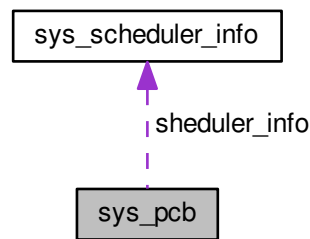
- [processes/data.h](#)

7.5 sys_pcb Struct Reference

Process Control Block contains all data for a single process.

```
#include <data.h>
```

Collaboration diagram for sys_pcb:



Data Fields

- [uint process_ID](#)
- [uint stackPointer](#)
- [uint framePointer](#)
- [uint stackPointerLimit](#)
- [sys_scheduler_info sheduler_info](#)
- [sys_process_event_handler * event_register](#)
- [uint * process_stack](#)

7.5.1 Detailed Description

Process Control Block contains all data for a single process.

It contains all information related to a single process. (including stack pointer, frame pointer, stack, etc.)

Definition at line 58 of file data.h.

7.5.2 Field Documentation

7.5.2.1 sys_process_event_handler* sys_pcb::event_register

Lists of all events the process is subscribed to

Definition at line 66 of file data.h.

7.5.2.2 uint sys_pcb::framePointer

Frame Pointer Register

Definition at line 62 of file data.h.

7.5.2.3 uint sys_pcb::process_ID

Process identifier

Definition at line 60 of file data.h.

7.5.2.4 uint* sys_pcb::process_stack

Pointer to the beginning of the stack

Definition at line 68 of file data.h.

7.5.2.5 `sys_scheduler_info` `sys_pcb::sheduler_info`

scheduler-specific datastructure

Definition at line 65 of file `data.h`.

7.5.2.6 `uint` `sys_pcb::stackPointer`

Stack Pointer Register

Definition at line 61 of file `data.h`.

7.5.2.7 `uint` `sys_pcb::stackPointerLimit`

Stack Pointer Limit Register

Definition at line 63 of file `data.h`.

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

- `processes/data.h`

7.6 `sys_pcb_list_element` Struct Reference

Double linked list element containing `sys_process_control_block`.

```
#include <data.h>
```

Data Fields

- `sys_process_control_block` [pcb](#)
- `struct sys_process_control_block_list_element_s *` [previous](#)
- `struct sys_process_control_block_list_element_s *` [next](#)

7.6.1 Detailed Description

Double linked list element containing `sys_process_control_block`.

It is a double linked list element containing the PCB of a process

Definition at line 77 of file `data.h`.

7.6.2 Field Documentation

7.6.2.1 `struct sys_process_control_block_list_element_s*` `sys_pcb_list_element::next`

pointer to the next PCB

Definition at line 82 of file `data.h`.

7.6.2.2 `sys_process_control_block` `sys_pcb_list_element::pcb`

Process Control Block of a process

Definition at line 79 of file `data.h`.

7.6.2.3 `struct sys_process_control_block_list_element_s*` `sys_pcb_list_element::previous`

pointer to the previous PCB

Definition at line 81 of file `data.h`.

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

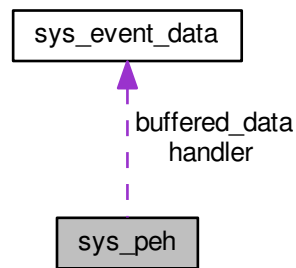
- [processes/data.h](#)

7.7 sys_peh Struct Reference

Double linked list element of process event-handlers.

```
#include <data.h>
```

Collaboration diagram for sys_peh:



Data Fields

- [uint eventId](#)
- [pEventHandlerFunction handler](#)
- [pConditionFunction condition](#)
- [sys_event_data * buffered_data](#)
- [struct sys_process_event_handler_s * previous](#)
- [struct sys_process_event_handler_s * next](#)

7.7.1 Detailed Description

Double linked list element of process event-handlers.

It is a double linked list containing all information needed to decide if the event-handler should be executed for an occurred event or not. It stores the pointer to the handler the condition function and data.

Definition at line 44 of file data.h.

7.7.2 Field Documentation

7.7.2.1 sys_event_data* sys_peh::buffered_data

stores a list of recieved event data that need to be processed

Definition at line 48 of file data.h.

7.7.2.2 pConditionFunction sys_peh::condition

Pointer to a function which checks if the event-handler should be executed (true) or not (false)

Definition at line 47 of file data.h.

7.7.2.3 uint sys_peh::eventID

ID of the occurred event

Definition at line 45 of file data.h.

7.7.2.4 pEventHandlerFunction sys_peh::handler

Pointer to a function which processes occurred events

Definition at line 46 of file data.h.

7.7.2.5 struct sys_process_event_handler_s* sys_peh::next

pointer to the next element in the linked list.

Definition at line 51 of file data.h.

7.7.2.6 struct sys_process_event_handler_s* sys_peh::previous

pointer to the previous element in the linked list.

Definition at line 50 of file data.h.

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

- [processes/data.h](#)

7.8 sys_pIOHandler Struct Reference

Linked list element containing I/O Handler function pointers.

```
#include <io_HDI.h>
```

Data Fields

- [pFunction function](#)
- struct sys_periodical_IOHandler_s * [next](#)

7.8.1 Detailed Description

Linked list element containing I/O Handler function pointers.

It is a single linked list element containing a function pointer to an I/O handler. I/O Handlers are functions that are periodically executed to interact with a specific I/O device.

Definition at line 32 of file io_HDI.h.

7.8.2 Field Documentation

7.8.2.1 pFunction sys_pIOHandler::function

function pointer to the I/O handler

Definition at line 33 of file io_HDI.h.

7.8.2.2 struct sys_periodical_IOHandler_s* sys_pIOHandler::next

pointer to the next I/O handler

Definition at line 35 of file io_HDI.h.

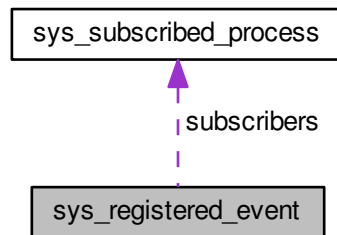
The documentation for this struct was generated from the following file:

- [platform/e-puck/io_HDI.h](#)

7.9 sys_registered_event Struct Reference

A single linked element containing a registered event and its subscribers.

Collaboration diagram for sys_registered_event:



Data Fields

- [uint eventID](#)
- [sys_subscribed_process * subscribers](#)
- [struct sys_registered_event_s * next](#)

7.9.1 Detailed Description

A single linked element containing a registered event and its subscribers.

It is a single linked list element that contains registered events and a list of processes that are subscribed to it.

Definition at line 34 of file events.c.

7.9.2 Field Documentation

7.9.2.1 uint sys_registered_event::eventID

event identifier

Definition at line 35 of file events.c.

7.9.2.2 struct sys_registered_event_s* sys_registered_event::next

pointer to the next element in the list

Definition at line 37 of file events.c.

7.9.2.3 sys_subscribed_process* sys_registered_event::subscribers

pointer to a list of subscribed processes

Definition at line 36 of file events.c.

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

- [events/events.c](#)

7.10 sys_rgb_pixel Struct Reference

This bitfield contains the structure of a received camera pixel.

```
#include <camera.h>
```

Data Fields

- [uint8 red](#): 5
- [uint8 green](#): 6
- [uint8 blue](#): 5

7.10.1 Detailed Description

This bitfield contains the structure of a received camera pixel.

Definition at line 57 of file camera.h.

7.10.2 Field Documentation

7.10.2.1 uint8 sys_rgb_pixel::blue

Definition at line 60 of file camera.h.

7.10.2.2 uint8 sys_rgb_pixel::green

Definition at line 59 of file camera.h.

7.10.2.3 uint8 sys_rgb_pixel::red

Definition at line 58 of file camera.h.

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

- [platform/e-puck/camera.h](#)

7.11 sys_scheduler_info Struct Reference

The scheduling information for each process.

```
#include <scheduler.h>
```

Data Fields

- [uint state](#)
- [uint priority](#)

7.11.1 Detailed Description

The scheduling information for each process.

This struct contains all values that are needed for the scheduling algorithm. For instance, if priority based round robin scheduling is used, the process priorities should be stored within this struct. This struct is designed to be reimplemented by the user, if required.

Definition at line 40 of file scheduler.h.

7.11.2 Field Documentation

7.11.2.1 uint sys_scheduler_info::priority

process priority level

Definition at line 42 of file scheduler.h.

7.11.2.2 uint sys_scheduler_info::state

Process state information

Definition at line 41 of file scheduler.h.

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

- [processes/scheduler.h](#)

7.12 sys_subscribed_process Struct Reference

A single linked list element containing the ID of a process that is subscribed to a specific event.

Data Fields

- [uint pid](#)
- struct sys_subscribed_process_s * [next](#)

7.12.1 Detailed Description

A single linked list element containing the ID of a process that is subscribed to a specific event.

Definition at line 24 of file events.c.

7.12.2 Field Documentation

7.12.2.1 struct sys_subscribed_process_s* sys_subscribed_process::next

pointer to the next element in the list

Definition at line 26 of file events.c.

7.12.2.2 uint sys_subscribed_process::pid

process identifier

Definition at line 25 of file events.c.

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

- [events/events.c](#)

7.13 sys_uart_txdata Struct Reference

Linked list element to store transmission data.

```
#include <uart_HDI.h>
```

Data Fields

- [uint8 * data](#)
- [uint length](#)
- `struct sys_uart_tx_data_s * next`

7.13.1 Detailed Description

Linked list element to store transmission data.

It is a single linked list containing a set of bytes that should be sent via UART.

Definition at line 45 of file `uart_HDI.h`.

7.13.2 Field Documentation

7.13.2.1 `uint8* sys_uart_txdata::data`

pointer to bytes that should be sent

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

7.13.2.2 `uint sys_uart_txdata::length`

number of bytes that need to be sent

Definition at line 47 of file `uart_HDI.h`.

7.13.2.3 `struct sys_uart_tx_data_s* sys_uart_txdata::next`

pointer to the next element in the list

Definition at line 49 of file `uart_HDI.h`.

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

- `platform/e-puck/uart_HDI.h`

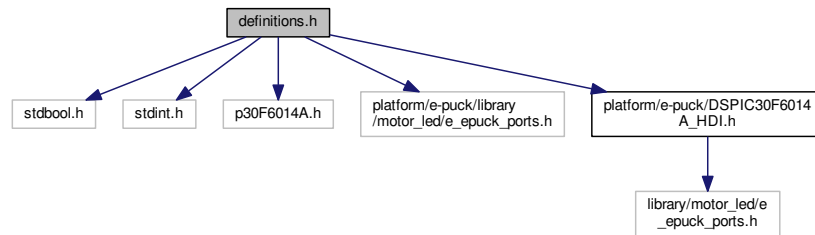
8 File Documentation

8.1 `definitions.h` File Reference

It declares general preprocessor variables and types.

```
#include <stdbool.h>
#include <stdint.h>
#include <p30F6014A.h>
#include "platform/e-puck/library/motor_led/e_epuck_ports.h"
#include "platform/e-puck/DSPIC30F6014A_HDI.h"
```

Include dependency graph for definitions.h:



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



Macros

- `#define` [EPUCK_USED](#)
- `#define` [UART1_RX_RF2](#)
- `#define` [UART1_TX_RF3](#)
- `#define` [UART2_RX_RF4](#)
- `#define` [UART2_TX_RF5](#)
- `#define` [UART1_RX_DIR_TRISF2](#)
- `#define` [UART1_TX_DIR_TRISF3](#)
- `#define` [UART2_RX_DIR_TRISF4](#)
- `#define` [UART2_TX_DIR_TRISF5](#)
- `#define` [SYS_EVENT_TERMINATION](#) 0x01
- `#define` [SYS_EVENT_IO_MOTOR_LEFT](#) 0x02
- `#define` [SYS_EVENT_IO_MOTOR_RIGHT](#) 0x03
- `#define` [SYS_EVENT_IO_CAMERA](#) 0x04
- `#define` [SYS_EVENT_IO_REMOECONTROL](#) 0x05
- `#define` [SYS_EVENT_IO_TO_BLUETOOTH](#) 0x06
- `#define` [SYS_EVENT_1ms_CLOCK](#) 0x07
- `#define` [ALL_FUNCTIONS](#) ((pEventHandlerFunction) 0xFFFFFFFF)

Typedefs

- `typedef` unsigned char [uint8](#)
- `typedef` unsigned short [uint16](#)
- `typedef` unsigned int [uint32](#)
- `typedef` signed char [sint8](#)
- `typedef` signed short [sint16](#)
- `typedef` signed int [sint32](#)
- `typedef` signed short [sint](#)
- `typedef` unsigned short [uint](#)
- `typedef` void(* [pFunction](#)) (void)
- `typedef` void(* [pByteFunction](#)) ([uint8](#))
- `typedef` void(* [pUART_reader](#)) ([uint8](#) data)

Enumerations

- enum `sys_colour` {
`BLACK` = 0b00000000, `RED` = 0b00000100, `YELLOW` = 0b00000110, `GREEN` = 0b00000010,
`CYAN` = 0b00000011, `BLUE` = 0b00000001, `MAGENTA` = 0b00000101, `WHITE` = 0b00000111 }
defines a system-wide colours

8.1.1 Detailed Description

It declares general preprocessor variables and types.

Author

Stefan M. Trenkwalder

Version

1.0

Date

2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.1.2 Macro Definition Documentation

8.1.2.1 `#define ALL_FUNCTIONS ((pEventHandlerFunction) 0xFFFFFFFF)`

the value to indicate all event handler

Definition at line 56 of file definitions.h.

8.1.2.2 `#define EPUCK_USED`

defines that the currently used platform is the e-puck

Definition at line 19 of file definitions.h.

8.1.2.3 `#define SYS_EVENT_1ms_CLOCK 0x07`

ID of the event that signals 1ms timer ticks

Definition at line 54 of file definitions.h.

8.1.2.4 `#define SYS_EVENT_IO_CAMERA 0x04`

ID of the event that is emitted by the camera

Definition at line 51 of file definitions.h.

8.1.2.5 `#define SYS_EVENT_IO_MOTOR_LEFT 0x02`

ID of the event that controls the left motor speed/direction

Definition at line 49 of file definitions.h.

8.1.2.6 #define SYS_EVENT_IO_MOTOR_RIGHT 0x03

ID of the event that controls the right motor speed/direction

Definition at line 50 of file definitions.h.

8.1.2.7 #define SYS_EVENT_IO_REMOECONTROL 0x05

ID of the event that is sent after receiving a remote control signal

Definition at line 52 of file definitions.h.

8.1.2.8 #define SYS_EVENT_IO_TO_BLUETOOTH 0x06

ID of the event that sends data via Bluetooth

Definition at line 53 of file definitions.h.

8.1.2.9 #define SYS_EVENT_TERMINATION 0x01

ID of the event that signal a general termination event

Definition at line 48 of file definitions.h.

8.1.2.10 #define UART1_RX_RF2

Definition at line 37 of file definitions.h.

8.1.2.11 #define UART1_RX_DIR_TRISF2

Definition at line 42 of file definitions.h.

8.1.2.12 #define UART1_TX_RF3

Definition at line 38 of file definitions.h.

8.1.2.13 #define UART1_TX_DIR_TRISF3

Definition at line 43 of file definitions.h.

8.1.2.14 #define UART2_RX_RF4

Definition at line 39 of file definitions.h.

8.1.2.15 #define UART2_RX_DIR_TRISF4

Definition at line 44 of file definitions.h.

8.1.2.16 #define UART2_TX_RF5

Definition at line 40 of file definitions.h.

8.1.2.17 #define UART2_TX_DIR_TRISF5

Definition at line 45 of file definitions.h.

8.1.3 Typedef Documentation**8.1.3.1 typedef void(* pByteFunction) (uint8)**

Defines a pointer to a function with no return value and one argument

Definition at line 86 of file definitions.h.

8.1.3.2 `typedef void(* pFunction) (void)`

Defines a pointer to a function with no return value and argument

Definition at line 85 of file definitions.h.

8.1.3.3 `typedef void(* pUART_reader) (uint8 data)`

Defines a pointer to a function with no return value and one argument

Definition at line 88 of file definitions.h.

8.1.3.4 `typedef signed short sint`

e-puck specific valued for the default signed integer

Definition at line 81 of file definitions.h.

8.1.3.5 `typedef signed short sint16`

Defines a signed 16bit integer

Definition at line 77 of file definitions.h.

8.1.3.6 `typedef signed int sint32`

Defines a signed 32bit integer

Definition at line 78 of file definitions.h.

8.1.3.7 `typedef signed char sint8`

Defines a signed 8bit integer

Definition at line 76 of file definitions.h.

8.1.3.8 `typedef unsigned short uint`

e-puck specific valued for the default unsigned integer

Definition at line 82 of file definitions.h.

8.1.3.9 `typedef unsigned short uint16`

Defines an unsigned 16bit integer

Definition at line 74 of file definitions.h.

8.1.3.10 `typedef unsigned int uint32`

Defines an unsigned 32bit integer

Definition at line 75 of file definitions.h.

8.1.3.11 `typedef unsigned char uint8`

Defines an unsigned 8bit integer

Definition at line 73 of file definitions.h.

8.1.4 Enumeration Type Documentation

8.1.4.1 `enum sys_colour`

defines a system-wide colours

This enum defines system-wide colours. (it is based on one bit for red, blue, and green). In total, 8 colours are defined with the first three bits.

Enumerator

BLACK
RED
YELLOW
GREEN
CYAN
BLUE
MAGENTA
WHITE

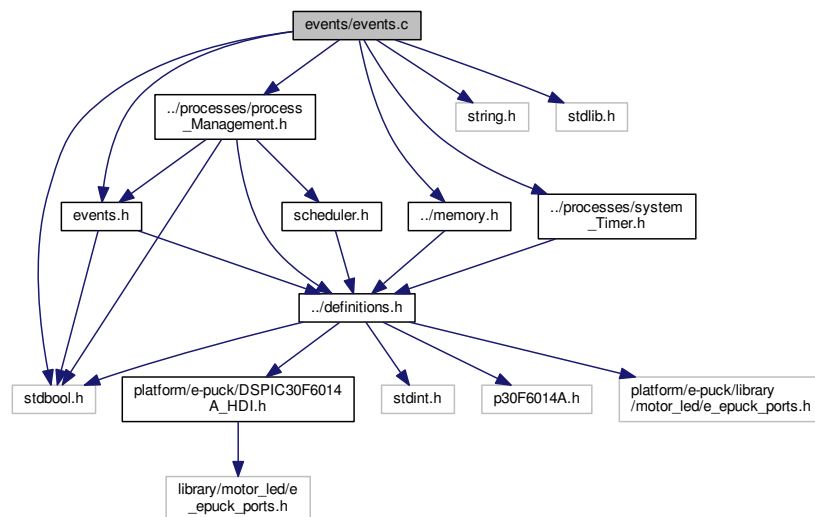
Definition at line 63 of file definitions.h.

8.2 events/events.c File Reference

defines functions to create, (un)subscribe, (un)register, and delete events and related handler.

```
#include "events.h"
#include "../processes/process_Management.h"
#include "../processes/system_Timer.h"
#include "../memory.h"
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
```

Include dependency graph for events.c:



Data Structures

- struct [sys_subscribed_process](#)
 A single linked list element containing the ID of a process that is subscribed to a specific event.
- struct [sys_registered_event](#)
 A single linked element containing a registered event and its subscribers.

Functions

- `sys_registered_event * Sys_Find_Event (uint eventID)`
- `bool Sys_Send_Event (uint eventID, void *data, uint data_size)`
- `bool Sys_Send_IntEvent (uint eventID, uint data)`
- `bool Sys_Register_Event (uint eventID)`
- `bool Sys_Subscribe_to_Event (uint eventID, uint pid, pEventHandlerFunction handler, pConditionFunction condition)`
- `void Sys_Unregister_Event (uint eventID)`
- `void Sys_Unsubscribe_from_Event (uint eventID, uint pid)`
- `void Sys_Unsubscribe_Handler_from_Event (uint eventID, pEventHandlerFunction func, uint pid)`
- `bool Sys_IsEventRegistered (uint eventID)`
- `void Sys_Unsubscribe_Process (uint pid)`

Variables

- `sys_registered_event * registered_events = 0`

8.2.1 Detailed Description

defines functions to create, (un)subscribe, (un)register, and delete events and related handler.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

23 March 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.2.2 Function Documentation

8.2.2.1 `sys_registered_event * Sys_Find_Event (uint eventID)`

This function returns the data structure of an event if the eventID was registered otherwise it's 0.

Parameters

in	<i>eventID</i>	ID of the event
----	----------------	-----------------

Returns

pointer to the data structure of the found event (or 0 if it wasn't found)

Definition at line 307 of file events.c.

8.2.2.2 `bool Sys_IsEventRegistered (uint eventID)`

returns true if the event was registered

Parameters

in	<i>eventID</i>	ID of the event
----	----------------	-----------------

Returns

is the event registered?

Definition at line 328 of file events.c.

8.2.2.3 bool Sys_Register_Event (uint *eventID*)

This function registers an new event. The registration tells the operating system that this event can occur.

Parameters

in	<i>eventID</i>	ID of the event
----	----------------	-----------------

Returns

was it successful.

Definition at line 100 of file events.c.

8.2.2.4 bool Sys_Send_Event (uint *eventID*, void * *data*, uint *data_size*)

This function sends an event to all subscribers.

Parameters

in	<i>eventID</i>	ID of the event
in	<i>data</i>	pointer to the data that want to be sent as an event
in	<i>data_size</i>	size of the data in bytes

Returns

true if it was successful.

Definition at line 61 of file events.c.

8.2.2.5 bool Sys_Send_IntEvent (uint *eventID*, uint *data*) [inline]

This function sends an integer (16-bit) to all subscribers.

Parameters

in	<i>eventID</i>	ID of the event
in	<i>data</i>	integer value that should be sent as an event

Returns

was it successful.

Definition at line 88 of file events.c.

8.2.2.6 bool Sys_Subscribe_to_Event (uint *eventID*, uint *pid*, pEventHandlerFunction *handler*, pConditionFunction *condition*)

This function subscribes a specific handler function to an process and a specific event

Parameters

in	<i>eventID</i>	ID of the event
in	<i>pid</i>	ID of the process
in	<i>handler</i>	pointer to the function that should handle the event data
in	<i>condition</i>	pointer to the function that decides if the handler should be executed or not

Returns

was it successful.

Definition at line 142 of file events.c.

8.2.2.7 void Sys_Unregister_Event (uint eventID)

This function unregisters an event

Parameters

in	<i>eventID</i>	ID of the event
----	----------------	-----------------

Definition at line 187 of file events.c.

8.2.2.8 void Sys_Unsubscribe_from_Event (uint eventID, uint pid)

This function unsubscribes an event

Parameters

in	<i>eventID</i>	ID of the event
in	<i>pid</i>	ID of the process

Definition at line 238 of file events.c.

8.2.2.9 void Sys_Unsubscribe_Handler_from_Event (uint eventID, pEventHandlerFunction func, uint pid)

This function only unsubscribes a specific handler function

Parameters

in	<i>eventID</i>	ID of the event
in	<i>func</i>	pointer to the handler function
in	<i>pid</i>	ID of the process

Definition at line 273 of file events.c.

8.2.2.10 void Sys_Unsubscribe_Process (uint pid)

unsubscribes all events that were subscribed to a process

Parameters

in	<i>pid</i>	process identifier
----	------------	--------------------

Definition at line 347 of file events.c.

8.2.3 Variable Documentation**8.2.3.1 sys_registered_event* registered_events = 0**

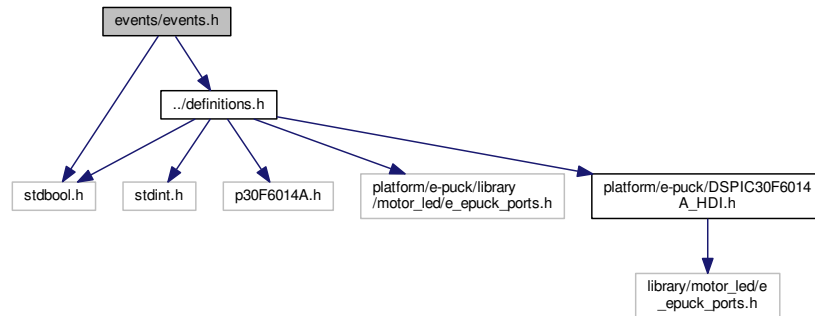
pointer to the List of registered events

Definition at line 48 of file events.c.

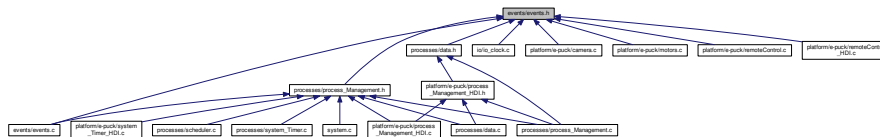
8.3 events/events.h File Reference

declares functions to create, (un)subscribe, (un)register, and delete events and related handler.

```
#include <stdbool.h>
#include "../definitions.h"
Include dependency graph for events.h:
```



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



Data Structures

- struct [sys_event_data](#)
It is a single linked list element and contains data of an occurred event.

Typedefs

- typedef bool(* [pEventHandlerFunction](#)) (uint, uint, [sys_event_data](#) *)
Event handler function pointer type (process id, event id, received data)
- typedef bool(* [pConditionFunction](#)) (void *)
Condition function pointer type.

Functions

- bool [Sys_Send_Event](#) (uint eventID, void *data, uint data_size)
- bool [Sys_Send_IntEvent](#) (uint eventID, uint data)
- bool [Sys_Register_Event](#) (uint eventID)
- void [Sys_Unregister_Event](#) (uint eventID)
- bool [Sys_Subscribe_to_Event](#) (uint eventID, uint pid, [pEventHandlerFunction](#) handler, [pConditionFunction](#) condition)
- void [Sys_Unsubscribe_from_Event](#) (uint eventID, uint pid)
- void [Sys_Unsubscribe_Process](#) (uint pid)
- bool [Sys_IsEventRegistered](#) (uint eventID)

8.3.1 Detailed Description

declares functions to create, (un)subscribe, (un)register, and delete events and related handler.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

23 March 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.3.2 Typedef Documentation

8.3.2.1 typedef bool(* pConditionFunction) (void *)

Condition function pointer type.

This function points to a condition function, which defines if an event handler should be executed or not.

Definition at line 114 of file events.h.

8.3.2.2 typedef bool(* pEventHandlerFunction) (uint, uint, sys_event_data *)

Event handler function pointer type (process id, event id, received data)

This function points to an event handler function, which processes incoming events and its data.

Definition at line 107 of file events.h.

8.3.3 Function Documentation

8.3.3.1 bool Sys_IsEventRegistered (uint *eventID*)

returns true if the event was registered

Parameters

in	<i>eventID</i>	ID of the event
----	----------------	-----------------

Returns

is the event registered?

Definition at line 328 of file events.c.

8.3.3.2 bool Sys_Register_Event (uint *eventID*)

This function registers an new event. The registration tells the operating system that this event can occur.

Parameters

in	<i>eventID</i>	ID of the event
----	----------------	-----------------

Returns

was it successful.

Definition at line 100 of file events.c.

8.3.3.3 bool Sys_Send_Event (uint eventID, void * data, uint data_size)

This function sends an event to all subscribers.

Parameters

in	<i>eventID</i>	ID of the event
in	<i>data</i>	pointer to the data that want to be sent as an event
in	<i>data_size</i>	size of the data in bytes

Returns

true if it was successful.

Definition at line 61 of file events.c.

8.3.3.4 bool Sys_Send_IntEvent (uint eventID, uint data) [inline]

This function sends an integer (16-bit) to all subscribers.

Parameters

in	<i>eventID</i>	ID of the event
in	<i>data</i>	integer value that should be sent as an event

Returns

was it successful.

Definition at line 88 of file events.c.

8.3.3.5 bool Sys_Subscribe_to_Event (uint eventID, uint pid, pEventHandlerFunction handler, pConditionFunction condition)

This function subscribes a specific handler function to an process and a specific event

Parameters

in	<i>eventID</i>	ID of the event
in	<i>pid</i>	ID of the process
in	<i>handler</i>	pointer to the function that should handle the event data
in	<i>condition</i>	pointer to the function that decides if the handler should be executed or not

Returns

was it successful.

Definition at line 142 of file events.c.

8.3.3.6 void Sys_Unregister_Event (uint eventID)

This function unregisters an event

Parameters

in	<i>eventID</i>	ID of the event
----	----------------	-----------------

Definition at line 187 of file events.c.

8.3.3.7 void Sys_Unsubscribe_from_Event (uint *eventID*, uint *pid*)

This function unsubscribes an event

Parameters

in	<i>eventID</i>	ID of the event
in	<i>pid</i>	ID of the process

Definition at line 238 of file events.c.

8.3.3.8 void Sys_Unsubscribe_Process (uint *pid*)

unsubscribes all events that were subscribed to a process

Parameters

in	<i>pid</i>	process identifier
----	------------	--------------------

Definition at line 347 of file events.c.

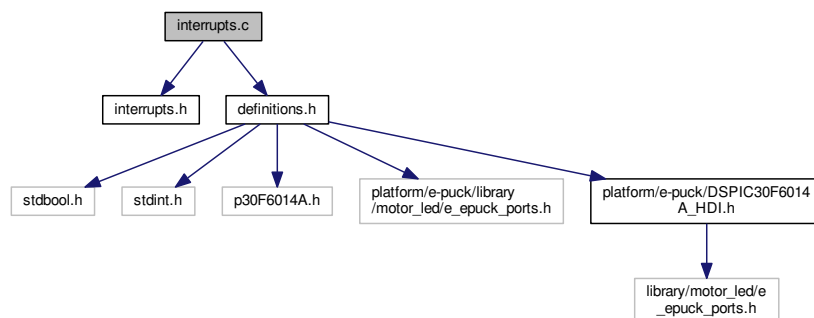
8.4 interrupts.c File Reference

It defines the functions to create atomic sections.

```
#include "interrupts.h"
```

```
#include "definitions.h"
```

Include dependency graph for interrupts.c:

**Functions**

- void [Sys_Start_AtomicSection](#) ()
- void [Sys_End_AtomicSection](#) ()

8.4.1 Detailed Description

It defines the functions to create atomic sections.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

To protect sections of code from any interruptions one has to use the following code:

```
// do something
Sys_Start_AtomicSection();

//do something which should not be interrupted
Sys_End_AtomicSection();

// do something else
```

8.4.2 Function Documentation**8.4.2.1 void Sys_End_AtomicSection (void) [inline]**

This Function ends an atomic section. This means the code afterwards can be interrupted by a interrupt.

Precondition

[Sys_Start_AtomicSection\(\)](#) must have been called.

Definition at line 56 of file interrupts.c.

8.4.2.2 void Sys_Start_AtomicSection (void) [inline]

This Function starts an atomic section. This means the code afterwards cannot be interrupted by any interrupt.

Note

This function can be called within an atomic section. However, it doesn't change the behaviour when called within an atomic section. To end an atomic section, [Sys_End_AtomicSection\(\)](#) must be called as often as [Sys_Start_AtomicSection\(\)](#) was called.

Postcondition

[Sys_End_AtomicSection\(\)](#) must be called to execute any interrupt that happened or will happen.

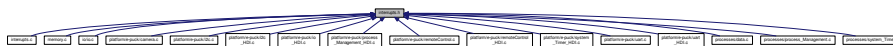
< maximum interrupt priority

Definition at line 42 of file interrupts.c.

8.5 interrupts.h File Reference

It declares interrupt priority levels and functions to create atomic sections.

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



Macros

- `#define SYS_IRQP_MAX 7`
- `#define SYS_IRQP_SYSTEM_TIMER 2`
- `#define SYS_IRQP_IO_TIMER 3`
- `#define SYS_IRQP_UART1 4`
- `#define SYS_IRQP_UART2 4`
- `#define SYS_IRQP_I2C 5`
- `#define SYS_IRQP_REMOTECONTROL 4`
- `#define SYS_IRQP_CAMERA_PIXEL 5`
- `#define SYS_IRQP_CAMERA_LINE 6`
- `#define SYS_IRQP_CAMERA_FRAME 7`

Functions

- `void Sys_Start_AtomicSection (void)`
- `void Sys_End_AtomicSection (void)`

8.5.1 Detailed Description

It declares interrupt priority levels and functions to create atomic sections.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

{03 September 2015}

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.5.2 Macro Definition Documentation

8.5.2.1 `#define SYS_IRQP_CAMERA_FRAME 7`

interrupt priority for the camera frame interrupt

Definition at line 35 of file `interrupts.h`.

8.5.2.2 #define SYS_IRQP_CAMERA_LINE 6

interrupt priority for the camera line interrupt

Definition at line 34 of file interrupts.h.

8.5.2.3 #define SYS_IRQP_CAMERA_PIXEL 5

interrupt priority for the camera pixel interrupt

Definition at line 33 of file interrupts.h.

8.5.2.4 #define SYS_IRQP_I2C 5

interrupt priority for the I2C interrupt

Definition at line 29 of file interrupts.h.

8.5.2.5 #define SYS_IRQP_IO_TIMER 3

interrupt priority for the I/O timer interrupt

Definition at line 24 of file interrupts.h.

8.5.2.6 #define SYS_IRQP_MAX 7

maximum interrupt priority

Definition at line 20 of file interrupts.h.

8.5.2.7 #define SYS_IRQP_REMOTECONTROL 4

interrupt priority for the remote control interrupt

Definition at line 31 of file interrupts.h.

8.5.2.8 #define SYS_IRQP_SYSTEM_TIMER 2

interrupt priority for the system timer interrupt

Definition at line 22 of file interrupts.h.

8.5.2.9 #define SYS_IRQP_UART1 4

interrupt priority for the UART1 interrupt

Definition at line 26 of file interrupts.h.

8.5.2.10 #define SYS_IRQP_UART2 4

interrupt priority for the UART2 interrupt

Definition at line 27 of file interrupts.h.

8.5.3 Function Documentation

8.5.3.1 void Sys_End_AtomicSection (void) [inline]

This Function ends an atomic section. This means the code afterwards can be interrupted by a interrupt.

Precondition

[Sys_Start_AtomicSection\(\)](#) must have been called.

Definition at line 56 of file interrupts.c.

8.5.3.2 void Sys_Start_AtomicSection (void) [inline]

This Function starts an atomic section. This means the code afterwards cannot be interrupted by any interrupt.

Note

This function can be called within an atomic section. However, it doesn't change the behaviour when called within an atomic section. To end an atomic section, [Sys_End_AtomicSection\(\)](#) must be called as often as [Sys_Start_AtomicSection\(\)](#) was called.

Postcondition

[Sys_End_AtomicSection\(\)](#) must be called to execute any interrupt that happened or will happen.

< maximum interrupt priority

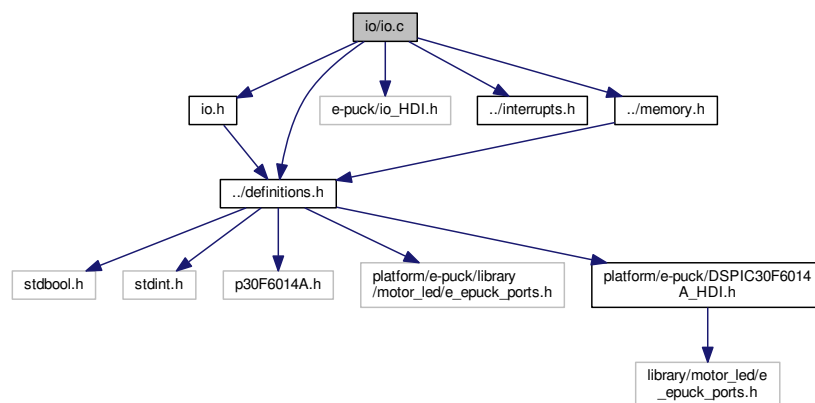
Definition at line 42 of file interrupts.c.

8.6 io/io.c File Reference

It defines functions to control the IO timer and to (un)register IO Handler.

```
#include "io.h"
#include "../definitions.h"
#include "e-puck/io_HDI.h"
#include "../interrupts.h"
#include "../memory.h"
```

Include dependency graph for io.c:



Functions

- void [Sys_Init_IOManagement](#) (void)
- void [Sys_Start_IOManagement](#) (void)
- void [Sys_Stop_IOManagement](#) (void)
- void [Sys_Stop_IOTimer](#) ()
- void [Sys_Continue_IOTimer](#) ()
- void [Sys_Reset_IOTimer](#) ()
- void [Sys_Disable_IOTimerInterrupt](#) ()
- void [Sys_Enable_IOTimerInterrupt](#) ()

- void `Sys_Force_IOTimerInterrupt` ()
- bool `Sys_Register_IOHandler` (pFunction func)
- void `Sys_Unregister_IOHandler` (pFunction func)

8.6.1 Detailed Description

It defines functions to control the IO timer and to (un)register IO Handler.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

10 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.6.2 Function Documentation

8.6.2.1 void Sys_Continue_IOTimer (void) [inline]

This function continues the I/O Timer. Note that the timer continues to count where it stops.

Definition at line 66 of file io.c.

8.6.2.2 void Sys_Disable_IOTimerInterrupt (void) [inline]

This function disables the I/O Timer interrupt. Note that the timer still continues to count.

Definition at line 84 of file io.c.

8.6.2.3 void Sys_Enable_IOTimerInterrupt (void) [inline]

This function enables the I/O Timer interrupt.

Definition at line 93 of file io.c.

8.6.2.4 void Sys_Force_IOTimerInterrupt (void) [inline]

This function forces a new I/O Timer interrupt even if the timer hasn't reached its threshold.

Definition at line 102 of file io.c.

8.6.2.5 void Sys_Init_IOManagement (void) [inline]

This function initialises the I/O Timer and therefore the I/O Management.

Definition at line 30 of file io.c.

8.6.2.6 bool Sys_Register_IOHandler (pFunction func)

This function registers a new I/O handler which is executed every time the I/O timer interrupt occurs.

Parameters

<i>in</i>	<i>func</i>	pointer to the function that should be executed by the I/O timer periodically
-----------	-------------	---

Returns

true if it was successful

Definition at line 113 of file io.c.

8.6.2.7 void Sys_Reset_IOTimer (void) [inline]

This function sets the I/O Timer counter to 0 and the I/O timer needs the full time duration to throw the interrupt.

Definition at line 75 of file io.c.

8.6.2.8 void Sys_Start_IOManagement (void) [inline]

This function starts the I/O Timer and therefore the I/O Management.

Definition at line 39 of file io.c.

8.6.2.9 void Sys_Stop_IOManagement (void) [inline]

This function stops the I/O Timer and therefore the I/O Management.

Definition at line 48 of file io.c.

8.6.2.10 void Sys_Stop_IOTimer (void) [inline]

This function stops the I/O Timer.

Definition at line 57 of file io.c.

8.6.2.11 void Sys_Unregister_IOHandler (pFunction func)

This function unregisters a I/O handler identified by its function address.

Parameters

<i>in</i>	<i>func</i>	pointer to the function that should be executed by the I/O timer periodically
-----------	-------------	---

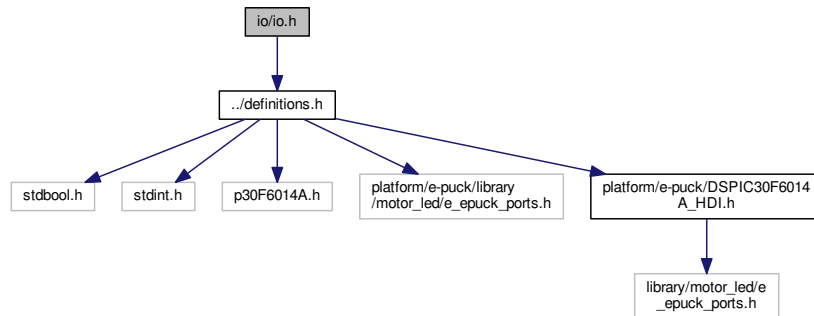
Definition at line 147 of file io.c.

8.7 io/io.h File Reference

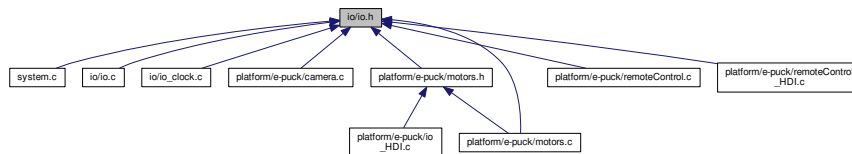
It declares functions to control the IO timer and to (un)register IO Handler.

```
#include "../definitions.h"
```

Include dependency graph for io.h:



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



Functions

- void `Sys_Init_IOManagement` (void)
- void `Sys_Start_IOManagement` (void)
- void `Sys_Stop_IOManagement` (void)
- void `Sys_Stop_IOTimer` (void)
- void `Sys_Continue_IOTimer` (void)
- void `Sys_Reset_IOTimer` (void)
- void `Sys_Disable_IOTimerInterrupt` (void)
- void `Sys_Enable_IOTimerInterrupt` (void)
- void `Sys_Force_IOTimerInterrupt` (void)
- bool `Sys_Register_IOHandler` (pFunction func)
- void `Sys_Unregister_IOHandler` (pFunction func)

8.7.1 Detailed Description

It declares functions to control the IO timer and to (un)register IO Handler.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

28 July 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.7.2 Function Documentation**8.7.2.1 void Sys_Continue_IOTimer (void) [inline]**

This function continues the I/O Timer. Note that the timer continues to count where it stops.

Definition at line 66 of file io.c.

8.7.2.2 void Sys_Disable_IOTimerInterrupt (void) [inline]

This function disables the I/O Timer interrupt. Note that the timer still continues to count.

Definition at line 84 of file io.c.

8.7.2.3 void Sys_Enable_IOTimerInterrupt (void) [inline]

This function enables the I/O Timer interrupt.

Definition at line 93 of file io.c.

8.7.2.4 void Sys_Force_IOTimerInterrupt (void) [inline]

This function forces a new I/O Timer interrupt even if the timer hasn't reached its threshold.

Definition at line 102 of file io.c.

8.7.2.5 void Sys_Init_IOManagement (void) [inline]

This function initialises the I/O Timer and therefore the I/O Management.

Definition at line 30 of file io.c.

8.7.2.6 bool Sys_Register_IOHandler (pFunction func)

This function registers a new I/O handler which is executed every time the I/O timer interrupt occurs.

Parameters

<i>in</i>	<i>func</i>	pointer to the function that should be executed by the I/O timer periodically
-----------	-------------	---

Returns

true if it was successful

Definition at line 113 of file io.c.

8.7.2.7 void Sys_Reset_IOTimer (void) [inline]

This function sets the I/O Timer counter to 0 and the I/O timer needs the full time duration to throw the interrupt.

Definition at line 75 of file io.c.

8.7.2.8 void Sys_Start_IOManagement (void) [inline]

This function starts the I/O Timer and therefore the I/O Management.

Definition at line 39 of file io.c.

8.7.2.9 void Sys_Stop_IOManagement (void) [inline]

This function stops the I/O Timer and therefore the I/O Management.

Definition at line 48 of file io.c.

8.7.2.10 void Sys_Stop_IOTimer (void) [inline]

This function stops the I/O Timer.

Definition at line 57 of file io.c.

8.7.2.11 void Sys_Unregister_IOHandler (pFunction func)

This function unregisters a I/O handler identified by its function address.

Parameters

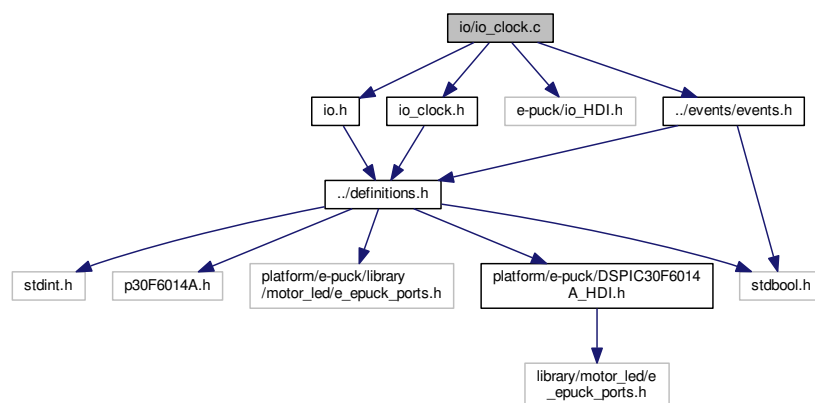
<i>in</i>	<i>func</i>	pointer to the function that should be executed by the I/O timer periodically
-----------	-------------	---

Definition at line 147 of file io.c.

8.8 io/io_clock.c File Reference

It defines the system clock that provides a continuous time value (granulation of 1 ms).

```
#include "io.h"
#include "io_clock.h"
#include "e-puck/io_HDI.h"
#include "../events/events.h"
Include dependency graph for io_clock.c:
```

**Functions**

- void [Sys_SystemClock_Counter](#) (void)
- void [Sys_Init_Clock](#) ()
- void [Sys_Init_SystemTime](#) ()
- uint32 [Sys_Get_SystemTime](#) ()
- uint32 [Sys_Get_SystemClock](#) ()

8.8.1 Detailed Description

It defines the system clock that provides a continuous time value (granulation of 1 ms).

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

28 July 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.8.2 Function Documentation

8.8.2.1 uint32 Sys_Get_SystemClock(void) [inline]

returns the system clock/time in milliseconds

Returns

uint32 time that has passed since OpenSwarm was started

Definition at line 77 of file io_clock.c.

8.8.2.2 uint32 Sys_Get_SystemTime(void) [inline]

Renaming of the function [Sys_Get_SystemClock\(\)](#).

Returns

time that has passed since OpenSwarm was started (uint32)

Definition at line 67 of file io_clock.c.

8.8.2.3 void Sys_Init_Clock(void) [inline]

This function initialises the system clock which is in principle a counter that indicates passed milli seconds. < ID of the event that signals 1ms timer ticks

Definition at line 29 of file io_clock.c.

8.8.2.4 void Sys_Init_SystemTime(void) [inline]

Renaming of the function [Sys_Init_Clock\(\)](#).

Definition at line 39 of file io_clock.c.

8.8.2.5 void Sys_SystemClock_Counter()

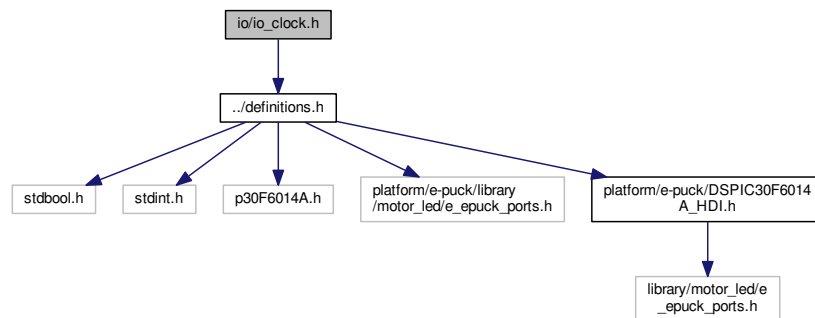
This function calculates the system clock tick and increases the counter if a millisecond passed. < ID of the event that signals 1ms timer ticks

Definition at line 48 of file io_clock.c.

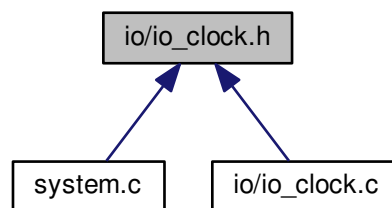
8.9 io/io_clock.h File Reference

It declares the system clock that provides a continuous time value (granulation of 1 ms).

```
#include "../definitions.h"
Include dependency graph for io_clock.h:
```



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



Functions

- void [Sys_Init_Clock](#) (void)
- void [Sys_Init_SystemTime](#) (void)
- uint32 [Sys_Get_SystemTime](#) (void)
- uint32 [Sys_Get_SystemClock](#) (void)

8.9.1 Detailed Description

It declares the system clock that provides a continuous time value (granulation of 1 ms).

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

28 July 2015

Copyrightadapted FreeBSD License (see <http://openswarm.org/license>)**8.9.2 Function Documentation****8.9.2.1 uint32 Sys_Get_SystemClock (void) [inline]**

returns the system clock/time in milliseconds

Returns

uint32 time that has passed since OpenSwarm was started

Definition at line 77 of file io_clock.c.

8.9.2.2 uint32 Sys_Get_SystemTime (void) [inline]Renaming of the function [Sys_Get_SystemClock\(\)](#).**Returns**

time that has passed since OpenSwarm was started (uint32)

Definition at line 67 of file io_clock.c.

8.9.2.3 void Sys_Init_Clock (void) [inline]

This function initialises the system clock which is in principle a counter that indicates passed milliseconds. < ID of the event that signals 1ms timer ticks

Definition at line 29 of file io_clock.c.

8.9.2.4 void Sys_Init_SystemTime (void) [inline]Renaming of the function [Sys_Init_Clock\(\)](#).

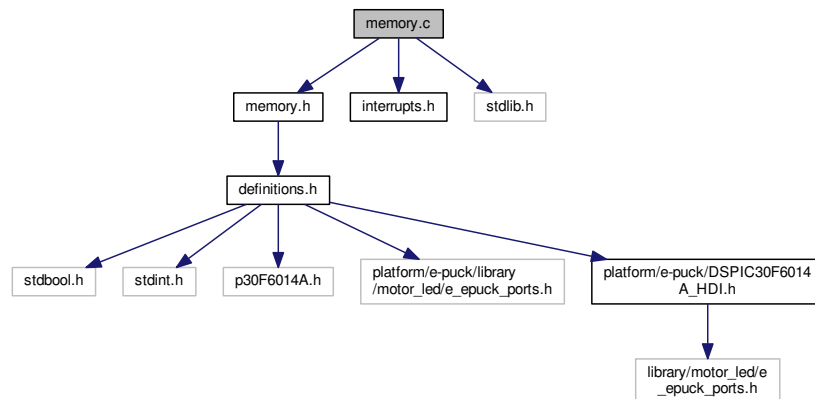
Definition at line 39 of file io_clock.c.

8.10 memory.c File Reference

It defines functions to allocate, free, and copy memory.

```
#include "memory.h"
#include "interrupts.h"
#include <stdlib.h>
```

Include dependency graph for memory.c:



Functions

- void * [Sys_Malloc](#) (uint length)
- void [Sys_Free](#) (void *data)
- void [Sys_Memcpy](#) (void *source_i, void *destination_o, uint length)

8.10.1 Detailed Description

It defines functions to allocate, free, and copy memory.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

{05 September 2015}

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.10.2 Function Documentation

8.10.2.1 void Sys_Free (void * data)

This Function frees dynamic allocated memory. This freeing is performed as atomic action.

Functions

- void * [Sys_Malloc](#) (uint length)
- void [Sys_Free](#) (void *)
- void [Sys_Memcpy](#) (void *source, void *destination, uint length)

8.11.1 Detailed Description

It declares functions to allocate, free, and copy memory.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

{05 September 2015}

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.11.2 Function Documentation

8.11.2.1 void Sys_Free (void * data)

This Function frees dynamic allocated memory. This freeing is performed as atomic action.

Parameters

<i>data</i>	pointer to memory that should be freed.
-------------	---

Definition at line 43 of file memory.c.

8.11.2.2 void* Sys_Malloc (uint length)

This Function allocates memory of the size **length**. This allocation is performed as atomic action.

Parameters

<i>length</i>	value how many bytes should be allocated
---------------	--

Returns

pointer to the allocated memory

Definition at line 25 of file memory.c.

8.11.2.3 void Sys_Memcpy (void * source_i, void * destination_o, uint length)

Function to copies memory of the size **length** from **source_i** to **destination_o**. This copying is performed as atomic action.

Parameters

in	<i>source_i</i>	pointer to the source
out	<i>destination_o</i>	pointer to the destination
in	<i>length</i>	size of the memory that has to be copied

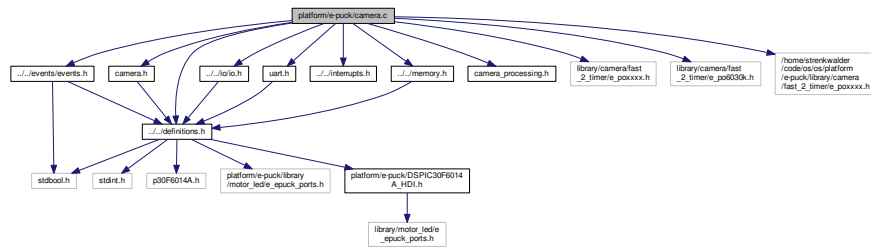
Definition at line 61 of file memory.c.

8.12 platform/e-puck/camera.c File Reference

It defines functions to process data retrieved by a camera.

```
#include "camera.h"
#include "../io/io.h"
#include "uart.h"
#include "../definitions.h"
#include "../events/events.h"
#include "../interrupts.h"
#include "../memory.h"
#include "camera_processing.h"
#include "library/camera/fast_2_timer/e_poxxxx.h"
#include "library/camera/fast_2_timer/e_po6030k.h"
```

Include dependency graph for camera.c:



Macros

- #define FRAME_WIDTH 10
- #define FRAME_HEIGHT 10
- #define CAMERA_I2C_ADDRESS 0xDC
- #define RED_MAX 0x0C1C
- #define GREEN_MAX 0x189C
- #define BLUE_MAX 0x0C1C
- #define RED_THRESHOLD 0x060E
- #define GREEN_THRESHOLD 0x0E4E
- #define BLUE_THRESHOLD 0x060E
- #define CAM_WIDTH 160
- #define CAM_HEIGHT 160
- #define CAM_ZOOM_X 8
- #define CAM_ZOOM_Y 8
- #define CAM_W_SIZE 20
- #define CAM_H_SIZE 20
- #define CP_WI 120
- #define CP_RI 80
- #define CP_GI 80
- #define CP_BI 100
- #define COLOUR_THRESHOLD 766

Functions

- void [Sys_Process_newPixel](#) (void)
- void [Sys_Process_newLine](#) (void)
- void [Sys_Process_newFrame](#) (void)
- void [Sys_Camera_PreProcessor](#) (void)
- void [Sys_Init_Camera](#) ()
- void [Sys_Start_Camera](#) ()
- void [Sys_Set_Preprocessing](#) (pCameraPreProcessor func)

8.12.1 Detailed Description

It defines functions to process data retrieved by a camera.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org
Yuri Kaszubowski Lopes yurikazuba@gmail.com

Version

1.0

Date

27 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

Todo The used functions from the e-puck library are very time and computational intensive. These function can be rewritten to decrease the processing load.

8.12.2 Macro Definition Documentation

8.12.2.1 `#define BLUE_MAX 0x0C1C`

maximum value for received blue

Definition at line 42 of file camera.c.

8.12.2.2 `#define BLUE_THRESHOLD 0x060E`

threshold value for received blue

Definition at line 45 of file camera.c.

8.12.2.3 `#define CAM_H_SIZE 20`

post scale height frame

Definition at line 79 of file camera.c.

8.12.2.4 `#define CAM_HEIGHT 160`

height of the camera input frame

Definition at line 75 of file camera.c.

8.12.2.5 #define CAM_W_SIZE 20

post scale width frame

Definition at line 78 of file camera.c.

8.12.2.6 #define CAM_WIDTH 160

width of the camera input frame

Definition at line 74 of file camera.c.

8.12.2.7 #define CAM_ZOOM_X 8

zoom factor to scale the frame

Definition at line 76 of file camera.c.

8.12.2.8 #define CAM_ZOOM_Y 8

zoom factor to scale the frame

Definition at line 77 of file camera.c.

8.12.2.9 #define CAMERA_I2C_ADDRESS 0xDC

I2C address of the camera

Definition at line 38 of file camera.c.

8.12.2.10 #define COLOUR_THRESHOLD 766

threshold to decide if a colour pixel has been measured

Definition at line 528 of file camera.c.

8.12.2.11 #define CP_BI 100

blue factor to process and calibrate the camera

Definition at line 527 of file camera.c.

8.12.2.12 #define CP_GI 80

green factor to process and calibrate the camera

Definition at line 526 of file camera.c.

8.12.2.13 #define CP_RI 80

red factor to process and calibrate the camera

Definition at line 525 of file camera.c.

8.12.2.14 #define CP_WI 120

whitness factor to process and calibrate the camera

Definition at line 524 of file camera.c.

8.12.2.15 #define FRAME_HEIGHT 10

Height of the subframe of the image

Definition at line 37 of file camera.c.

8.12.2.16 #define FRAME_WIDTH 10

Width of the subframe of the image

Definition at line 36 of file camera.c.

8.12.2.17 #define GREEN_MAX 0x189C

maximum value for received green

Definition at line 41 of file camera.c.

8.12.2.18 #define GREEN_THRESHOLD 0x0E4E

threshold value for received green

Definition at line 44 of file camera.c.

8.12.2.19 #define RED_MAX 0x0C1C

maximum value for received red

Definition at line 40 of file camera.c.

8.12.2.20 #define RED_THRESHOLD 0x060E

threshold value for received red

Definition at line 43 of file camera.c.

8.12.3 Function Documentation**8.12.3.1 void Sys_Camera_PreProcessor (void)**

This function processes an incoming camera frame and emits events according to used algorithm.

Todo rewrite the camera to computational less intensive functions

< ID of the event that is emitted by the camera

Definition at line 538 of file camera.c.

8.12.3.2 void Sys_Init_Camera (void)

This function initialises the camera using e-puck library from Subversion at [svn://svn.gna.org/svn/e-puck/trunk](http://svn.gna.org/svn/e-puck/trunk)

Todo rewrite the camera to computational less intensive functions

< ID of the event that is emitted by the camera

< width of the camera input frame

< height of the camera input frame

< width of the camera input frame

< height of the camera input frame

< zoom factor to scale the frame

< zoom factor to scale the frame

Definition at line 89 of file camera.c.

8.12.3.3 void Sys_Process_newFrame (void) [inline]

8.12.3.4 `void Sys_Process_newLine (void) [inline]`

8.12.3.5 `void Sys_Process_newPixel (void) [inline]`

8.12.3.6 `void Sys_Set_Preprocessing (pCameraPreProcessor func)`

Defines a preprocessor callback functions to process the frame.

Parameters

<code>in</code>	<code>func</code>	camera preprocessor which computes events out of the raw image
-----------------	-------------------	--

Definition at line 307 of file camera.c.

8.12.3.7 `void Sys_Start_Camera (void)`

This function starts the capturing using e-puck library from Subversion at [svn://svn.gna.org/svn/e-puck/trunk](http://svn.gna.org/svn/e-puck/trunk)

Todo rewrite the camera to computational less intensive functions

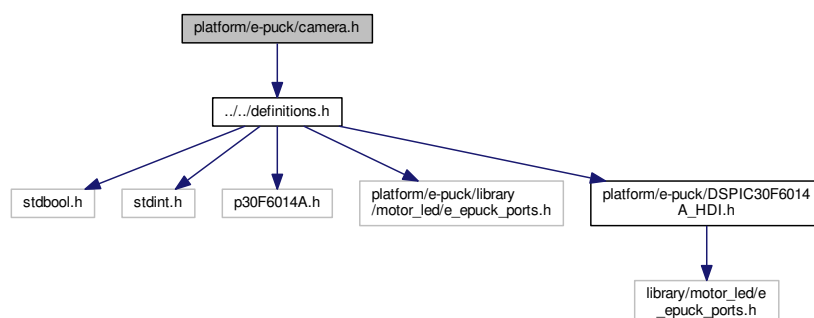
Definition at line 287 of file camera.c.

8.13 platform/e-puck/camera.h File Reference

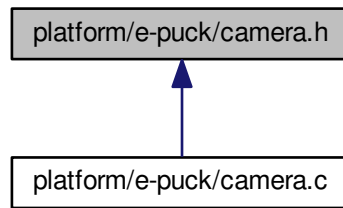
It declares functions to process data retrieved by a camera.

```
#include "../..definitions.h"
```

Include dependency graph for camera.h:



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



Data Structures

- struct [sys_rgb_pixel](#)

This bitfield contains the structure of a received camera pixel.

Macros

- #define [SYS_MAX_RED](#) 0b00011111;
- #define [SYS_MAX_GREEN](#) 0b00111111;
- #define [SYS_MAX_BLUE](#) 0b00011111;

Typedefs

- typedef void(* [pCameraPreProcessor](#)) ([sys_rgb_pixel](#) **frame, [uint16](#) width, [uint16](#) height)

Functions

- void [Sys_Init_Camera](#) (void)
- void [Sys_Start_Camera](#) (void)
- void [Sys_Set_Preprocessing](#) ([pCameraPreProcessor](#) func)
- [sys_rgb_pixel](#) * [getFinishedFrame](#) ()
- bool [isNewFrameAvailable](#) ()

8.13.1 Detailed Description

It declares functions to process data retrieved by a camera.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

27 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.13.2 Macro Definition Documentation**8.13.2.1 #define SYS_MAX_BLUE 0b00011111;**

blue bits received

Definition at line 52 of file camera.h.

8.13.2.2 #define SYS_MAX_GREEN 0b00111111;

green bits received

Definition at line 51 of file camera.h.

8.13.2.3 #define SYS_MAX_RED 0b00011111;

red bits received

Definition at line 50 of file camera.h.

8.13.3 Typedef Documentation**8.13.3.1 typedef void(* pCameraPreProcessor) (sys_rgb_pixel **frame, uint16 width, uint16 height)**

pointer to a camera preprocessor

Definition at line 63 of file camera.h.

8.13.4 Function Documentation**8.13.4.1 sys_rgb_pixel* getFinishedFrame ()****8.13.4.2 bool isNewFrameAvailable ()****8.13.4.3 void Sys_Init_Camera (void)**

This function initialises the camera using e-puck library from Subversion at <svn://svn.gna.org/svn/e-puck/trunk>

Todo rewrite the camera to computational less intensive functions

< ID of the event that is emitted by the camera

< width of the camera input frame

< height of the camera input frame

< width of the camera input frame

< height of the camera input frame

< zoom factor to scale the frame

< zoom factor to scale the frame

Definition at line 89 of file camera.c.

8.13.4.4 void Sys_Set_Preprocessing (pCameraPreProcessor *func*)

Defines a preprocessor callback functions to process the frame.

Parameters

<i>in</i>	<i>func</i>	camera preprocessor which computes events out of the raw image
-----------	-------------	--

Definition at line 307 of file camera.c.

8.13.4.5 void Sys_Start_Camera (void)

This function starts the capturing using e-puck library from Subversion at [svn://svn.gna.org/svn/e-puck/trunk](http://svn.gna.org/svn/e-puck/trunk)

Todo rewrite the camera to computational less intensive functions

Definition at line 287 of file camera.c.

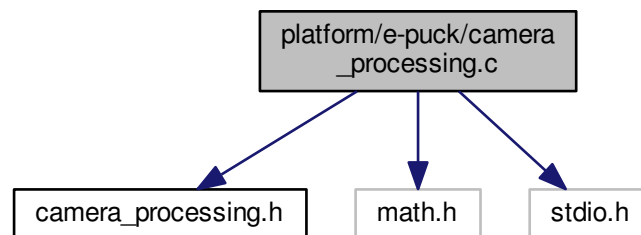
8.14 platform/e-puck/camera_processing.c File Reference

```
#include "camera_processing.h"
```

```
#include <math.h>
```

```
#include <stdio.h>
```

Include dependency graph for camera_processing.c:

**Macros**

- #define [CP_WI](#) 100
- #define [CP_WGB_I](#) 80
- #define [CP_RI](#) 80
- #define [CP_GI](#) 40
- #define [CP_BI](#) 100
- #define [CBP_WI](#) 16
- #define [CBP_RI](#) 11
- #define [CBP_GI](#) 11
- #define [CBP_BI](#) 13
- #define [CBP_DI](#) 2

Functions

- void [convertRGB565ToRGB888](#) (unsigned char rgb565[], unsigned char rgb888[])
- void [getRGB565at](#) (char *buffer, unsigned char rgb585[], int x, int y)
- void [getRGB888at](#) (char *buffer, unsigned char rgb888[], int x, int y)
- char [nearestNeighborRGB](#) (unsigned char *rbg888, char flag)
- char [brushedColorFromRGB565](#) (unsigned char rgb565[], char flag)
- char [getBrushedColorAt](#) (char *buffer, char flag, int x, int y, int w)

Variables

- const unsigned char `colorPositions` [8][4]
- const int `powerTbl` [33] = {0,1,4,9,16,25,36,49,64,81,100,121,144,169,196,225,256,289,324,361,400,441,484,529,576,625,676}
- const unsigned char `colorBrushedPositions` [8][4]

8.14.1 Detailed Description

Author

Yuri Kaszubowski Lopes yurikazuba@gmail.com

Version

1.0

Date

2014

8.14.2 Macro Definition Documentation

8.14.2.1 `#define CBP_BI 13`

Definition at line 72 of file camera_processing.c.

8.14.2.2 `#define CBP_DI 2`

Definition at line 73 of file camera_processing.c.

8.14.2.3 `#define CBP_GI 11`

Definition at line 71 of file camera_processing.c.

8.14.2.4 `#define CBP_RI 11`

Definition at line 70 of file camera_processing.c.

8.14.2.5 `#define CBP_WI 16`

Definition at line 69 of file camera_processing.c.

8.14.2.6 `#define CP_BI 100`

Definition at line 36 of file camera_processing.c.

8.14.2.7 `#define CP_GI 40`

Definition at line 35 of file camera_processing.c.

8.14.2.8 `#define CP_RI 80`

Definition at line 34 of file camera_processing.c.

8.14.2.9 `#define CP_WGB_I 80`

Definition at line 33 of file camera_processing.c.

8.14.2.10 `#define CP_WI 100`

Definition at line 32 of file camera_processing.c.

8.14.3 Function Documentation

8.14.3.1 char brushedColorFromRGB565 (unsigned char *rgb565*[], char *flag*)

Definition at line 86 of file camera_processing.c.

8.14.3.2 void convertRGB565ToRGB888 (unsigned char *rgb565*[], unsigned char *rgb888*[])

Definition at line 15 of file camera_processing.c.

8.14.3.3 char getBrushedColorAt (char * *buffer*, char *flag*, int *x*, int *y*, int *w*)

Definition at line 109 of file camera_processing.c.

8.14.3.4 void getRGB565at (char * *buffer*, unsigned char *rgb585*[], int *x*, int *y*)

Definition at line 21 of file camera_processing.c.

8.14.3.5 void getRGB888at (char * *buffer*, unsigned char *rgb888*[], int *x*, int *y*)

Definition at line 26 of file camera_processing.c.

8.14.3.6 char nearestNeighborRGB (unsigned char * *rbg888*, char *flag*)

Definition at line 50 of file camera_processing.c.

8.14.4 Variable Documentation

8.14.4.1 const unsigned char colorBrushedPositions[8][4]

Initial value:

```
= {
    { 2, 2, 2, 'd' },
    { 2, 11, 13, 'c' },
    { 11, 2, 13, 'm' },
    { 11, 11, 2, 'y' },
    { 2, 2, 13, 'b' },
    { 2, 11, 2, 'g' },
    { 11, 2, 2, 'r' },
    { 16, 16, 16, 'w' }
}
```

Definition at line 74 of file camera_processing.c.

8.14.4.2 const unsigned char colorPositions[8][4]

Initial value:

```
= {
    { 0, 0, 0, 'd' },
    { 0, 40, 100, 'c' },
    { 80, 0, 100, 'm' },
    { 80, 40, 0, 'y' },
    { 0, 0, 100, 'b' },
    { 0, 40, 0, 'g' },
    { 80, 0, 0, 'r' },
    { 100, 80, 80, 'w' }
}
```

Definition at line 37 of file camera_processing.c.

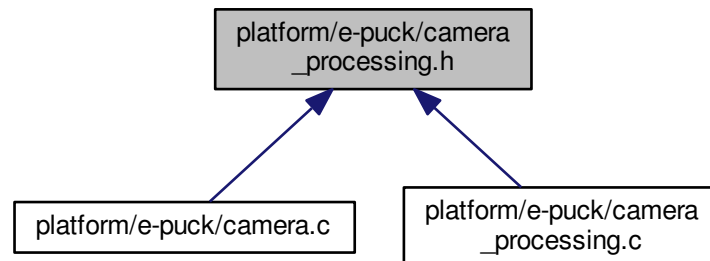
8.14.4.3 const int powerTbl[33] = {0,1,4,9,16,25,36,49,64,81,100,121,144,169,196,225,256,289,324,361,400,441,484,529,576,625,676,729,784,841,900,961,

Definition at line 68 of file camera_processing.c.

8.15 platform/e-puck/camera_processing.h File Reference

External set of functions to assist the use of the camera. (provided by [Shefpuck](#))

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



Functions

- void [convertRGB565ToRGB888](#) (unsigned char rgb565[], unsigned char rgb888[])
- void [getRGB565at](#) (char *buffer, unsigned char rgb585[], int x, int y)
- void [getRGB888at](#) (char *buffer, unsigned char rgb888[], int x, int y)
- char [nearestNeighborRGB](#) (unsigned char *rbg888, char flag)
- char [brushedColorFromRGB565](#) (unsigned char rgb565[], char flag)
- char [getBrushedColorAt](#) (char *buffer, char flag, int x, int y, int w)

8.15.1 Detailed Description

External set of functions to assist the use of the camera. (provided by [Shefpuck](#))

Author

Yuri Kaszubowski Lopes yurikazuba@gmail.com

Version

1.0

Date

2014

8.15.2 Function Documentation

8.15.2.1 char brushedColorFromRGB565 (unsigned char *rgb565*[], char *flag*)

Definition at line 86 of file camera_processing.c.

8.15.2.2 void convertRGB565ToRGB888 (unsigned char *rgb565*[], unsigned char *rgb888*[])

Definition at line 15 of file camera_processing.c.

8.15.2.3 `char getBrushedColorAt (char * buffer, char flag, int x, int y, int w)`

Definition at line 109 of file `camera_processing.c`.

8.15.2.4 `void getRGB565at (char * buffer, unsigned char rgb565[], int x, int y)`

Definition at line 21 of file `camera_processing.c`.

8.15.2.5 `void getRGB888at (char * buffer, unsigned char rgb888[], int x, int y)`

Definition at line 26 of file `camera_processing.c`.

8.15.2.6 `char nearestNeighborRGB (unsigned char * rgb888, char flag)`

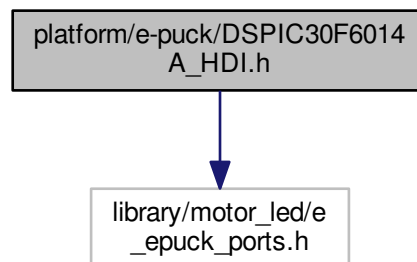
Definition at line 50 of file `camera_processing.c`.

8.16 platform/e-puck/DSPIC30F6014A_HDI.h File Reference

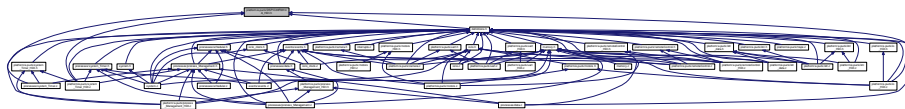
declares e-puck specific types and preprocessor variables

```
#include "library/motor_led/e_epuck_ports.h"
```

Include dependency graph for DSPIC30F6014A_HDI.h:



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



Macros

- `#define ADDRESS_IVT 0x000004`
- `#define ADDRESS_ITV_OSC_FAIL ADDRESS_IVT+2`
- `#define ADDRESS_ITV_ADDRESS_ERROR ADDRESS_IVT+4`
- `#define ADDRESS_ITV_STACK_ERROR ADDRESS_IVT+6`
- `#define ADDRESS_ITV_MATH_ERROR ADDRESS_IVT+8`
- `#define ADDRESS_IVT_T1 0x00001A`
- `#define ADDRESS_AIVT 0x000084`

- #define ADDRESS_AITV_OSC_FAIL ADDRESS_AIVT+2
- #define ADDRESS_AITV_ADDRESS_ERROR ADDRESS_AIVT+4
- #define ADDRESS_AITV_STACK_ERROR ADDRESS_AIVT+6
- #define ADDRESS_AITV_MATH_ERROR ADDRESS_AIVT+8
- #define ADDRESS_AIVT_T1 0x00009A

8.16.1 Detailed Description

declares e-puck specific types and preprocessor variables

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

07 July 2014

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.16.2 Macro Definition Documentation

8.16.2.1 #define ADDRESS_AITV_ADDRESS_ERROR ADDRESS_AIVT+4

Definition at line 73 of file DSPIC30F6014A_HDI.h.

8.16.2.2 #define ADDRESS_AITV_MATH_ERROR ADDRESS_AIVT+8

Definition at line 75 of file DSPIC30F6014A_HDI.h.

8.16.2.3 #define ADDRESS_AITV_OSC_FAIL ADDRESS_AIVT+2

Definition at line 72 of file DSPIC30F6014A_HDI.h.

8.16.2.4 #define ADDRESS_AITV_STACK_ERROR ADDRESS_AIVT+6

Definition at line 74 of file DSPIC30F6014A_HDI.h.

8.16.2.5 #define ADDRESS_AIVT 0x000084

Definition at line 71 of file DSPIC30F6014A_HDI.h.

8.16.2.6 #define ADDRESS_AIVT_T1 0x00009A

Definition at line 76 of file DSPIC30F6014A_HDI.h.

8.16.2.7 #define ADDRESS_ITV_ADDRESS_ERROR ADDRESS_IVT+4

Definition at line 65 of file DSPIC30F6014A_HDI.h.

8.16.2.8 #define ADDRESS_ITV_MATH_ERROR ADDRESS_IVT+8

Definition at line 67 of file DSPIC30F6014A_HDI.h.

8.16.2.9 #define ADDRESS_ITV_OSC_FAIL ADDRESS_IVT+2

Definition at line 64 of file DSPIC30F6014A_HDI.h.

8.16.2.10 #define ADDRESS_ITV_STACK_ERROR ADDRESS_IVT+6

Definition at line 66 of file DSPIC30F6014A_HDI.h.

8.16.2.11 #define ADDRESS_IVT 0x000004

Definition at line 63 of file DSPIC30F6014A_HDI.h.

8.16.2.12 #define ADDRESS_IVT_T1 0x00001A

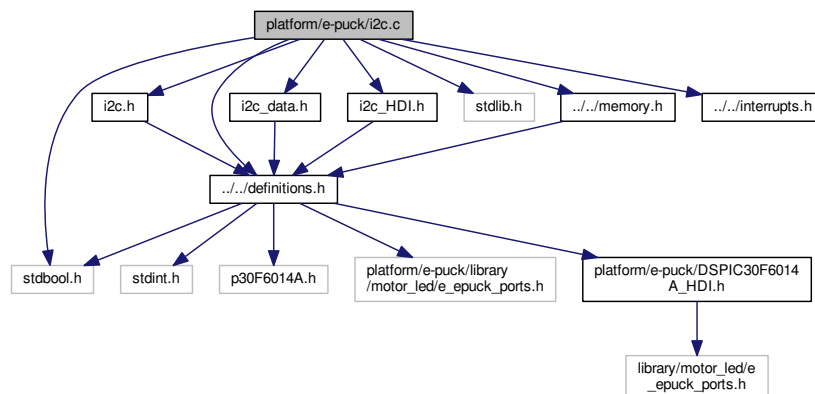
Definition at line 68 of file DSPIC30F6014A_HDI.h.

8.17 platform/e-puck/i2c.c File Reference

It defines functions to read and write on the I2C interface.

```
#include "i2c.h"
#include "i2c_data.h"
#include "i2c_HDI.h"
#include <stdlib.h>
#include <stdbool.h>
#include "../definitions.h"
#include "../memory.h"
#include "../interrupts.h"
```

Include dependency graph for i2c.c:



Functions

- void [Sys_I2C_Send_Start](#) ()
- void [Sys_I2C_Send_Restart](#) (void)
- void [Sys_I2C_Send_Stop](#) (void)
- void [Sys_I2C_Send_ACK](#) (void)
- void [Sys_I2C_Send_NACK](#) (void)
- void [Sys_I2C_Start_Reading](#) (void)
- char [Sys_I2C_ReadByte](#) (void)
- void [Sys_I2C_WriteByte](#) (uint8 byte)

- void `Sys_Init_I2C` ()
- void `Sys_Start_I2C` ()
- void `Sys_Pause_I2C` ()
- void `Sys_Contine_I2C` ()
- void `Sys_Stop_I2C` ()
- void `Sys_I2C_SentBytes` (uint8 address, uint8 *bytes, uint length)
- void `Sys_I2C_Read` (uint8 address, uint8 *intern_address, uint length, pByteFunction bytehandler)

8.17.1 Detailed Description

It defines functions to read and write on the I2C interface.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

10 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.17.2 Function Documentation

8.17.2.1 void Sys_Contine_I2C (void) [inline]

This function continues the I2C interface.

Definition at line 66 of file i2c.c.

8.17.2.2 void Sys_I2C_Read (uint8 address, uint8 * intern_address, uint length, pByteFunction bytehandler)

This function first sends a reading request to the I2C device and, then, handles the incoming bytes with a callback function.

Parameters

in	<i>address</i>	The address of the I2C device that should receive the request
in	<i>intern_address</i>	A pointer to the address which should be read
in	<i>length</i>	the number of bytes of the address
in	<i>bytehandler</i>	a pointer to the handler function that processes the incoming bytes.

Definition at line 350 of file i2c.c.

8.17.2.3 char Sys_I2C_ReadByte () [inline]

This function reads a byte.

Definition at line 302 of file i2c.c.

8.17.2.4 void Sys_I2C_Send_ACK () [inline]

This function sends a ack bits.

Definition at line 275 of file i2c.c.

8.17.2.5 void Sys_I2C_Send_NACK () [inline]

This function sends a NACK bits.

Definition at line 284 of file i2c.c.

8.17.2.6 void Sys_I2C_Send_Restart () [inline]

This function sends a restart bits.

Definition at line 257 of file i2c.c.

8.17.2.7 void Sys_I2C_Send_Start () [inline]

This function sends a start bits via the I2C interface

Definition at line 248 of file i2c.c.

8.17.2.8 void Sys_I2C_Send_Stop () [inline]

This function sends a stop bits.

Definition at line 266 of file i2c.c.

8.17.2.9 void Sys_I2C_SentBytes (uint8 address, uint8 * bytes, uint length)

This function adds bytes into a writing buffer that are written as soon as the I2C is idle.

Note

all bytes are written in sequence

Parameters

in	<i>address</i>	The address of the I2C device that should receive the data
in	<i>bytes</i>	A pointer to the data which should be sent
in	<i>length</i>	the number of bytes to send

Definition at line 325 of file i2c.c.

8.17.2.10 void Sys_I2C_Start_Reading () [inline]

This function sends a reading bits.

Definition at line 293 of file i2c.c.

8.17.2.11 void Sys_I2C_WriteByte (uint8 byte) [inline]

This function writes a byte.

Parameters

in	<i>byte</i>	the byte that has to be written
----	-------------	---------------------------------

Definition at line 312 of file i2c.c.

8.17.2.12 void Sys_Init_I2C (void) [inline]

This function initialises the I2C interface.

Definition at line 39 of file i2c.c.

8.17.2.13 void Sys_Pause_I2C (void) [inline]

This function pauses the I2C interface.

Definition at line 57 of file i2c.c.

8.17.2.14 void Sys_Start_I2C (void) [inline]

This function starts the I2C interface.

Definition at line 48 of file i2c.c.

8.17.2.15 void Sys_Stop_I2C (void) [inline]

This function stops the I2C interface.

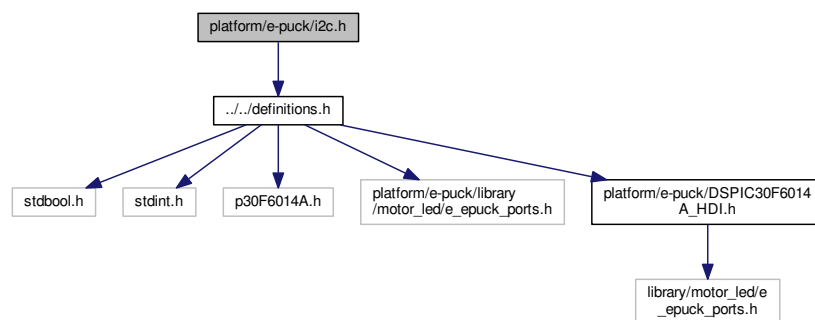
Definition at line 75 of file i2c.c.

8.18 platform/e-puck/i2c.h File Reference

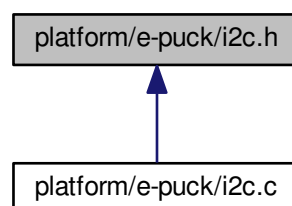
It declares functions to read and write on the I2C interface.

```
#include "../..definitions.h"
```

Include dependency graph for i2c.h:



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



Functions

- void [Sys_Init_I2C](#) (void)
- void [Sys_Start_I2C](#) (void)
- void [Sys_Pause_I2C](#) (void)
- void [Sys_Continue_I2C](#) (void)

- void `Sys_Stop_I2C` (void)
- void `Sys_I2C_SentBytes` (uint8 address, uint8 *bytes, uint length)
- void `Sys_I2C_Read` (uint8 address, uint8 *intern_address, uint length, pByteFunction bytehandler)

8.18.1 Detailed Description

It declares functions to read and write on the I2C interface.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

10 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.18.2 Function Documentation

8.18.2.1 void Sys_Continue_I2C (void) [inline]

This function continues the I2C interface.

Definition at line 66 of file i2c.c.

8.18.2.2 void Sys_I2C_Read (uint8 address, uint8 * intern_address, uint length, pByteFunction bytehandler)

This function first sends a reading request to the I2C device and, then, handles the incoming bytes with a callback function.

Parameters

in	<i>address</i>	The address of the I2C device that should receive the request
in	<i>intern_address</i>	A pointer to the address which should be read
in	<i>length</i>	the number of bytes of the address
in	<i>bytehandler</i>	a pointer to the handler function that processes the incoming bytes.

Definition at line 350 of file i2c.c.

8.18.2.3 void Sys_I2C_SentBytes (uint8 address, uint8 * bytes, uint length)

This function adds bytes into a writing buffer that are written as soon as the I2C is idle.

Note

all bytes are written in sequence

Parameters

in	<i>address</i>	The address of the I2C device that should receive the data
in	<i>bytes</i>	A pointer to the data which should be sent
in	<i>length</i>	the number of bytes to send

Definition at line 325 of file i2c.c.

8.18.2.4 void Sys_Init_I2C (void) [inline]

This function initialises the I2C interface.

Definition at line 39 of file i2c.c.

8.18.2.5 void Sys_Pause_I2C (void) [inline]

This function pauses the I2C interface.

Definition at line 57 of file i2c.c.

8.18.2.6 void Sys_Start_I2C (void) [inline]

This function starts the I2C interface.

Definition at line 48 of file i2c.c.

8.18.2.7 void Sys_Stop_I2C (void) [inline]

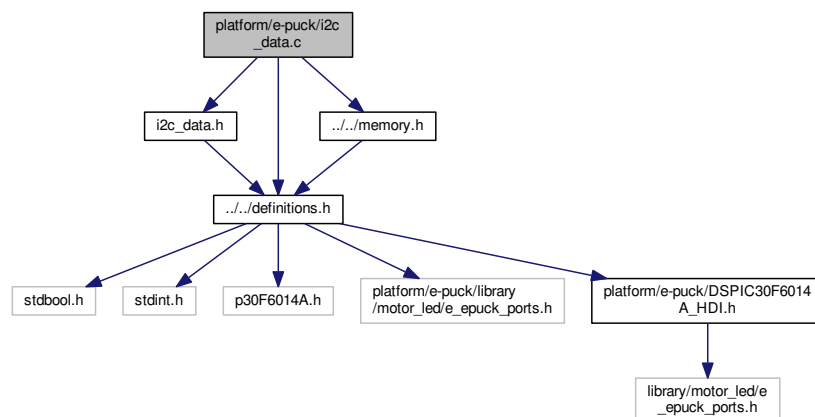
This function stops the I2C interface.

Definition at line 75 of file i2c.c.

8.19 platform/e-puck/i2c_data.c File Reference

It defines functions to manage the I2C queue.

```
#include "i2c_data.h"
#include "../../definitions.h"
#include "../../memory.h"
Include dependency graph for i2c_data.c:
```



Functions

- void [Sys_I2C_RemoveOldestMessage](#) (sys_i2c_messages **list)
- void [Sys_I2C_FreeMessages](#) (sys_i2c_messages *list)
- void [Sys_I2C_AppendMessages](#) (sys_i2c_msg *item)

Variables

- sys_i2c_messages * [sys_i2c_msgs](#) = 0

8.19.1 Detailed Description

It defines functions to manage the I2C queue.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

10 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.19.2 Function Documentation

8.19.2.1 void Sys_I2C_AppendMessages (sys_i2c_msg * item)

This function appends on the bottom of the linked list.

Parameters

<i>in, out</i>	<i>item</i>	pointer to a element that should be added
----------------	-------------	---

Definition at line 64 of file i2c_data.c.

8.19.2.2 void Sys_I2C_FreeMessages (sys_i2c_messages * list)

This function frees all messages of the linked list.

Parameters

<i>in</i>	<i>list</i>	pointer to a list of elements that should be removed
-----------	-------------	--

Definition at line 43 of file i2c_data.c.

8.19.2.3 void Sys_I2C_RemoveOldestMessage (sys_i2c_messages ** list)

This function removes the oldest message (first element) of the linked list

Parameters

<code>in, out</code>	<code>list</code>	pointer to the linked list
----------------------	-------------------	----------------------------

Definition at line 27 of file i2c_data.c.

8.19.3 Variable Documentation

8.19.3.1 `sys_i2c_messages* sys_i2c_msgs = 0`

Pointer to the linked list of messages

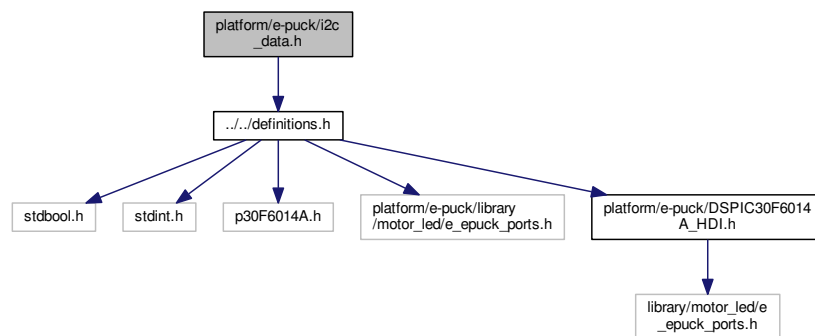
Definition at line 19 of file i2c_data.c.

8.20 platform/e-puck/i2c_data.h File Reference

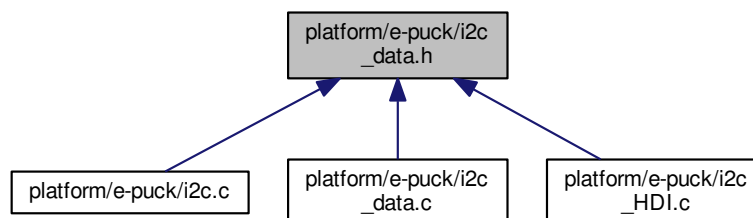
It declares functions to manage the I2C queue.

```
#include "../definitions.h"
```

Include dependency graph for i2c_data.h:



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



Data Structures

- struct [sys_i2c_msg](#)

It is a single linked list element containing messages that need to be sent via I2C. This list acts as a message buffer.

Enumerations

- enum `sys_i2C_state` {
`I2C_IDLE = 0, I2C_IS_STARTING, I2C_STARTED, I2C_IS_READING,`
`I2C_IS_SENDING, I2C_SENT, I2C_ACKNOWLEDGED, I2C_IS_STOPPING,`
`I2C_ERROR` }
- enum `sys_i2C_mode` {
`I2C_IDLE_MODE = 0, I2C_WRITING_ADDRESS_MODE, I2C_READING_BYTES_MODE, I2C_WRITING_BYTES_MODE,`
`I2C_ERROR_MODE` }

Functions

- void `Sys_I2C_AppendMessages` (`sys_i2c_msg *item`)
- void `Sys_I2C_RemoveOldestMessage` (`sys_i2c_messages **list`)
- void `Sys_I2C_FreeMessages` (`sys_i2c_messages *list`)

Variables

- `sys_i2c_messages` * `sys_i2c_msgs`

8.20.1 Detailed Description

It declares functions to manage the I2C queue.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

10 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.20.2 Enumeration Type Documentation

8.20.2.1 enum `sys_i2C_mode`

Enumerator

`I2C_IDLE_MODE`
`I2C_WRITING_ADDRESS_MODE`
`I2C_READING_BYTES_MODE`
`I2C_WRITING_BYTES_MODE`
`I2C_ERROR_MODE`

Definition at line 23 of file `i2c_data.h`.

8.20.2.2 enum sys_i2C_state

Enumerator

I2C_IDLE
I2C_IS_STARTING
I2C_STARTED
I2C_IS_READING
I2C_IS_SENDING
I2C_SENT
I2C_ACKNOWLEDGED
I2C_IS_STOPPING
I2C_ERROR

Definition at line 22 of file i2c_data.h.

8.20.3 Function Documentation

8.20.3.1 void Sys_I2C_AppendMessages (sys_i2c_msg * *item*)

This function appends on the bottom of the linked list.

Parameters

<i>in, out</i>	<i>item</i>	pointer to a element that should be added
----------------	-------------	---

Definition at line 64 of file i2c_data.c.

8.20.3.2 void Sys_I2C_FreeMessages (sys_i2c_messages * *list*)

This function frees all messages of the linked list.

Parameters

<i>in</i>	<i>list</i>	pointer to a list of elements that should be removed
-----------	-------------	--

Definition at line 43 of file i2c_data.c.

8.20.3.3 void Sys_I2C_RemoveOldestMessage (sys_i2c_messages ** *list*)

This function removes the oldest message (first element) of the linked list

Parameters

<i>in, out</i>	<i>list</i>	pointer to the linked list
----------------	-------------	----------------------------

Definition at line 27 of file i2c_data.c.

8.20.4 Variable Documentation

8.20.4.1 sys_i2c_messages* sys_i2c_msgs

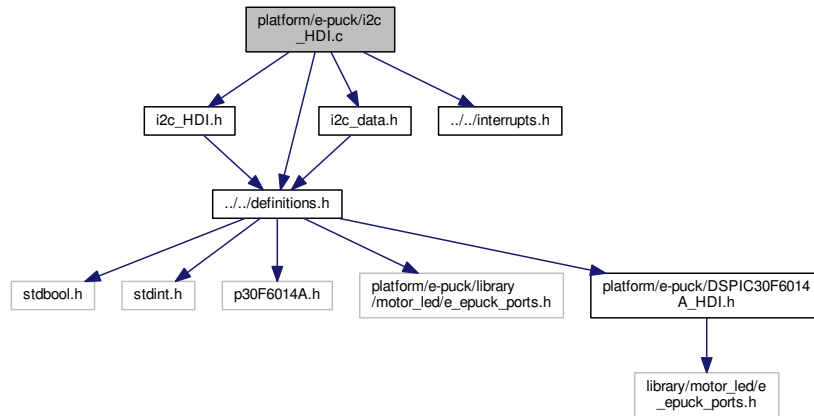
Pointer to the linked list of messages

Definition at line 19 of file i2c_data.c.

8.21 platform/e-puck/i2c_HDI.c File Reference

Hardware dependent implementations to read and write on the I2C interface.

```
#include "i2c_HDI.h"
#include "i2c_data.h"
#include "../definitions.h"
#include "../interrupts.h"
Include dependency graph for i2c_HDI.c:
```



Functions

- void [Sys_Init_I2C_HDI](#) ()
- void [Sys_Start_I2C_HDI](#) (void)
- void [Sys_Pause_I2C_HDI](#) (void)
- void [Sys_Contine_I2C_HDI](#) (void)
- void [Sys_Stop_I2C_HDI](#) (void)
- void [Sys_I2C_Send_Start_HDI](#) ()
- void [Sys_I2C_Send_Restart_HDI](#) (void)
- void [Sys_I2C_Send_Stop_HDI](#) (void)
- void [Sys_I2C_Send_ACK_HDI](#) (void)
- void [Sys_I2C_Send_NACK_HDI](#) (void)
- void [Sys_I2C_Start_Reading_HDI](#) ()
- char [Sys_I2C_ReadByte_HDI](#) ()
- void [Sys_I2C_WriteByte_HDI](#) (uint8 byte)

8.21.1 Detailed Description

Hardware dependent implementations to read and write on the I2C interface.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

10 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.21.2 Function Documentation

8.21.2.1 void Sys_Contine_I2C_HDI (void) [inline]

This function continues the I2C interface.

Definition at line 69 of file i2c_HDI.c.

8.21.2.2 char Sys_I2C_ReadByte_HDI (void) [inline]

This function reads a byte.

Definition at line 165 of file i2c_HDI.c.

8.21.2.3 void Sys_I2C_Send_ACK_HDI (void) [inline]

This function sends a ack bits.

Definition at line 120 of file i2c_HDI.c.

8.21.2.4 void Sys_I2C_Send_NACK_HDI (void) [inline]

This function sends a nack bits.

Definition at line 135 of file i2c_HDI.c.

8.21.2.5 void Sys_I2C_Send_Restart_HDI (void) [inline]

This function sends a restart bits.

Definition at line 100 of file i2c_HDI.c.

8.21.2.6 void Sys_I2C_Send_Start_HDI () [inline]

This function sends a start bits.

Definition at line 89 of file i2c_HDI.c.

8.21.2.7 void Sys_I2C_Send_Stop_HDI (void) [inline]

This function sends a stop bits.

Definition at line 111 of file i2c_HDI.c.

8.21.2.8 void Sys_I2C_Start_Reading_HDI (void) [inline]

This function sends a reading bits.

Definition at line 150 of file i2c_HDI.c.

8.21.2.9 void Sys_I2C_WriteByte_HDI (uint8 byte) [inline]

This function writes a byte.

Parameters

in	byte	the byte that has to be written
----	------	---------------------------------

Definition at line 175 of file i2c_HDI.c.

8.21.2.10 void Sys_Init_I2C_HDI (void) [inline]

This function initialises the I2C interface. < interrupt priority for the I2C interrupt

Definition at line 25 of file i2c_HDI.c.

8.21.2.11 void Sys_Pause_I2C_HDI (void) [inline]

This function pauses the I2C interface.

Definition at line 60 of file i2c_HDI.c.

8.21.2.12 void Sys_Start_I2C_HDI (void) [inline]

This function starts the I2C interface.

Definition at line 49 of file i2c_HDI.c.

8.21.2.13 void Sys_Stop_I2C_HDI (void) [inline]

This function stops the I2C interface.

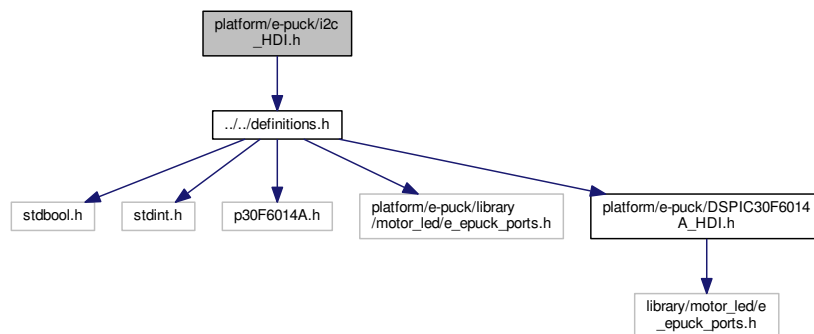
Definition at line 78 of file i2c_HDI.c.

8.22 platform/e-puck/i2c_HDI.h File Reference

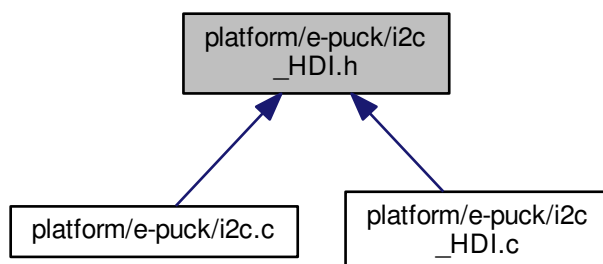
Hardware dependent implementations to read and write on the I2C interface.

```
#include "../definitions.h"
```

Include dependency graph for i2c_HDI.h:



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



Functions

- void [Sys_I2C_Send_Start_HDI](#) ()
- void [Sys_I2C_Send_Restart_HDI](#) (void)
- void [Sys_I2C_Send_Stop_HDI](#) (void)
- void [Sys_I2C_Send_ACK_HDI](#) (void)
- void [Sys_I2C_Send_NACK_HDI](#) (void)
- void [Sys_I2C_Start_Reading_HDI](#) (void)
- char [Sys_I2C_ReadByte_HDI](#) (void)
- void [Sys_I2C_WriteByte_HDI](#) (uint8 byte)
- void [Sys_Init_I2C_HDI](#) (void)
- void [Sys_Start_I2C_HDI](#) (void)
- void [Sys_Pause_I2C_HDI](#) (void)
- void [Sys_Continue_I2C_HDI](#) (void)
- void [Sys_Stop_I2C_HDI](#) (void)

8.22.1 Detailed Description

Hardware dependent implementations to read and write on the I2C interface.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

10 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.22.2 Function Documentation

8.22.2.1 void Sys_Contine_I2C_HDI (void) [inline]

This function continues the I2C interface.

Definition at line 69 of file i2c_HDI.c.

8.22.2.2 char Sys_I2C_ReadByte_HDI (void) [inline]

This function reads a byte.

Definition at line 165 of file i2c_HDI.c.

8.22.2.3 void Sys_I2C_Send_ACK_HDI (void) [inline]

This function sends a ack bits.

Definition at line 120 of file i2c_HDI.c.

8.22.2.4 void Sys_I2C_Send_NACK_HDI (void) [inline]

This function sends a nack bits.

Definition at line 135 of file i2c_HDI.c.

8.22.2.5 void Sys_I2C_Send_Restart_HDI (void) [inline]

This function sends a restart bits.

Definition at line 100 of file i2c_HDI.c.

8.22.2.6 void Sys_I2C_Send_Start_HDI () [inline]

This function sends a start bits.

Definition at line 89 of file i2c_HDI.c.

8.22.2.7 void Sys_I2C_Send_Stop_HDI (void) [inline]

This function sends a stop bits.

Definition at line 111 of file i2c_HDI.c.

8.22.2.8 void Sys_I2C_Start_Reading_HDI (void) [inline]

This function sends a reading bits.

Definition at line 150 of file i2c_HDI.c.

8.22.2.9 void Sys_I2C_WriteByte_HDI (uint8 byte) [inline]

This function writes a byte.

Parameters

<i>in</i>	<i>byte</i>	the byte that has to be written
-----------	-------------	---------------------------------

Definition at line 175 of file i2c_HDI.c.

8.22.2.10 void Sys_Init_I2C_HDI (void) [inline]

This function initialises the I2C interface. < interrupt priority for the I2C interrupt

Definition at line 25 of file i2c_HDI.c.

8.22.2.11 void Sys_Pause_I2C_HDI(void) [inline]

This function pauses the I2C interface.

Definition at line 60 of file i2c_HDI.c.

8.22.2.12 void Sys_Start_I2C_HDI(void) [inline]

This function starts the I2C interface.

Definition at line 49 of file i2c_HDI.c.

8.22.2.13 void Sys_Stop_I2C_HDI(void) [inline]

This function stops the I2C interface.

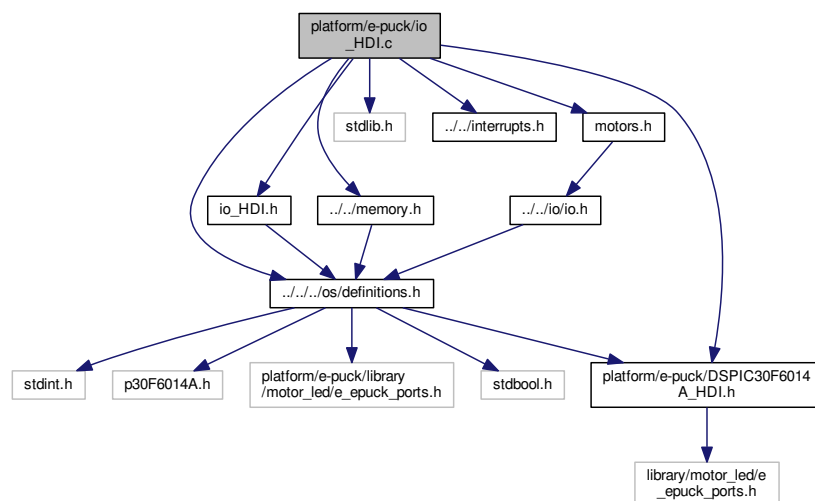
Definition at line 78 of file i2c_HDI.c.

8.23 platform/e-puck/io_HDI.c File Reference

Hardware dependent implementations to control the IO timer and to (un)register IO Handler.

```
#include "io_HDI.h"
#include <stdlib.h>
#include "DSPIC30F6014A_HDI.h"
#include "../definitions.h"
#include "../interrupts.h"
#include "../memory.h"
#include "motors.h"
```

Include dependency graph for io_HDI.c:

**Functions**

- void [Sys_Init_IOTimer_HDI](#) ()
- void [Sys_Start_IOTimer_HDI](#) ()
- void [Sys_Stop_IOTimer_HDI](#) ()
- void [Sys_Continue_IOTimer_HDI](#) ()
- void [Sys_Reset_IOTimer_HDI](#) ()

- void `__attribute__ ((interrupt, no_auto_psv))`
- void `Sys_Disable_IOTimerInterrupt_HDI ()`
- void `Sys_Enable_IOTimerInterrupt_HDI ()`
- void `Sys_Force_IOTimerInterrupt_HDI ()`
- void `Sys_IOTimer_code_HDI ()`

Variables

- `sys_periodical_IOHandler` * `sys_iohandlers`

8.23.1 Detailed Description

Hardware dependent implementations to control the IO timer and to (un)register IO Handler.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

10 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.23.2 Function Documentation

8.23.2.1 void `__attribute__ ((interrupt, no_auto_psv))`

This Function starts the task-scheduling algorithm

Interrupt Service Routine for the Timer1 (alternate) starts the task-scheduling algorithm

Address error trap.

This function is called when an address error occurs. That means that a call address of a function or in the stack addresses an area outside the memory. Similarly, if a pointer points to memory outside the range, this trap happens.

Stack error trap.

This function is called when an stack error occurs. That means that the stack pointer, stack pointer limit, or frame pointer are pointing outside their range.

Math error trap.

This function is called when an math error occurs. That means an illegal math operation was performed (such as division by 0 or NaN).

Alternative Oscillator fail trap.

This function is called when an oscillator fail occurs. This should never happen.

Alternative address error trap.

This function is called when an address error occurs. That means that a call address of a function or in the stack addresses an area outside the memory. Similarly, if a pointer points to memory outside the range, this trap happens.

Alternative stack error trap.

This function is called when an stack error occurs. That means that the stack pointer, stack pointer limit, or frame pointer are pointing outside their range.

Alternative math error trap.

This function is called when an math error occurs. That means an illegal math operation was performed (such as division by 0 or NaN).

Default interrupt service routine.

This function is called when no other interrupt routine is specified.

Definition at line 103 of file io_HDI.c.

8.23.2.2 void Sys_Continue_IOTimer_HDI(void) [inline]

This function continues the I/O Timer.

Definition at line 81 of file io_HDI.c.

8.23.2.3 void Sys_Disable_IOTimerInterrupt_HDI(void) [inline]

Disables the Timer1 interrupt and sets the interrupt flag to 0

Definition at line 122 of file io_HDI.c.

8.23.2.4 void Sys_Enable_IOTimerInterrupt_HDI(void) [inline]

Enables the Timer1 interrupt and leaves the interrupt flag to its value. If the flag was set, the Timer1 interrupt will be emitted after executing this function.

Definition at line 132 of file io_HDI.c.

8.23.2.5 void Sys_Force_IOTimerInterrupt_HDI(void) [inline]

forces the Timer1 interrupt to occur.

Definition at line 140 of file io_HDI.c.

8.23.2.6 void Sys_Init_IOTimer_HDI() [inline]

This function initialises the I/O Timer.

Definition at line 33 of file io_HDI.c.

8.23.2.7 void Sys_IOTimer_code_HDI() [inline]

This function is executed every time the I/O timer is active and executes all I/O handlers

Definition at line 149 of file io_HDI.c.

8.23.2.8 void Sys_Reset_IOTimer_HDI(void) [inline]

This function resets the I/O Timer.

Definition at line 93 of file io_HDI.c.

8.23.2.9 void Sys_Start_IOTimer_HDI() [inline]

This function starts the I/O Timer. < interrupt priority for the I/O timer interrupt

Definition at line 57 of file io_HDI.c.

8.23.2.10 void Sys_Stop_IOTimer_HDI(void) [inline]

This function stops the I/O Timer.

Definition at line 69 of file io_HDI.c.

8.23.3 Variable Documentation

8.23.3.1 `sys_periodical_IOHandler* sys_iohandlers`

List of I/O handlers

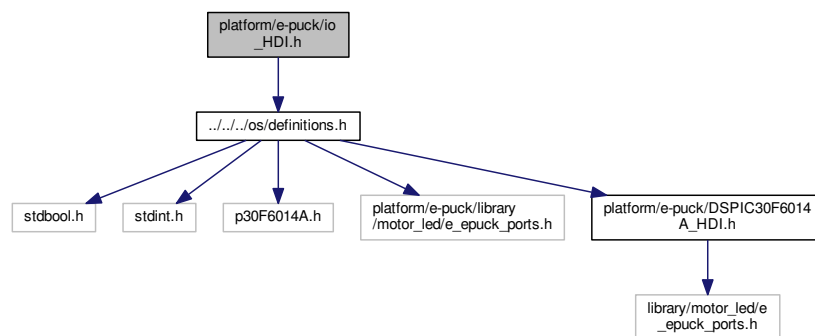
Definition at line 25 of file io_HDI.c.

8.24 platform/e-puck/io_HDI.h File Reference

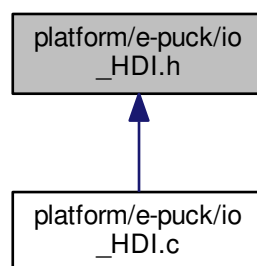
Hardware dependent implementations to control the IO timer and to (un)register IO Handler.

```
#include "../../os/definitions.h"
```

Include dependency graph for io_HDI.h:



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



Data Structures

- struct [sys_pIOHandler](#)

Linked list element containing I/O Handler function pointers.

Macros

- `#define STEPS_PER_SECOND 10000`
- `#define STEPS_PER_MILLISECOND 10`

Functions

- void `Sys_Init_IOTimer_HDI` ()
- void `Sys_Start_IOTimer_HDI` ()
- void `Sys_IOTimer_code_HDI` ()
- void `Sys_Stop_IOTimer_HDI` (void)
- void `Sys_Continue_IOTimer_HDI` (void)
- void `Sys_Reset_IOTimer_HDI` (void)
- void `Sys_Disable_IOTimerInterrupt_HDI` (void)
- void `Sys_Enable_IOTimerInterrupt_HDI` (void)
- void `Sys_Force_IOTimerInterrupt_HDI` (void)

Variables

- `sys_periodical_IOHandler` * `sys_iohandlers`

8.24.1 Detailed Description

Hardware dependent implementations to control the IO timer and to (un)register IO Handler.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

10 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.24.2 Macro Definition Documentation

8.24.2.1 `#define STEPS_PER_MILLISECOND 10`

Definition at line 25 of file `io_HDI.h`.

8.24.2.2 `#define STEPS_PER_SECOND 10000`

Definition at line 24 of file `io_HDI.h`.

8.24.3 Function Documentation

8.24.3.1 `void Sys_Continue_IOTimer_HDI(void) [inline]`

This function continues the I/O Timer.

Definition at line 81 of file `io_HDI.c`.

8.24.3.2 `void Sys_Disable_IOTimerInterrupt_HDI(void) [inline]`

Disables the Timer1 interrupt and sets the interrupt flag to 0

Definition at line 122 of file `io_HDI.c`.

8.24.3.3 `void Sys_Enable_IOTimerInterrupt_HDI(void) [inline]`

Enables the Timer1 interrupt and leaves the interrupt flag to its value. If the flag was set, the Timer1 interrupt will be emitted after executing this function.

Definition at line 132 of file `io_HDI.c`.

8.24.3.4 `void Sys_Force_IOTimerInterrupt_HDI(void) [inline]`

forces the Timer1 interrupt to occur.

Definition at line 140 of file `io_HDI.c`.

8.24.3.5 `void Sys_Init_IOTimer_HDI() [inline]`

This function initialises the I/O Timer.

Definition at line 33 of file `io_HDI.c`.

8.24.3.6 `void Sys_IOTimer_code_HDI() [inline]`

This function is executed every time the I/O timer is active and executes all I/O handlers

Definition at line 149 of file `io_HDI.c`.

8.24.3.7 `void Sys_Reset_IOTimer_HDI(void) [inline]`

This function resets the I/O Timer.

Definition at line 93 of file `io_HDI.c`.

8.24.3.8 `void Sys_Start_IOTimer_HDI() [inline]`

This function starts the I/O Timer. < interrupt priority for the I/O timer interrupt

Definition at line 57 of file `io_HDI.c`.

8.24.3.9 `void Sys_Stop_IOTimer_HDI(void) [inline]`

This function stops the I/O Timer.

Definition at line 69 of file `io_HDI.c`.

8.24.4 Variable Documentation

8.24.4.1 `sys_periodical_IOHandler* sys_iohandlers`

List of I/O handlers

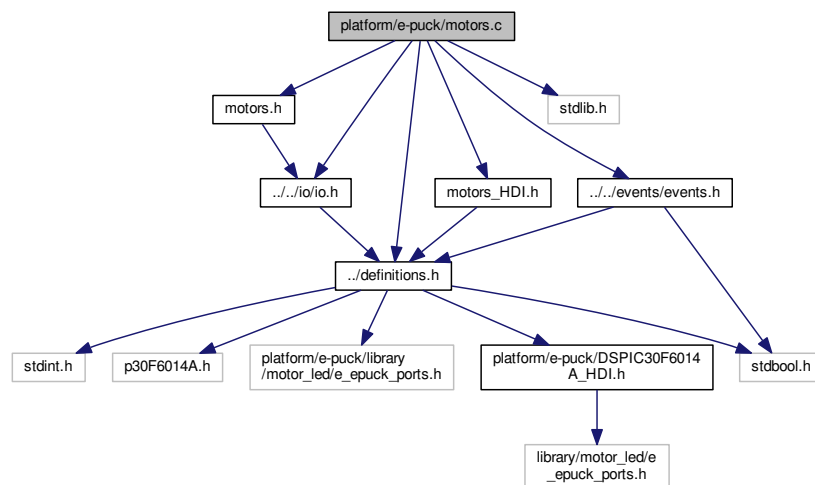
Definition at line 25 of file `io_HDI.c`.

8.25 platform/e-puck/motors.c File Reference

It defines function to drive motors.

```
#include "motors.h"
#include "motors_HDI.h"
#include "../../io/io.h"
#include "../../events/events.h"
#include "../../definitions.h"
#include <stdlib.h>
```

Include dependency graph for motors.c:



Data Structures

- struct [sys_motors](#)

This struct contains the speed for a motor.

Macros

- #define [MAX_WHEEL_SPEED](#) 128
- #define [POWER_SAVE_WAIT](#) 15

Functions

- void [Sys_LeftMotor_Controller](#) (void)
- void [Sys_RightMotor_Controller](#) (void)
- bool [Sys_LeftMotor_EventHandler](#) (uint, uint, sys_event_data *)
- bool [Sys_RightMotor_EventHandler](#) (uint, uint, sys_event_data *)
- void [Sys_Init_Motors](#) ()
- void [Sys_LeftMotor_Reset](#) ()
- void [Sys_RightMotor_Reset](#) ()
- void [Sys_Set_LeftWheelSpeed](#) (sint16 speed)
- void [Sys_Set_RightWheelSpeed](#) (sint16 speed)
- sint16 [Sys_Get_LeftWheelSpeed](#) (void)
- sint16 [Sys_Get_RightWheelSpeed](#) (void)

8.25.1 Detailed Description

It defines function to drive motors.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org
 Gabriel Kapellmann Zafra <gkapellmann@gmail.com>

Version

1.0

Date

30 July 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.25.2 Macro Definition Documentation

8.25.2.1 #define MAX_WHEEL_SPEED 128

Maximum wheel speed in steps

Definition at line 25 of file motors.c.

8.25.2.2 #define POWER_SAVE_WAIT 15

amount of steps needed to move the motor one step further

Definition at line 26 of file motors.c.

8.25.3 Function Documentation

8.25.3.1 sint16 Sys_Get_LeftWheelSpeed (void)

This function returns the speed of the left motor.

Definition at line 269 of file motors.c.

8.25.3.2 sint16 Sys_Get_RightWheelSpeed (void)

This function returns the speed of the right motor.

Definition at line 278 of file motors.c.

8.25.3.3 void Sys_Init_Motors (void)

This function initialises the motor module including both left and right motor. < ID of the event that controls the left motor speed/direction

< ID of the event that controls the right motor speed/direction

< ID of the event that controls the left motor speed/direction

< ID of the event that controls the right motor speed/direction

< ID of the event that controls the left motor speed/direction

< ID of the event that controls the right motor speed/direction
 < ID of the event that controls the left motor speed/direction
 < ID of the event that controls the right motor speed/direction

Definition at line 49 of file motors.c.

8.25.3.4 void Sys_LeftMotor_Controller ()

This function controls the speed of the left motor. The speed is set by moving the to the next step within the appropriate time step. < Maximum wheel speed in steps

< amount of steps needed to move the motor one step further
 < Maximum wheel speed in steps
 < amount of steps needed to move the motor one step further

Definition at line 110 of file motors.c.

8.25.3.5 bool Sys_LeftMotor_EventHandler (uint *pid*, uint *eventID*, sys_event_data * *data*)

This function sets the left motor speed that is received by the event SYS_EVENT_IO_MOTOR_LEFT.

Parameters

in	<i>pid</i>	the process id to which the event handler is registered
in	<i>eventID</i>	the event id which identifies the event that is handled
in	<i>data</i>	the event data that contain the motor speed.

< Maximum wheel speed in steps
 < Maximum wheel speed in mm/s

Definition at line 207 of file motors.c.

8.25.3.6 void Sys_LeftMotor_Reset () [inline]

This function resets the left motor to a reset state. < the reset value for the motor phase

Definition at line 92 of file motors.c.

8.25.3.7 void Sys_RightMotor_Controller ()

This function controls the speed of the right motor. The speed is set by moving the to the next step within the appropriate time step. < Maximum wheel speed in steps

< amount of steps needed to move the motor one step further
 < Maximum wheel speed in steps
 < amount of steps needed to move the motor one step further

Definition at line 156 of file motors.c.

8.25.3.8 bool Sys_RightMotor_EventHandler (uint *pid*, uint *eventID*, sys_event_data * *data*)

This function sets the right motor speed that is received by the event SYS_EVENT_IO_MOTOR_RIGHT.

Parameters

in	<i>pid</i>	the process id to which the event handler is registered
in	<i>eventID</i>	the event id which identifies the event that is handled
in	<i>data</i>	the event data that contain the motor speed.

< Maximum wheel speed in steps
 < Maximum wheel speed in mm/s

Definition at line 221 of file motors.c.

8.25.3.9 void Sys_RightMotor_Reset () [inline]

This function resets the right motor to a reset state. < the reset value for the motor phase

Definition at line 101 of file motors.c.

8.25.3.10 void Sys_Set_LeftWheelSpeed (sint16 speed)

This function sets the value for the speed of the left motor.

Parameters

<i>speed</i>	of the left wheel
--------------	-------------------

< Maximum wheel speed in steps

< Maximum wheel speed in steps

< Maximum wheel speed in steps

< Maximum wheel speed in steps

Definition at line 236 of file motors.c.

8.25.3.11 void Sys_Set_RightWheelSpeed (sint16 speed)

This function sets the value for the speed of the right motor.

Parameters

<i>speed</i>	of the right wheel
--------------	--------------------

< Maximum wheel speed in steps

< Maximum wheel speed in steps

< Maximum wheel speed in steps

< Maximum wheel speed in steps

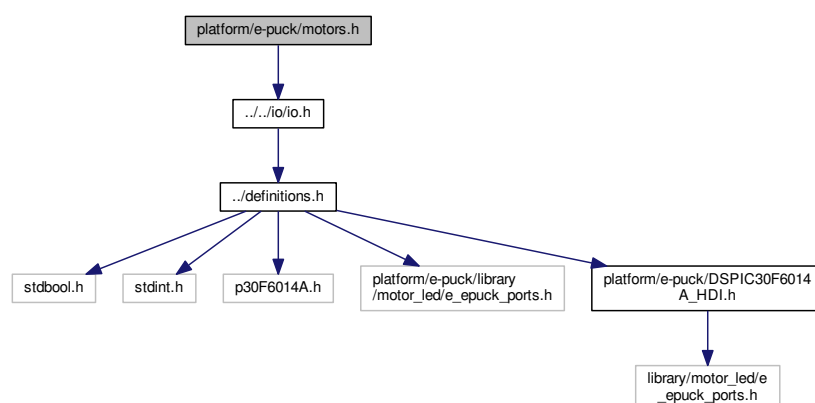
Definition at line 253 of file motors.c.

8.26 platform/e-puck/motors.h File Reference

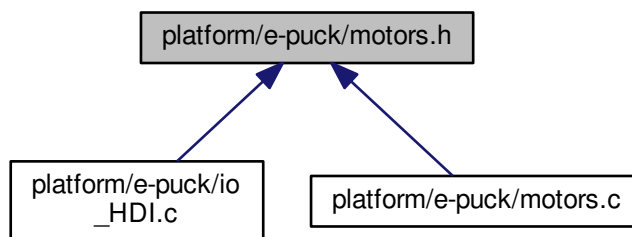
It declares functions to drive motors.

```
#include "../..io/io.h"
```

Include dependency graph for motors.h:



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



Macros

- `#define MAX_WHEEL_SPEED_MM_S 129 /*mm/s*/`

Functions

- void `Sys_Init_Motors` (void)
- void `Sys_Set_LeftWheelSpeed` (sint16 speed)
- void `Sys_Set_RightWheelSpeed` (sint16 speed)
- sint16 `Sys_Get_LeftWheelSpeed` (void)
- sint16 `Sys_Get_RightWheelSpeed` (void)

8.26.1 Detailed Description

It declares functions to drive motors.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org
Gabriel Kapellmann Zafrá <gkapellmann@gmail.com>

Version

1.0

Date

30 July 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.26.2 Macro Definition Documentation

8.26.2.1 `#define MAX_WHEEL_SPEED_MM_S 129 /*mm/s*/`

Maximum wheel speed in mm/s

Definition at line 45 of file motors.h.

8.26.3 Function Documentation

8.26.3.1 sint16 Sys_Get_LeftWheelSpeed (void)

This function returns the speed of the left motor.

Definition at line 269 of file motors.c.

8.26.3.2 sint16 Sys_Get_RightWheelSpeed (void)

This function returns the speed of the right motor.

Definition at line 278 of file motors.c.

8.26.3.3 void Sys_Init_Motors (void)

This function initialises the motor module including both left and right motor. < ID of the event that controls the left motor speed/direction

< ID of the event that controls the right motor speed/direction

< ID of the event that controls the left motor speed/direction

< ID of the event that controls the right motor speed/direction

< ID of the event that controls the left motor speed/direction

< ID of the event that controls the right motor speed/direction

< ID of the event that controls the left motor speed/direction

< ID of the event that controls the right motor speed/direction

Definition at line 49 of file motors.c.

8.26.3.4 void Sys_Set_LeftWheelSpeed (sint16 speed)

This function sets the value for the speed of the left motor.

Parameters

<i>speed</i>	of the left wheel
--------------	-------------------

< Maximum wheel speed in steps

< Maximum wheel speed in steps

< Maximum wheel speed in steps

< Maximum wheel speed in steps

Definition at line 236 of file motors.c.

8.26.3.5 void Sys_Set_RightWheelSpeed (sint16 speed)

This function sets the value for the speed of the right motor.

Parameters

<i>speed</i>	of the right wheel
--------------	--------------------

< Maximum wheel speed in steps

< Maximum wheel speed in steps

< Maximum wheel speed in steps

< Maximum wheel speed in steps

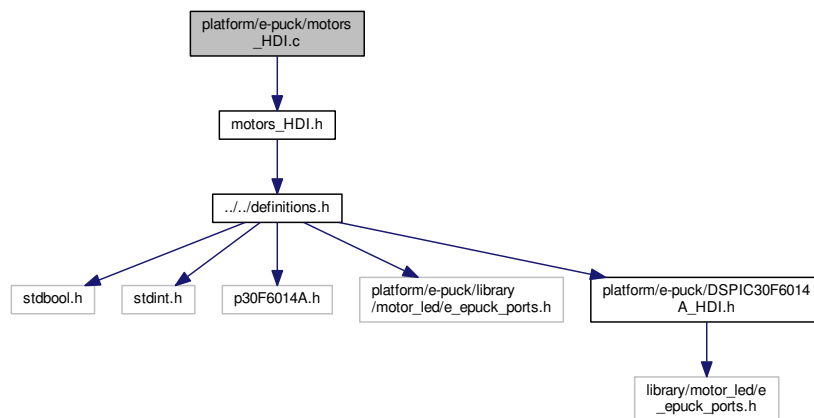
Definition at line 253 of file motors.c.

8.27 platform/e-puck/motors_HDI.c File Reference

Hardware dependent implementations to drive motors.

```
#include "motors_HDI.h"
```

Include dependency graph for motors_HDI.c:



Functions

- void [Sys_LeftMotor_SetPhase_HDI](#) (sint8 phase)
- void [Sys_RightMotor_SetPhase_HDI](#) (sint8 phase)

8.27.1 Detailed Description

Hardware dependent implementations to drive motors.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

27 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.27.2 Function Documentation

8.27.2.1 void `Sys_LeftMotor_SetPhase_HDI` (sint8 phase) [inline]

This function sets the left motor phase

Parameters

<code>in</code>	<code>phase</code>	indicates the phase of the left motor
-----------------	--------------------	---------------------------------------

Definition at line 25 of file `motors_HDI.c`.

8.27.2.2 `void Sys_RightMotor_SetPhase_HDI (sint8 phase) [inline]`

This function sets the right motor phase

Parameters

<code>in</code>	<code>phase</code>	indicates the phase of the right motor
-----------------	--------------------	--

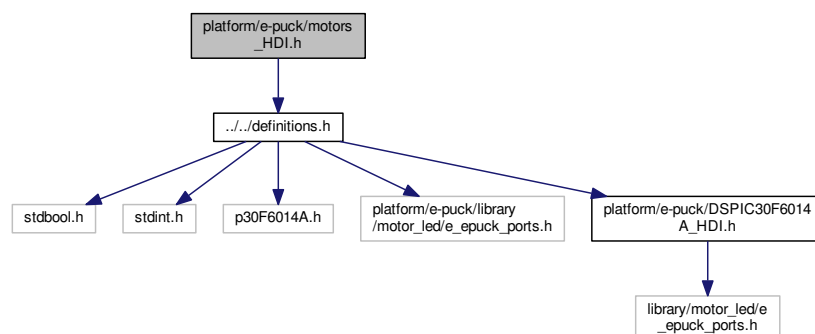
Definition at line 78 of file `motors_HDI.c`.

8.28 platform/e-puck/motors_HDI.h File Reference

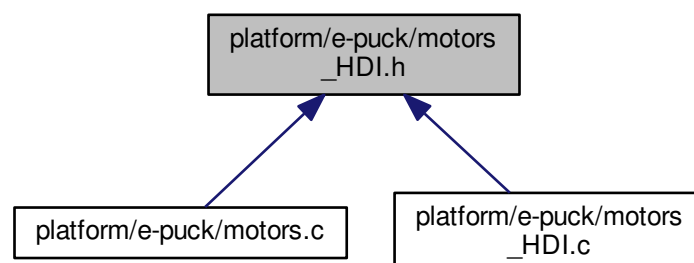
Hardware dependent implementations to drive motors.

```
#include "../definitions.h"
```

Include dependency graph for `motors_HDI.h`:



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



Macros

- #define [MOTORPHASE_RESET](#) -1

Functions

- void [Sys_LeftMotor_SetPhase_HDI](#) (sint8 phase)
- void [Sys_RightMotor_SetPhase_HDI](#) (sint8 phase)

8.28.1 Detailed Description

Hardware dependent implementations to drive motors.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

27 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.28.2 Macro Definition Documentation

8.28.2.1 #define MOTORPHASE_RESET -1

the reset value for the motor phase

Definition at line 24 of file motors_HDI.h.

8.28.3 Function Documentation

8.28.3.1 void Sys_LeftMotor_SetPhase_HDI (sint8 *phase*) [inline]

This function sets the left motor phase

Parameters

<i>in</i>	<i>phase</i>	indicates the phase of the left motor
-----------	--------------	---------------------------------------

Definition at line 25 of file motors_HDI.c.

8.28.3.2 void Sys_RightMotor_SetPhase_HDI (sint8 *phase*) [inline]

This function sets the right motor phase

Parameters

in	<i>phase</i>	indicates the phase of the right motor
----	--------------	--

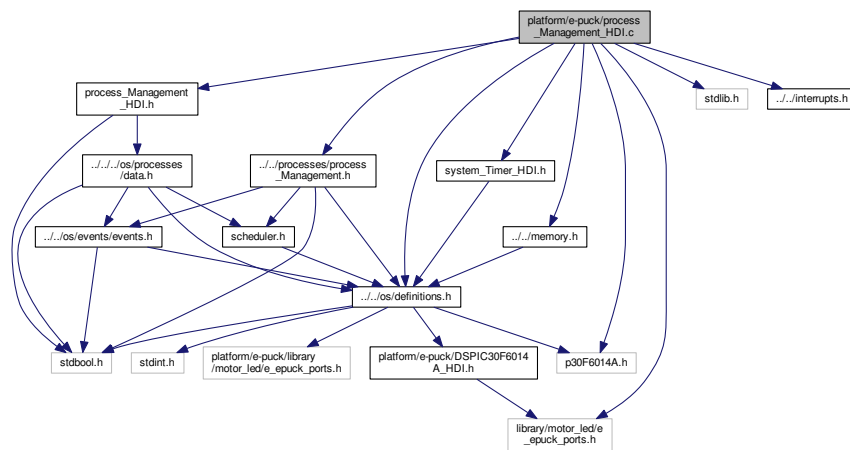
Definition at line 78 of file motors_HDI.c.

8.29 platform/e-puck/process_Management_HDI.c File Reference

Hardware dependent implementations to manage processes (e.g. task swichting)

```
#include "process_Management_HDI.h"
#include "../..//processes/process_Management.h"
#include <stdlib.h>
#include "system_Timer_HDI.h"
#include "../..//interrupts.h"
#include "../..//memory.h"
#include "../..//definitions.h"
#include <p30F6014A.h>
#include "library/motor_led/e_epuck_ports.h"
```

Include dependency graph for process_Management_HDI.c:



Functions

- void [Sys_Init_Process_Management_HDI](#) ()
- bool [Sys_Start_Process_HDI](#) (pFunction function)
- void [Sys_Save_Running_Process_HDI](#) ()
- void [Sys_Change_Stack_HDI](#) (unsigned short fp, unsigned short sp, unsigned short lm)
- void [Sys_Switch_Process_HDI](#) (sys_pcb_list_element *new_process)

8.29.1 Detailed Description

Hardware dependent implementations to manage processes (e.g. task swichting)

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

{08 July 2014}

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.29.2 Function Documentation

8.29.2.1 void Sys_Change_Stack_HDI (unsigned short *fp*, unsigned short *sp*, unsigned short *lm*)

This function changes stackpointers to the new stack

Parameters

in	<i>fp</i>	FramePointer address
in	<i>sp</i>	StackPointer address
in	<i>lm</i>	StackPointer Limit

Definition at line 212 of file process_Management_HDI.c.

8.29.2.2 void Sys_Init_Process_Management_HDI (void)

This function initialises the process management and creates the first elements in the linked list < Process priority: System = highest

Definition at line 44 of file process_Management_HDI.c.

8.29.2.3 void Sys_Save_Running_Process_HDI (void) [inline]

This function stores all registers and information of the running process into the corresponding struct

Definition at line 149 of file process_Management_HDI.c.

8.29.2.4 bool Sys_Start_Process_HDI (pFunction *function*)

This function creates a new sys_process_control_block (in a sys_process_control_block_list_element) which contains all information which is used to execute this process.

Parameters

in	<i>function</i>	This argument points to a function in memory which should be executed as a new task
----	-----------------	---

< State to indicate that a process is created but not yet ready to be executed

Definition at line 94 of file process_Management_HDI.c.

8.29.2.5 void Sys_Switch_Process_HDI (sys_pcb_list_element * *new_process*)

This function switches from sys_running_process to new_process

Parameters

in	<i>new_process</i>	pointer to the process which should be executed
----	--------------------	---

< State to indicate that a process is waiting to be executed

< State to indicate that a process is created but not yet ready to be executed

< State to indicate that a process is executed

< State to indicate that a process is executed

Definition at line 244 of file process_Management_HDI.c.

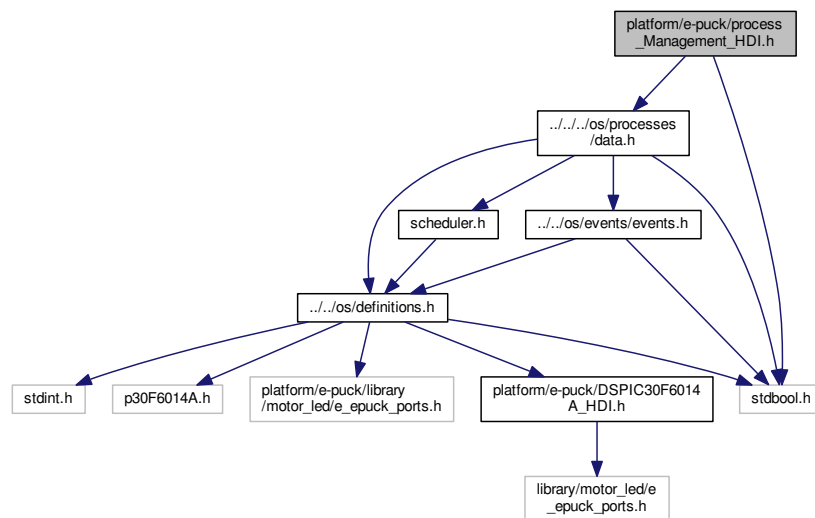
8.30 platform/e-puck/process_Management_HDI.h File Reference

Hardware dependent implementations to manage processes (e.g. task swichting)

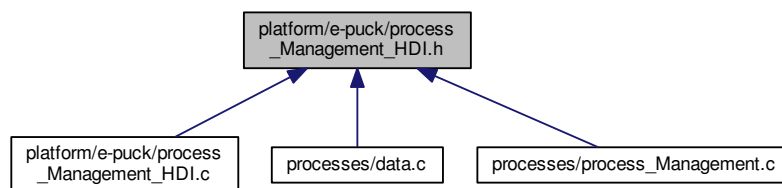
```
#include "../os/processes/data.h"
```

```
#include <stdbool.h>
```

Include dependency graph for process_Management_HDI.h:



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



Functions

- void [Sys_Init_Process_Management_HDI](#) (void)
- bool [Sys_Start_Process_HDI](#) (pFunction function)
- void [Sys_Save_Running_Process_HDI](#) (void)
- void [Sys_Change_Stack_HDI](#) (unsigned short fp, unsigned short sp, unsigned short lm)
- void [Sys_Switch_Process_HDI](#) (sys_pcb_list_element *new_process)

8.30.1 Detailed Description

Hardware dependent implementations to manage processes (e.g. task swichting)

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

{08 July 2014}

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.30.2 Function Documentation

8.30.2.1 void Sys_Change_Stack_HDI (unsigned short *fp*, unsigned short *sp*, unsigned short *lm*)

This function changes stackpointers to the new stack

Parameters

in	<i>fp</i>	FramePointer address
in	<i>sp</i>	StackPointer address
in	<i>lm</i>	StackPointer Limit

Definition at line 212 of file process_Management_HDI.c.

8.30.2.2 void Sys_Init_Process_Management_HDI (void)

This function initialises the process management and creates the first elements in the linked list < Process priority: System = highest

Definition at line 44 of file process_Management_HDI.c.

8.30.2.3 void Sys_Save_Running_Process_HDI (void) [inline]

This function stores all registers and information of the running process into the corresponding struct

Definition at line 149 of file process_Management_HDI.c.

8.30.2.4 bool Sys_Start_Process_HDI (pFunction *function*)

This function creates a new sys_process_control_block (in a sys_process_control_block_list_element) which contains all information wich is used to execute this process.

Parameters

in	<i>function</i>	This argument points to a function in memory which should be executed as an new task
----	-----------------	--

< State to indicate that a process is created but not yet ready to be executed

Definition at line 94 of file process_Management_HDI.c.

8.30.2.5 void Sys_Switch_Process_HDI (sys_pcb_list_element * *new_process*)

This function switches from sys_running_process to new_process

Parameters

<code>in</code>	<code>new_process</code>	pointer to the process which should be executed
-----------------	--------------------------	---

< State to indicate that a process is waiting to be executed

< State to indicate that a process is created but not yet ready to be executed

< State to indicate that a process is executed

< State to indicate that a process is executed

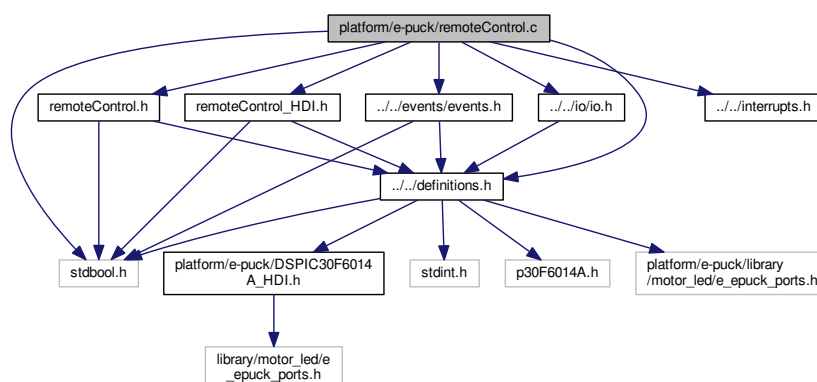
Definition at line 244 of file process_Management_HDI.c.

8.31 platform/e-puck/remoteControl.c File Reference

It defines functions to receive and decode messages from a remote control.

```
#include "remoteControl.h"
#include "remoteControl_HDI.h"
#include <stdbool.h>
#include "../io/io.h"
#include "../events/events.h"
#include "../definitions.h"
#include "../interrupts.h"
```

Include dependency graph for remoteControl.c:

**Functions**

- void [Sys_Init_RemoteControl](#) (void)
- void [Sys_Start_RemoteControl](#) (void)
- void [Sys_Receive_RemoteControl_Msg](#) ()
- bool [Sys_HasRemoteC_Sent_New_Data](#) ()
- uint8 [Sys_RemoteC_Get_CheckBit](#) ()
- uint8 [Sys_RemoteC_Get_Address](#) ()
- uint8 [Sys_RemoteC_Get_Data](#) ()

8.31.1 Detailed Description

It defines functions to receive and decode messages from a remote control.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org
Yuri Kaszubowski Lopes yurikazuba@gmail.com

Version

1.0

Date

27 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.31.2 Function Documentation**8.31.2.1 bool Sys_HasRemoteC_Sent_New_Data ()**

This function returns true if a new remote control command has arrived

Returns

bool true if a new remote control command has arrived

Definition at line 118 of file remoteControl.c.

8.31.2.2 void Sys_Init_RemoteControl (void) [inline]

This function initialises the handler of the remote control to receive signals from the remote control.

Definition at line 34 of file remoteControl.c.

8.31.2.3 void Sys_Receive_RemoteControl_Msg (void)

This function reads a remote control signal and reads it's transmitted value. When a signal arrives, an external interrupt is triggered. The remaining values are obtained by using time not interrupts. < The initial state of the state machine to decode a remote control message

- < Cycles that are needed to wait a single bit duration
- < Cycles that are needed to wait a quarter of a single bit duration
- < Cycles that are needed to wait at the beginning of a message
- < Cycles that are needed to wait a half of a single bit duration
- < Cycles that are needed to wait a single bit duration
- < The initial state of the state machine to decode a remote control message
- < ID of the event that is sent after receiving a remote control signal

Definition at line 52 of file remoteControl.c.

8.31.2.4 uint8 Sys_RemoteC_Get_Address (void)

This function returns the address of an remote control.

Returns

address of the remote control

Definition at line 138 of file remoteControl.c.

8.31.2.5 uint8 Sys_RemoteC_Get_CheckBit (void)

This function returns the check bit value of an remote control to indicate if a button was pressed continuously or sequential.

Returns

bit to indicate the check bit

Definition at line 128 of file remoteControl.c.

8.31.2.6 uint8 Sys_RemoteC_Get_Data (void)

returns the value received by the remote control

Returns

value received by the remote control

Definition at line 148 of file remoteControl.c.

8.31.2.7 void Sys_Start_RemoteControl (void) [inline]

This function start the handler of the remote control to receive signals from the remote control.

Definition at line 43 of file remoteControl.c.

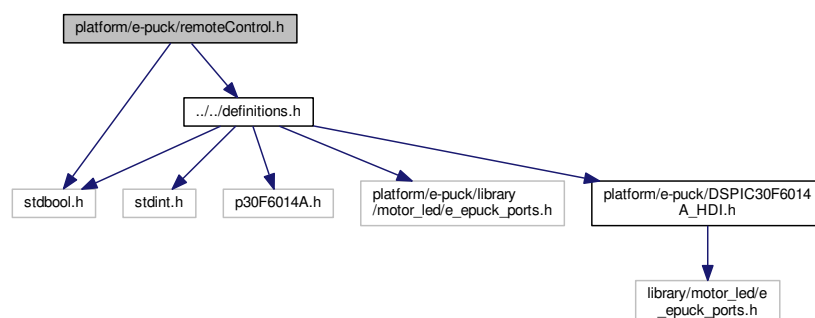
8.32 platform/e-puck/remoteControl.h File Reference

It declares functions to receive and decode messages from a remote control.

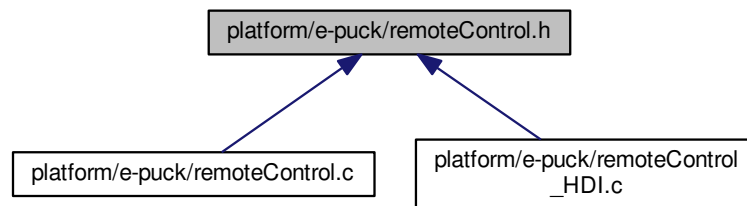
```
#include <stdbool.h>
```

```
#include "../..definitions.h"
```

Include dependency graph for remoteControl.h:



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



Macros

- `#define RC_BUTTON_STANDBY 12`
- `#define RC_BUTTON_SCREEN 11`
- `#define RC_BUTTON_LANG 15`
- `#define RC_BUTTON_SUBTTL 31`
- `#define RC_BUTTON_INTERNET 46`
- `#define RC_BUTTON_RED 55`
- `#define RC_BUTTON_GREEN 54`
- `#define RC_BUTTON_YELLOW 50`
- `#define RC_BUTTON_BLUE 52`
- `#define RC_BUTTON_0 0`
- `#define RC_BUTTON_1 1`
- `#define RC_BUTTON_2 2`
- `#define RC_BUTTON_3 3`
- `#define RC_BUTTON_4 4`
- `#define RC_BUTTON_5 5`
- `#define RC_BUTTON_6 6`
- `#define RC_BUTTON_7 7`
- `#define RC_BUTTON_8 8`
- `#define RC_BUTTON_9 9`
- `#define RC_BUTTON_TELE_TEXT 60`
- `#define RC_BUTTON_SWAP 34`
- `#define RC_BUTTON_OK 53`
- `#define RC_BUTTON_CURSOR_UP 20`
- `#define RC_BUTTON_CURSOR_DOWN 19`
- `#define RC_BUTTON_CURSOR_LEFT 21`
- `#define RC_BUTTON_CURSOR_RIGHT 22`
- `#define RC_BUTTON_BACK 10`
- `#define RC_BUTTON_MENU 48`
- `#define RC_BUTTON_EPG 47`
- `#define RC_BUTTON_FAV 40`
- `#define RC_BUTTON_SOURCE 56`
- `#define RC_BUTTON_INFO 18`
- `#define RC_BUTTON_PRESETS 14`
- `#define RC_BUTTON_SLEEP 42`
- `#define RC_BUTTON_VOLUME_UP 16`
- `#define RC_BUTTON_VOLUME_DOWN 17`
- `#define RC_BUTTON_MUTE 13`

- `#define RC_BUTTON_CHANNEL_UP` 32
- `#define RC_BUTTON_CHANNEL_DOWN` 33
- `#define RC_BUTTON_PAUSE` 48
- `#define RC_BUTTON_REWIND` 50
- `#define RC_BUTTON_WIND` 52
- `#define RC_BUTTON_PLAY` 53
- `#define RC_BUTTON_STOP` 54
- `#define RC_BUTTON_RECORD` 55

Functions

- void `Sys_Init_RemoteControl` (void)
- void `Sys_Start_RemoteControl` (void)
- bool `Sys_RemoteC_Received_New_Data` (void)
- uint8 `Sys_RemoteC_Get_CheckBit` (void)
- uint8 `Sys_RemoteC_Get_Address` (void)
- uint8 `Sys_RemoteC_Get_Data` (void)
- void `Sys_Receive_RemoteControl_Msg` (void)

8.32.1 Detailed Description

It declares functions to receive and decode messages from a remote control.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org
Yuri Kaszubowski Lopes yurikazuba@gmail.com

Version

1.0

Date

27 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.32.2 Macro Definition Documentation

8.32.2.1 `#define RC_BUTTON_0` 0

Value for the 0 button (RC-5 coding for a Toshiba RC-3910)

Definition at line 60 of file `remoteControl.h`.

8.32.2.2 `#define RC_BUTTON_1` 1

Value for the 1 button (RC-5 coding for a Toshiba RC-3910)

Definition at line 61 of file `remoteControl.h`.

8.32.2.3 #define RC_BUTTON_2 2

Value for the 2 button (RC-5 coding for a Toshiba RC-3910)

Definition at line 62 of file remoteControl.h.

8.32.2.4 #define RC_BUTTON_3 3

Value for the 3 button (RC-5 coding for a Toshiba RC-3910)

Definition at line 63 of file remoteControl.h.

8.32.2.5 #define RC_BUTTON_4 4

Value for the 4 button (RC-5 coding for a Toshiba RC-3910)

Definition at line 64 of file remoteControl.h.

8.32.2.6 #define RC_BUTTON_5 5

Value for the 5 button (RC-5 coding for a Toshiba RC-3910)

Definition at line 65 of file remoteControl.h.

8.32.2.7 #define RC_BUTTON_6 6

Value for the 6 button (RC-5 coding for a Toshiba RC-3910)

Definition at line 66 of file remoteControl.h.

8.32.2.8 #define RC_BUTTON_7 7

Value for the 7 button (RC-5 coding for a Toshiba RC-3910)

Definition at line 67 of file remoteControl.h.

8.32.2.9 #define RC_BUTTON_8 8

Value for the 8 button (RC-5 coding for a Toshiba RC-3910)

Definition at line 68 of file remoteControl.h.

8.32.2.10 #define RC_BUTTON_9 9

Value for the 9 button (RC-5 coding for a Toshiba RC-3910)

Definition at line 69 of file remoteControl.h.

8.32.2.11 #define RC_BUTTON_BACK 10

Value for the back button (RC-5 coding for a Toshiba RC-3910)

Definition at line 78 of file remoteControl.h.

8.32.2.12 #define RC_BUTTON_BLUE 52

Value for the blue button (RC-5 coding for a Toshiba RC-3910)

Definition at line 58 of file remoteControl.h.

8.32.2.13 #define RC_BUTTON_CHANNEL_DOWN 33

Value for the channel down button (RC-5 coding for a Toshiba RC-3910)

Definition at line 92 of file remoteControl.h.

8.32.2.14 #define RC_BUTTON_CHANNEL_UP 32

Value for the channel up button (RC-5 coding for a Toshiba RC-3910)

Definition at line 91 of file remoteControl.h.

8.32.2.15 #define RC_BUTTON_CURSOR_DOWN 19

Value for the courser down button (RC-5 coding for a Toshiba RC-3910)

Definition at line 75 of file remoteControl.h.

8.32.2.16 #define RC_BUTTON_CURSOR_LEFT 21

Value for the courser left button (RC-5 coding for a Toshiba RC-3910)

Definition at line 76 of file remoteControl.h.

8.32.2.17 #define RC_BUTTON_CURSOR_RIGHT 22

Value for the courser right button (RC-5 coding for a Toshiba RC-3910)

Definition at line 77 of file remoteControl.h.

8.32.2.18 #define RC_BUTTON_CURSOR_UP 20

Value for the cursor up button (RC-5 coding for a Toshiba RC-3910)

Definition at line 74 of file remoteControl.h.

8.32.2.19 #define RC_BUTTON_EPG 47

Value for the epg button (RC-5 coding for a Toshiba RC-3910)

Definition at line 80 of file remoteControl.h.

8.32.2.20 #define RC_BUTTON_FAV 40

Value for the favourite button (RC-5 coding for a Toshiba RC-3910)

Definition at line 81 of file remoteControl.h.

8.32.2.21 #define RC_BUTTON_GREEN 54

Value for the green button (RC-5 coding for a Toshiba RC-3910)

Definition at line 56 of file remoteControl.h.

8.32.2.22 #define RC_BUTTON_INFO 18

Value for the info button (RC-5 coding for a Toshiba RC-3910)

Definition at line 84 of file remoteControl.h.

8.32.2.23 #define RC_BUTTON_INTERNET 46

Value for the internet button (RC-5 coding for a Toshiba RC-3910)

Definition at line 53 of file remoteControl.h.

8.32.2.24 #define RC_BUTTON_LANG 15

Value for the language button (RC-5 coding for a Toshiba RC-3910)

Definition at line 51 of file remoteControl.h.

8.32.2.25 #define RC_BUTTON_MENU 48

Value for the menu button (RC-5 coding for a Toshiba RC-3910)

Definition at line 79 of file remoteControl.h.

8.32.2.26 #define RC_BUTTON_MUTE 13

Value for the mute button (RC-5 coding for a Toshiba RC-3910)

Definition at line 90 of file remoteControl.h.

8.32.2.27 #define RC_BUTTON_OK 53

Value for the OK button (RC-5 coding for a Toshiba RC-3910)

Definition at line 73 of file remoteControl.h.

8.32.2.28 #define RC_BUTTON_PAUSE 48

Value for the pause button (RC-5 coding for a Toshiba RC-3910)

Definition at line 95 of file remoteControl.h.

8.32.2.29 #define RC_BUTTON_PLAY 53

Value for the play button (RC-5 coding for a Toshiba RC-3910)

Definition at line 98 of file remoteControl.h.

8.32.2.30 #define RC_BUTTON_PRESETS 14

Value for the preset button (RC-5 coding for a Toshiba RC-3910)

Definition at line 85 of file remoteControl.h.

8.32.2.31 #define RC_BUTTON_RECORD 55

Value for the record button (RC-5 coding for a Toshiba RC-3910)

Definition at line 100 of file remoteControl.h.

8.32.2.32 #define RC_BUTTON_RED 55

Value for the red button (RC-5 coding for a Toshiba RC-3910)

Definition at line 55 of file remoteControl.h.

8.32.2.33 #define RC_BUTTON_REWIND 50

Value for the rewind button (RC-5 coding for a Toshiba RC-3910)

Definition at line 96 of file remoteControl.h.

8.32.2.34 #define RC_BUTTON_SCREEN 11

Value for the screen button (RC-5 coding for a Toshiba RC-3910)

Definition at line 50 of file remoteControl.h.

8.32.2.35 #define RC_BUTTON_SLEEP 42

Value for the sleep button (RC-5 coding for a Toshiba RC-3910)

Definition at line 86 of file remoteControl.h.

8.32.2.36 #define RC_BUTTON_SOURCE 56

Value for the source button (RC-5 coding for a Toshiba RC-3910)

Definition at line 83 of file remoteControl.h.

8.32.2.37 #define RC_BUTTON_STANDBY 12

Value for the standby button (RC-5 coding for a Toshiba RC-3910)

Definition at line 48 of file remoteControl.h.

8.32.2.38 #define RC_BUTTON_STOP 54

Value for the stop button (RC-5 coding for a Toshiba RC-3910)

Definition at line 99 of file remoteControl.h.

8.32.2.39 #define RC_BUTTON_SUBTTL 31

Value for the subtitle button (RC-5 coding for a Toshiba RC-3910)

Definition at line 52 of file remoteControl.h.

8.32.2.40 #define RC_BUTTON_SWAP 34

Value for the swap button (RC-5 coding for a Toshiba RC-3910)

Definition at line 71 of file remoteControl.h.

8.32.2.41 #define RC_BUTTON_TELE_TEXT 60

Value for the tele text button (RC-5 coding for a Toshiba RC-3910)

Definition at line 70 of file remoteControl.h.

8.32.2.42 #define RC_BUTTON_VOLUME_DOWN 17

Value for the volume down button (RC-5 coding for a Toshiba RC-3910)

Definition at line 89 of file remoteControl.h.

8.32.2.43 #define RC_BUTTON_VOLUME_UP 16

Value for the volume up button (RC-5 coding for a Toshiba RC-3910)

Definition at line 88 of file remoteControl.h.

8.32.2.44 #define RC_BUTTON_WIND 52

Value for the wind button (RC-5 coding for a Toshiba RC-3910)

Definition at line 97 of file remoteControl.h.

8.32.2.45 #define RC_BUTTON_YELLOW 50

Value for the yellow button (RC-5 coding for a Toshiba RC-3910)

Definition at line 57 of file remoteControl.h.

8.32.3 Function Documentation**8.32.3.1 void Sys_Init_RemoteControl (void) [inline]**

This function initialises the handler of the remote control to receive signals from the remote control.

Definition at line 34 of file remoteControl.c.

8.32.3.2 void Sys_Receive_RemoteControl_Msg (void)

This function reads a remote control signal and reads it's transmitted value. When a signal arrives, an external interrupt is triggered. The remaining values are obtained by using time not interrupts. < The initial state of the state machine to decode a remote control message

- < Cycles that are needed to wait a single bit duration
- < Cycles that are needed to wait a quarter of a single bit duration
- < Cycles that are needed to wait at the beginning of a message
- < Cycles that are needed to wait a half of a single bit duration
- < Cycles that are needed to wait a single bit duration
- < The initial state of the state machine to decode a remote control message
- < ID of the event that is sent after receiving a remote control signal

Definition at line 52 of file remoteControl.c.

8.32.3.3 uint8 Sys_RemoteC_Get_Address (void)

This function returns the address of an remote control.

Returns

address of the remote control

Definition at line 138 of file remoteControl.c.

8.32.3.4 uint8 Sys_RemoteC_Get_CheckBit (void)

This function returns the check bit value of an remote control to indicate if a button was pressed continuously or sequential.

Returns

bit to indicate the check bit

Definition at line 128 of file remoteControl.c.

8.32.3.5 uint8 Sys_RemoteC_Get_Data (void)

returns the value received by the remote control

Returns

value received by the remote control

Definition at line 148 of file remoteControl.c.

8.32.3.6 bool Sys_RemoteC_Received_New_Data (void)

8.32.3.7 void Sys_Start_RemoteControl (void) [inline]

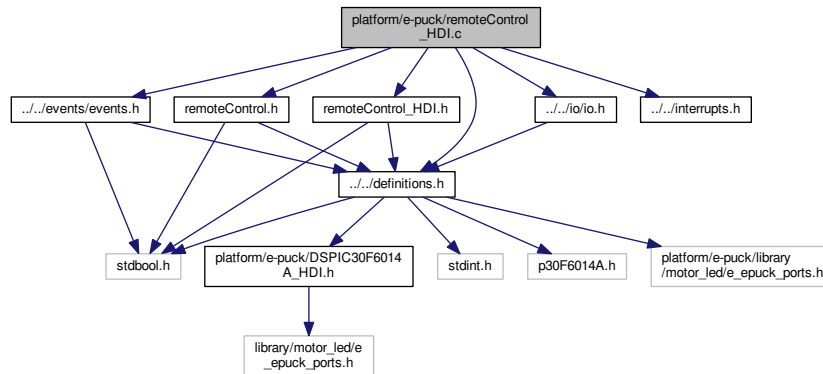
This function start the handler of the remote control to receive signals from the remote control.

Definition at line 43 of file remoteControl.c.

8.33 platform/e-puck/remoteControl_HDI.c File Reference

Hardware dependent implementations to receive and decode messages from a remote control.

```
#include "remoteControl_HDI.h"
#include "../..//definitions.h"
#include "../..//interrupts.h"
#include "../..//io/io.h"
#include "remoteControl.h"
#include "../..//events/events.h"
Include dependency graph for remoteControl_HDI.c:
```



Functions

- void [Sys_Init_RemoteControl_HDI](#) (void)
- void [Sys_Start_RemoteControl_HDI](#) (void)
- void [__attribute__](#) ((__interrupt__, auto_psv))

Variables

- bool [message_arriving](#) = false
- [sint8 waiting_cycles](#) = 20
- [uint rx_buffer](#) = 0
- bool [isNewDataAvailable](#) = false
- [sint8 receiving_bit](#) = -1

8.33.1 Detailed Description

Hardware dependent implementations to receive and decode messages from a remote control.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org
 Yuri Kaszubowski Lopes yurikazuba@gmail.com

Version

1.0

Date

27 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.33.2 Function Documentation

8.33.2.1 void __attribute__ ((__interrupt__, auto_psv))

This function is executed at the arrival of a new remote control message. < Cycles that are needed to wait a single bit duration

< Cycles that are needed to wait a quarter of a single bit duration

< Cycles that are needed to wait at the beginning of a message

Definition at line 69 of file remoteControl_HDI.c.

8.33.2.2 void Sys_Init_RemoteControl_HDI (void) [inline]

This function initialises the handler of the remote control to receive signals from the remote control. < interrupt priority for the remote control interrupt

< ID of the event that is sent after receiving a remote control signal

Definition at line 40 of file remoteControl_HDI.c.

8.33.2.3 void Sys_Start_RemoteControl_HDI (void) [inline]

This function start the handler of the remote control to receive signals from the remote control. clear to IRQ flag

Definition at line 59 of file remoteControl_HDI.c.

8.33.3 Variable Documentation

8.33.3.1 bool isNewDataAvailable = false

a flag to indicate that a new message was received

Definition at line 31 of file remoteControl_HDI.c.

8.33.3.2 bool message_arriving = false

A flag that is set as soon as a message is recieved

Definition at line 26 of file remoteControl_HDI.c.

8.33.3.3 sint8 receiving_bit = -1

The initial state of the state machine to decode a remote control message State indicator (for the state machine)

Definition at line 33 of file remoteControl_HDI.c.

8.33.3.4 uint rx_buffer = 0

The initial state of the state machine to decode a remote control message

Definition at line 29 of file remoteControl_HDI.c.

8.33.3.5 sint8 waiting_cycles = 20

The cycles that need to be waited until the next stage (set for 100us)

Definition at line 27 of file remoteControl_HDI.c.

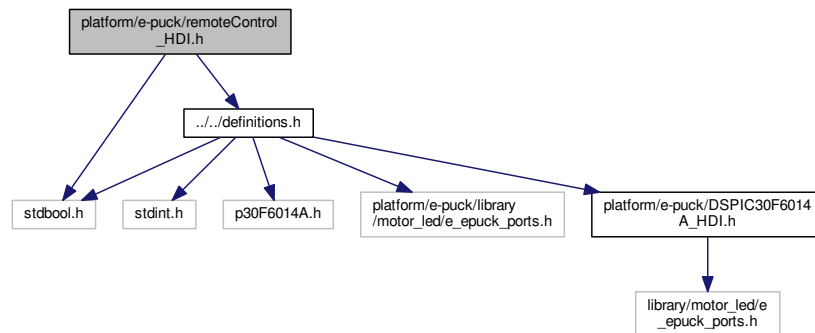
8.34 platform/e-puck/remoteControl_HDI.h File Reference

Hardware dependent implementations to receive and decode messages from a remote control.

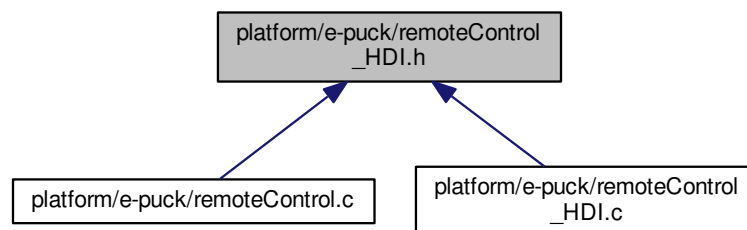
```
#include <stdbool.h>
```

```
#include "../..../definitions.h"
```

Include dependency graph for remoteControl_HDI.h:



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



Macros

- `#define RC_WAIT_FOR_QUARTERBIT 4`
- `#define RC_WAIT_FOR_HALFBIT 8`
- `#define RC_WAIT_FOR_BIT 18`
- `#define RC_WAIT_INITIALLY RC_WAIT_FOR_BIT+RC_WAIT_FOR_QUARTERBIT`
- `#define RC_NOT_STARTED -1`

Functions

- void `Sys_Init_RemoteControl_HDI` (void)
- void `Sys_Start_RemoteControl_HDI` (void)

Variables

- bool `message_arriving`

- [sint8 waiting_cycles](#)
- [uint rx_buffer](#)
- [bool isNewDataAvailable](#)
- [sint8 receiving_bit](#)

8.34.1 Detailed Description

Hardware dependent implementations to receive and decode messages from a remote control.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org
Yuri Kaszubowski Lopes yurikazuba@gmail.com

Version

1.0

Date

27 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.34.2 Macro Definition Documentation

8.34.2.1 `#define RC_NOT_STARTED -1`

The initial state of the state machine to decode a remote control message

Definition at line 33 of file remoteControl_HDI.h.

8.34.2.2 `#define RC_WAIT_FOR_BIT 18`

Cycles that are needed to wait a single bit duration

Definition at line 31 of file remoteControl_HDI.h.

8.34.2.3 `#define RC_WAIT_FOR_HALFBIT 8`

Cycles that are needed to wait a half of a single bit duration

Definition at line 30 of file remoteControl_HDI.h.

8.34.2.4 `#define RC_WAIT_FOR_QUARTERBIT 4`

Cycles that are needed to wait a quarter of a single bit duration

Definition at line 29 of file remoteControl_HDI.h.

8.34.2.5 `#define RC_WAIT_INITIALLY RC_WAIT_FOR_BIT+RC_WAIT_FOR_QUARTERBIT`

Cycles that are needed to wait at the beginning of a message

Definition at line 32 of file remoteControl_HDI.h.

8.34.3 Function Documentation

8.34.3.1 void Sys_Init_RemoteControl_HDI(void) [inline]

This function initialises the handler of the remote control to receive signals from the remote control. < interrupt priority for the remote control interrupt

< ID of the event that is sent after receiving a remote control signal

Definition at line 40 of file remoteControl_HDI.c.

8.34.3.2 void Sys_Start_RemoteControl_HDI(void) [inline]

This function start the handler of the remote control to receive signals from the remote control. clear to IRQ flag

Definition at line 59 of file remoteControl_HDI.c.

8.34.4 Variable Documentation

8.34.4.1 bool isNewDataAvailable

a flag to indicate that a new message was received

Definition at line 31 of file remoteControl_HDI.c.

8.34.4.2 bool message_arriving

A flag that is set as soon as a message is recieved

Definition at line 26 of file remoteControl_HDI.c.

8.34.4.3 sint8 receiving_bit

The initial state of the state machine to decode a remote control message State indicator (for the state machine)

Definition at line 33 of file remoteControl_HDI.c.

8.34.4.4 uint rx_buffer

The initial state of the state machine to decode a remote control message

Definition at line 29 of file remoteControl_HDI.c.

8.34.4.5 sint8 waiting_cycles

The cycles that need to be waited until the next stage (set for 100us)

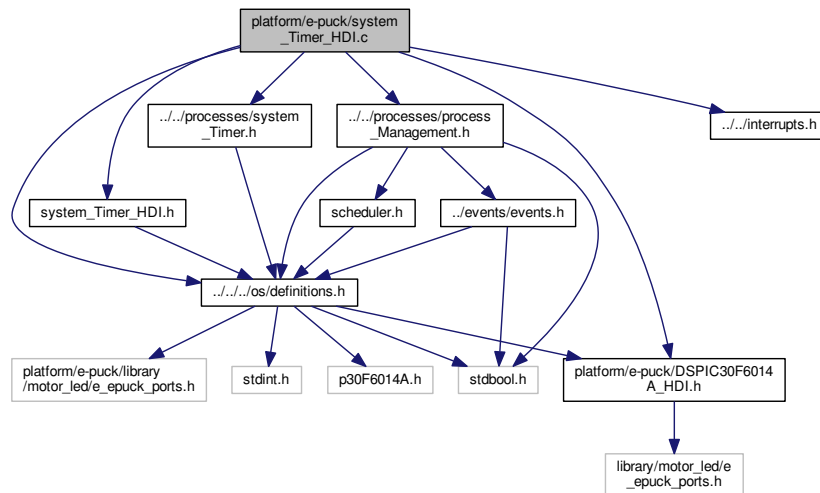
Definition at line 27 of file remoteControl_HDI.c.

8.35 platform/e-puck/system_Timer_HDI.c File Reference

Hardware dependent implementations to initialise, configure and run the system Timer.

```
#include "system_Timer_HDI.h"
#include "../processes/system_Timer.h"
#include "../processes/process_Management.h"
#include "DSPIC30F6014A_HDI.h"
#include "../interrupts.h"
#include "../definitions.h"
```

Include dependency graph for system_Timer_HDI.c:



Functions

- void [Sys_Init_SystemTimer_HDI](#) (pFunction scheduler)
- void [Sys_Start_SystemTimer_HDI](#) ()
- void [Sys_Stop_SystemTimer_HDI](#) ()
- void [Sys_Continue_SystemTimer_HDI](#) ()
- void [Sys_Reset_SystemTimer_HDI](#) ()
- void [__attribute__](#) ((interrupt, no_auto_psv))
- void [Sys_Disable_TimerInterrupt_HDI](#) (void)
- void [Sys_Enable_TimerInterrupt_HDI](#) (void)
- void [Sys_Force_TimerInterrupt_HDI](#) (void)

Variables

- pFunction [sys_process_scheduler](#) = 0

8.35.1 Detailed Description

Hardware dependent implementations to initialise, configure and run the system Timer.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

07 July 2014

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.35.2 Function Documentation

8.35.2.1 void __attribute__ ((interrupt, no_auto_psv))

This Function starts the task-scheduling algorithm

Definition at line 105 of file system_Timer_HDI.c.

8.35.2.2 void Sys_Continue_SystemTimer_HDI () [inline]

This Function deactivated the Timer2 Interrupt

Definition at line 84 of file system_Timer_HDI.c.

8.35.2.3 void Sys_Disable_TimerInterrupt_HDI (void) [inline]

Disables the Timer2 interrupt and sets the interrupt flag to 0

Definition at line 123 of file system_Timer_HDI.c.

8.35.2.4 void Sys_Enable_TimerInterrupt_HDI (void) [inline]

Enables the Timer2 interrupt and leaves the interrupt flag to its value. If the flag was set, the Timer1 interrupt will be emitted after executing this function.

Definition at line 133 of file system_Timer_HDI.c.

8.35.2.5 void Sys_Force_TimerInterrupt_HDI (void) [inline]

forces the Timer2 interrupt.

Definition at line 142 of file system_Timer_HDI.c.

8.35.2.6 void Sys_Init_SystemTimer_HDI (pFunction scheduler)

This Function sets the Timer0 of the DSPIC 30F6014A for timer intervals of 10 ms. The timer will be startet with Start_SystemTimer_HDI()

Parameters

<i>in, out</i>	<i>scheduler</i>	This is a pointer t an callback function, which schuld becalled whenever a timer interrupt is emmitted.
----------------	------------------	---

Definition at line 34 of file system_Timer_HDI.c.

8.35.2.7 void Sys_Reset_SystemTimer_HDI () [inline]

This Function resets the Timer2 value

Definition at line 96 of file system_Timer_HDI.c.

8.35.2.8 void Sys_Start_SystemTimer_HDI (void)

This Function starts the Timer2 of the DSPIC 30F6014A for timer intervals of 10 ms. The MUST be initialised first with Init_SystemTimer_HDI() < interrupt priority for the system timer interrupt

Definition at line 59 of file system_Timer_HDI.c.

8.35.2.9 void Sys_Stop_SystemTimer_HDI () [inline]

This Function activated the Timer2 Interrupt

Definition at line 72 of file system_Timer_HDI.c.

8.35.3 Variable Documentation

8.35.3.1 pFunction sys_process_scheduler = 0

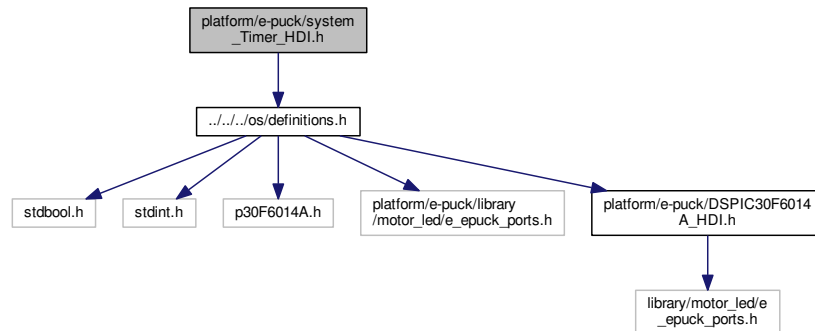
Definition at line 26 of file system_Timer_HDI.c.

8.36 platform/e-puck/system_Timer_HDI.h File Reference

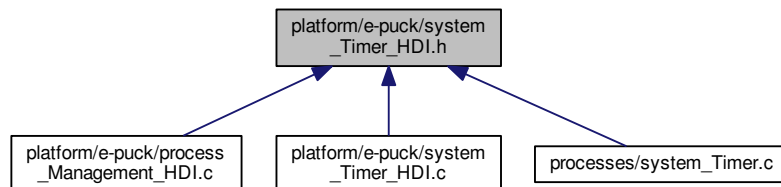
Hardware dependent implementations to initialise, configure and run the system Timer.

```
#include "../../os/definitions.h"
```

Include dependency graph for system_Timer_HDI.h:



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



Functions

- void [Sys_Init_SystemTimer_HDI](#) (pFunction)
- void [Sys_Start_SystemTimer_HDI](#) (void)
- void [Sys_Stop_SystemTimer_HDI](#) ()
- void [Sys_Continue_SystemTimer_HDI](#) ()
- void [Sys_Disable_TimerInterrupt_HDI](#) (void)
- void [Sys_Enable_TimerInterrupt_HDI](#) (void)
- void [Sys_Force_TimerInterrupt_HDI](#) (void)
- void [Sys_Reset_SystemTimer_HDI](#) ()
- void [Sys_todo_SystemTimer](#) ()

Variables

- [pFunction sys_process_scheduler](#)

8.36.1 Detailed Description

Hardware dependent implementations to initialise, configure and run the system Timer.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

07 July 2014

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.36.2 Function Documentation

8.36.2.1 void Sys_Continue_SystemTimer_HDI () [inline]

This Function deactivated the Timer2 Interrupt

Definition at line 84 of file system_Timer_HDI.c.

8.36.2.2 void Sys_Disable_TimerInterrupt_HDI (void) [inline]

Disables the Timer2 interrupt and sets the interrupt flag to 0

Definition at line 123 of file system_Timer_HDI.c.

8.36.2.3 void Sys_Enable_TimerInterrupt_HDI (void) [inline]

Enables the Timer2 interrupt and leaves the interrupt flag to its value. If the flag was set, the Timer1 interrupt will be emitted after executing this function.

Definition at line 133 of file system_Timer_HDI.c.

8.36.2.4 void Sys_Force_TimerInterrupt_HDI (void) [inline]

forces the Timer2 interrupt.

Definition at line 142 of file system_Timer_HDI.c.

8.36.2.5 void Sys_Init_SystemTimer_HDI (pFunction scheduler)

This Function sets the Timer0 of the DSPIC 30F6014A for timer intervals of 10 ms. The timer will be startet with Start_SystemTimer_HDI()

Parameters

in, out	<i>scheduler</i>	This is a pointer to a callback function, which should be called whenever a timer interrupt is emitted.
---------	------------------	---

Definition at line 34 of file system_Timer_HDI.c.

8.36.2.6 void Sys_Reset_SystemTimer_HDI () [inline]

This Function resets the Timer2 value

Definition at line 96 of file system_Timer_HDI.c.

8.36.2.7 void Sys_Start_SystemTimer_HDI (void)

This Function starts the Timer2 of the DSPIC 30F6014A for timer intervals of 10 ms. The MUST be initialised first with Init_SystemTimer_HDI() < interrupt priority for the system timer interrupt

Definition at line 59 of file system_Timer_HDI.c.

8.36.2.8 void Sys_Stop_SystemTimer_HDI () [inline]

This Function activated the Timer2 Interrupt

Definition at line 72 of file system_Timer_HDI.c.

8.36.2.9 void Sys_todo_SystemTimer () [inline]

This function is executed periodically by the system timer interrupt. It kills all zombies, executes event handlers and executes the scheduling algorithm.

Definition at line 73 of file system_Timer.c.

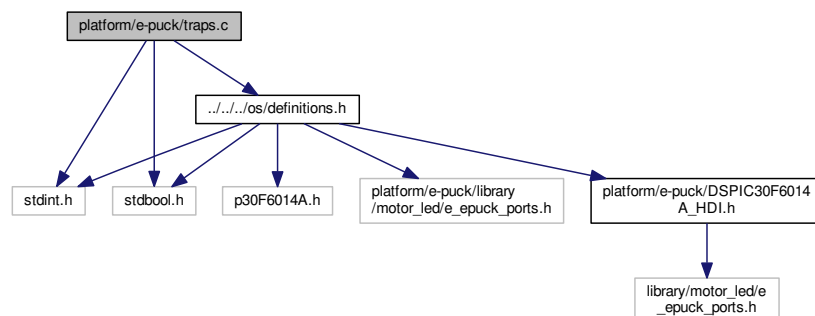
8.36.3 Variable Documentation**8.36.3.1 pFunction sys_process_scheduler**

Definition at line 26 of file system_Timer_HDI.c.

8.37 platform/e-puck/traps.c File Reference

Hardware dependent implementations to catch hardware traps.

```
#include <stdint.h>
#include <stdbool.h>
#include "../os/definitions.h"
Include dependency graph for traps.c:
```

**Functions**

- void [__attribute__](#) ((interrupt, no_auto_psv))

8.37.1 Detailed Description

Hardware dependent implementations to catch hardware traps.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

07 July 2014

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.37.2 Function Documentation**8.37.2.1 void __attribute__((interrupt, no_auto_psv))**

Address error trap.

This function is called when an address error occurs. That means that a call address of a function or in the stack addresses an area outside the memory. Similarly, if a pointer points to memory outside the range, this trap happens.

Stack error trap.

This function is called when an stack error occurs. That means that the stack pointer, stack pointer limit, or frame pointer are pointing outside their range.

Math error trap.

This function is called when an math error occurs. That means an illegal math operation was performed (such as division by 0 or NaN).

Alternative Oscillator fail trap.

This function is called when an oscillator fail occurs. This should never happen.

Alternative address error trap.

This function is called when an address error occurs. That means that a call address of a function or in the stack addresses an area outside the memory. Similarly, if a pointer points to memory outside the range, this trap happens.

Alternative stack error trap.

This function is called when an stack error occurs. That means that the stack pointer, stack pointer limit, or frame pointer are pointing outside their range.

Alternative math error trap.

This function is called when an math error occurs. That means an illegal math operation was performed (such as division by 0 or NaN).

Default interrupt service routine.

This function is called when no other interrupt routine is specified.

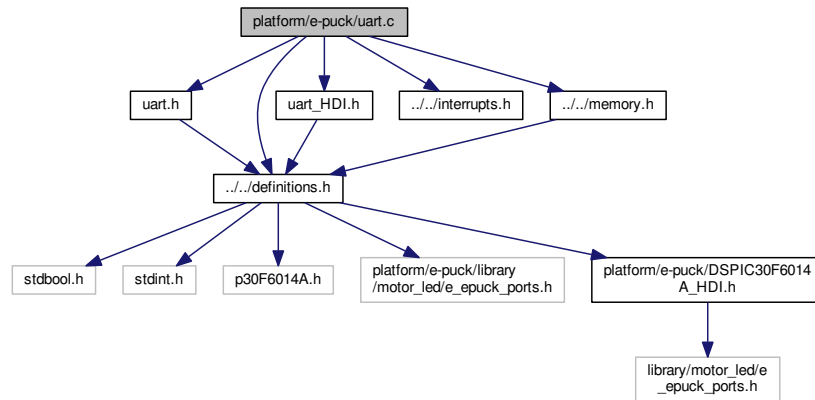
Definition at line 68 of file traps.c.

8.38 platform/e-puck/uart.c File Reference

It declares functions to transmit bytes via UART.

```
#include "uart.h"
#include "uart_HDI.h"
#include "../definitions.h"
#include "../interrupts.h"
#include "../memory.h"
```

Include dependency graph for uart.c:



Macros

- `#define` [SYS_UART1_BAUDRATE](#) 115000
- `#define` [SYS_UART2_BAUDRATE](#) 115000

Functions

- void [Sys_Init_UART1](#) (void)
- void [Sys_Init_UART2](#) (void)
- void [Sys_Start_UART1](#) (void)
- void [Sys_Start_UART2](#) (void)
- void [Sys_SetReadingFunction_UART1](#) (pUART_reader func)
- void [Sys_SetReadingFunction_UART2](#) (pUART_reader func)
- void [Sys_Writeto_UART1](#) (void *data, uint length)
- void [Sys_Writeto_UART2](#) (void *data, uint length)

8.38.1 Detailed Description

It declares functions to transmit bytes via UART.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

27 August 2015

Copyrightadapted FreeBSD License (see <http://openswarm.org/license>)**8.38.2 Macro Definition Documentation****8.38.2.1 #define SYS_UART1_BAUDRATE 115000**

Baudrate for UART 1 (bits/s)

Definition at line 22 of file uart.c.

8.38.2.2 #define SYS_UART2_BAUDRATE 115000

Baudrate for UART 2 (bits/s)

Definition at line 23 of file uart.c.

8.38.3 Function Documentation**8.38.3.1 void Sys_Init_UART1 (void) [inline]**

This function initialises UART1.

Definition at line 31 of file uart.c.

8.38.3.2 void Sys_Init_UART2 (void) [inline]

This function initialises UART2.

Definition at line 40 of file uart.c.

8.38.3.3 void Sys_SetReadingFunction_UART1 (pUART_reader func)

defines a function that processes received bytes (UART1). This defined callback function is only executed once by arrival of one byte.

Parameters

<i>in</i>	<i>func</i>	pointer to the function that should process the received byte(s).
-----------	-------------	---

Definition at line 72 of file uart.c.

8.38.3.4 void Sys_SetReadingFunction_UART2 (pUART_reader func)

defines a function that processes received bytes (UART2). This defined callback function is only executed once by arrival of one byte.

Parameters

<i>in</i>	<i>func</i>	pointer to the function that should process the received byte(s).
-----------	-------------	---

Definition at line 82 of file uart.c.

8.38.3.5 void Sys_Start_UART1 (void) [inline]

This function starts UART1.

Note

When executed this function, bytes can be received or transmitted at any time.

Definition at line 51 of file uart.c.

8.38.3.6 `void Sys_Start_UART2 (void) [inline]`

This function starts UART2.

Note

When executed this function, bytes can be received or transmitted at any time.

Definition at line 62 of file uart.c.

8.38.3.7 `void Sys_Writeto_UART1 (void * data, uint length)`

This function writes sequentially the bytes on the UART1.

Note

The data will be put into a queue, where it will be sent as soon as the UART is idle

Parameters

in	<i>data</i>	pointer to the bytes that should be transmitted.
in	<i>length</i>	number of bytes to send.

Definition at line 95 of file uart.c.

8.38.3.8 `void Sys_Writeto_UART2 (void * data, uint length)`

This function writes sequentially the bytes on the UART2.

Note

The data will be put into a queue, where it will be sent as soon as the UART is idle

Parameters

in	<i>data</i>	pointer to the bytes that should be transmitted.
in	<i>length</i>	number of bytes to send.

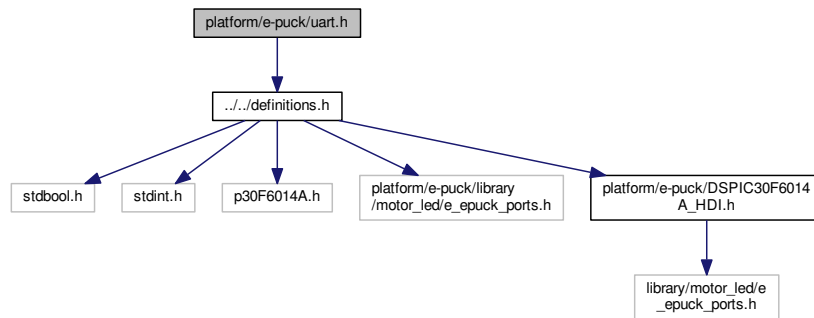
Definition at line 134 of file uart.c.

8.39 platform/e-puck/uart.h File Reference

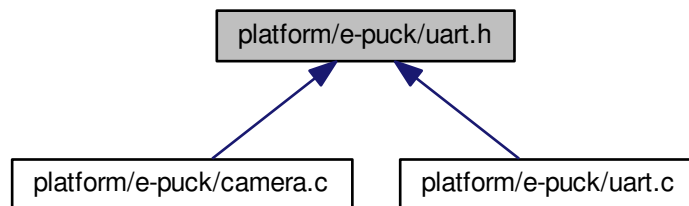
It declares functions to transmit bytes via UART.

```
#include "../../definitions.h"
```

Include dependency graph for uart.h:



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



Functions

- void `Sys_Init_UART1` (void)
- void `Sys_Init_UART2` (void)
- void `Sys_Start_UART1` (void)
- void `Sys_Start_UART2` (void)
- void `Sys_SetReadingFunction_UART1` (pUART_reader func)
- void `Sys_SetReadingFunction_UART2` (pUART_reader func)
- void `Sys_Writeto_UART1` (void *data, uint length)
- void `Sys_Writeto_UART2` (void *data, uint length)

8.39.1 Detailed Description

It declares functions to transmit bytes via UART.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

27 August 2015

Copyrightadapted FreeBSD License (see <http://openswarm.org/license>)**8.39.2 Function Documentation****8.39.2.1 void Sys_Init_UART1 (void) [inline]**

This function initialises UART1.

Definition at line 31 of file uart.c.

8.39.2.2 void Sys_Init_UART2 (void) [inline]

This function initialises UART2.

Definition at line 40 of file uart.c.

8.39.2.3 void Sys_SetReadingFunction_UART1 (pUART_reader func)

defines a function that processes received bytes (UART1). This defined callback function is only executed once by arrival of one byte.

Parameters

<i>in</i>	<i>func</i>	pointer to the function that should process the received byte(s).
-----------	-------------	---

Definition at line 72 of file uart.c.

8.39.2.4 void Sys_SetReadingFunction_UART2 (pUART_reader func)

defines a function that processes received bytes (UART2). This defined callback function is only executed once by arrival of one byte.

Parameters

<i>in</i>	<i>func</i>	pointer to the function that should process the received byte(s).
-----------	-------------	---

Definition at line 82 of file uart.c.

8.39.2.5 void Sys_Start_UART1 (void) [inline]

This function starts UART1.

Note

When executed this function, bytes can be received or transmitted at any time.

Definition at line 51 of file uart.c.

8.39.2.6 void Sys_Start_UART2 (void) [inline]

This function starts UART2.

Note

When executed this function, bytes can be received or transmitted at any time.

Definition at line 62 of file uart.c.

8.39.2.7 void Sys_Writeto_UART1 (void * *data*, uint *length*)

This function writes sequentially the bytes on the UART1.

Note

The data will be put into a queue, where it will be sent as soon as the UART is idle

Parameters

in	<i>data</i>	pointer to the bytes that should be transmitted.
in	<i>length</i>	number of bytes to send.

Definition at line 95 of file uart.c.

8.39.2.8 void Sys_Writeto_UART2 (void * *data*, uint *length*)

This function writes sequentially the bytes on the UART2.

Note

The data will be put into a queue, where it will be sent as soon as the UART is idle

Parameters

in	<i>data</i>	pointer to the bytes that should be transmitted.
in	<i>length</i>	number of bytes to send.

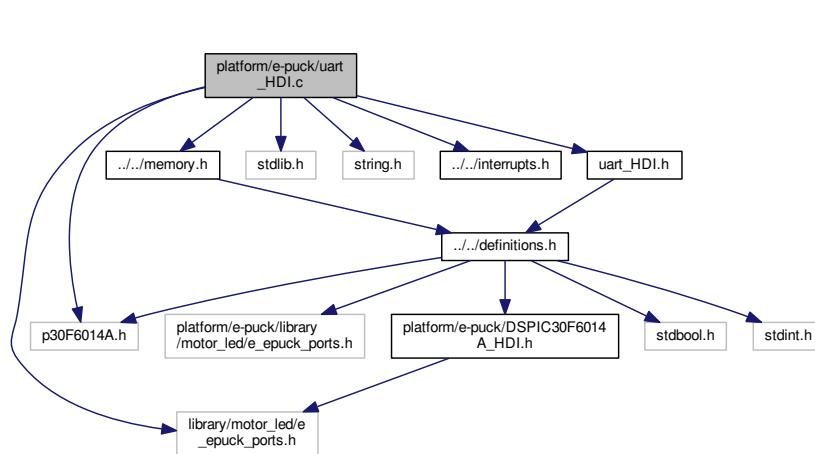
Definition at line 134 of file uart.c.

8.40 platform/e-puck/uart_HDI.c File Reference

Hardware dependent implementations to transmit bytes via UART.

```
#include "uart_HDI.h"
#include <p30F6014A.h>
#include <stdlib.h>
#include <string.h>
#include "library/motor_led/e_epuck_ports.h"
#include "../interrupts.h"
#include "../memory.h"
```

Include dependency graph for uart_HDI.c:



Functions

- void [Sys_Init_UART1_HDI](#) (void)
- void [Sys_Init_UART2_HDI](#) (void)
- void [Sys_Start_UART1_HDI](#) (void)
- void [Sys_Start_UART2_HDI](#) (void)
- void [__attribute__](#) ((interrupt, auto_psv))
- void [Sys_Read_UART1_ISR](#) ()
- void [Sys_Write_UART1_ISR](#) ()
- void [Sys_Read_UART2_ISR](#) ()
- void [Sys_Write_UART2_ISR](#) ()

Variables

- [pUART_reader read_uart_1](#) = 0
- [pUART_reader read_uart_2](#) = 0
- [sys_uart_txdata * sys_UART1_TX_data](#) = 0
- [sys_uart_txdata * sys_UART2_TX_data](#) = 0
- [uint byte_counter_uart1](#) = 0
- [uint byte_counter_uart2](#) = 0

8.40.1 Detailed Description

Hardware dependent implementations to transmit bytes via UART.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

27 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.40.2 Function Documentation**8.40.2.1 void __attribute__ ((interrupt, auto_psv))**

UART1 reading interrupt.

Alternative UART1 reading interrupt.

UART1 writing interrupt.

Alternative UART1 writing interrupt.

UART2 reading interrupt.

Alternative UART2 reading interrupt.

UART2 writing interrupt.

Alternative UART2 writing interrupt.

Definition at line 136 of file uart_HDI.c.

8.40.2.2 void Sys_Init_UART1_HDI (void)

This function initialises UART1. < interrupt priority for the UART1 interrupt

< interrupt priority for the UART1 interrupt

< Baud rate for UART1

Definition at line 42 of file uart_HDI.c.

8.40.2.3 void Sys_Init_UART2_HDI (void)

This function initialises UART2. < Baud rate for UART2

< interrupt priority for the UART2 interrupt

< interrupt priority for the UART2 interrupt

Definition at line 79 of file uart_HDI.c.

8.40.2.4 void Sys_Read_UART1_ISR () [inline]

This function is executed at occurrence of the UART1 reading interrupt.

Definition at line 213 of file uart_HDI.c.

8.40.2.5 void Sys_Read_UART2_ISR () [inline]

This function is executed at occurrence of the UART2 reading interrupt.

Definition at line 271 of file uart_HDI.c.

8.40.2.6 void Sys_Start_UART1_HDI (void)

This function starts UART2.

Note

When executed this function, bytes can be received or transmitted at any time.

Definition at line 106 of file uart_HDL.c.

8.40.2.7 void Sys_Start_UART2_HDL (void)

This function starts UART2.

Note

When executed this function, bytes can be received or transmitted at any time.

Definition at line 122 of file uart_HDL.c.

8.40.2.8 void Sys_Write_UART1_ISR () [inline]

This function is executed at occurrence of the UART1 writing interrupt.

Definition at line 233 of file uart_HDL.c.

8.40.2.9 void Sys_Write_UART2_ISR () [inline]

This function is executed at occurrence of the UART1 writing interrupt.

Definition at line 291 of file uart_HDL.c.

8.40.3 Variable Documentation**8.40.3.1 uint byte_counter_uart1 = 0**

Bytes that were written

Definition at line 34 of file uart_HDL.c.

8.40.3.2 uint byte_counter_uart2 = 0

Bytes that were written

Definition at line 35 of file uart_HDL.c.

8.40.3.3 pUART_reader read_uart_1 = 0

pointer to the functions that processes incoming bytes from UART1

Definition at line 28 of file uart_HDL.c.

8.40.3.4 pUART_reader read_uart_2 = 0

pointer to the functions that processes incoming bytes from UART2

Definition at line 29 of file uart_HDL.c.

8.40.3.5 sys_uart_txdata* sys_UART1_TX_data = 0

Linked list of messages that need to be sent via UART1

Definition at line 31 of file uart_HDL.c.

8.40.3.6 sys_uart_txdata* sys_UART2_TX_data = 0

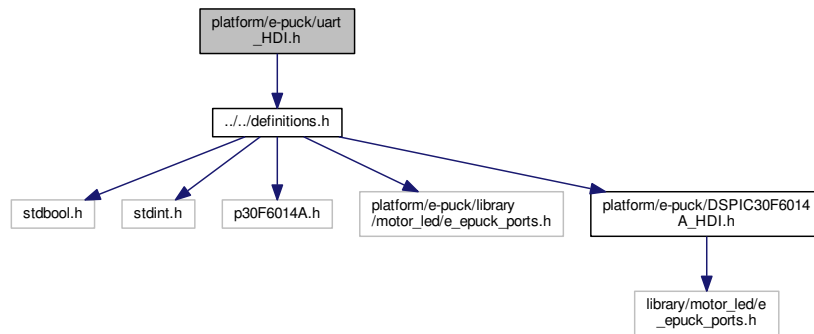
Linked list of messages that need to be sent via UART2

Definition at line 32 of file uart_HDL.c.

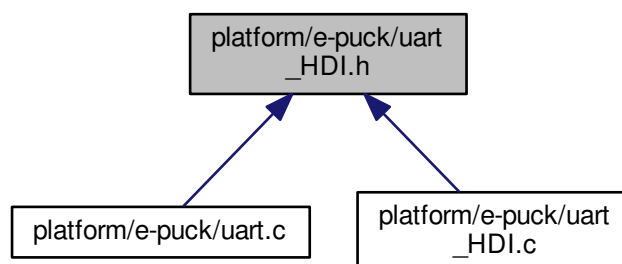
8.41 platform/e-puck/uart_HDI.h File Reference

Hardware dependent implementations to transmit bytes via UART.

```
#include "../definitions.h"
Include dependency graph for uart_HDI.h:
```



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



Data Structures

- struct [sys_uart_txdata](#)
Linked list element to store transmission data.

Macros

- `#define UART1_RX_RF2`
- `#define UART1_TX_RF3`
- `#define UART2_RX_RF4`
- `#define UART2_TX_RF5`
- `#define UART1_RX_DIR_TRISF2`
- `#define UART1_TX_DIR_TRISF3`
- `#define UART2_RX_DIR_TRISF4`

- `#define UART2_TX_DIR_TRISF5`
- `#define SYS_UART1_BAUDRATE 115000`
- `#define SYS_UART2_BAUDRATE 115000`

Functions

- void `Sys_Init_UART1_HDI` (void)
- void `Sys_Init_UART2_HDI` (void)
- void `Sys_Start_UART1_HDI` (void)
- void `Sys_Start_UART2_HDI` (void)
- void `Sys_Read_UART1_ISR` ()
- void `Sys_Write_UART1_ISR` ()
- void `Sys_Read_UART2_ISR` ()
- void `Sys_Write_UART2_ISR` ()

Variables

- `sys_uart_txdata * sys_UART1_TX_data`
- `sys_uart_txdata * sys_UART2_TX_data`
- `uint byte_counter_uart1`
- `uint byte_counter_uart2`
- `pUART_reader read_uart_1`
- `pUART_reader read_uart_2`

8.41.1 Detailed Description

Hardware dependent implementations to transmit bytes via UART.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

27 August 2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.41.2 Macro Definition Documentation

8.41.2.1 `#define SYS_UART1_BAUDRATE 115000`

Baud rate for UART1

Definition at line 36 of file `uart_HDI.h`.

8.41.2.2 `#define SYS_UART2_BAUDRATE 115000`

Baud rate for UART2

Definition at line 37 of file `uart_HDI.h`.

8.41.2.3 `#define UART1_RX_RF2`

Used port on the microcontroller to read from the UART1

Definition at line 26 of file `uart_HDI.h`.

8.41.2.4 `#define UART1_RX_DIR_TRISF2`

direction of the used port on the microcontroller (reading from the UART1)

Definition at line 31 of file `uart_HDI.h`.

8.41.2.5 `#define UART1_TX_RF3`

Used port on the microcontroller to write on the UART1

Definition at line 27 of file `uart_HDI.h`.

8.41.2.6 `#define UART1_TX_DIR_TRISF3`

direction of the used port on the microcontroller (writing from the UART1)

Definition at line 32 of file `uart_HDI.h`.

8.41.2.7 `#define UART2_RX_RF4`

Used port on the microcontroller to read from the UART2

Definition at line 28 of file `uart_HDI.h`.

8.41.2.8 `#define UART2_RX_DIR_TRISF4`

direction of the used port on the microcontroller (reading from the UART2)

Definition at line 33 of file `uart_HDI.h`.

8.41.2.9 `#define UART2_TX_RF5`

Used port on the microcontroller to write on the UART2

Definition at line 29 of file `uart_HDI.h`.

8.41.2.10 `#define UART2_TX_DIR_TRISF5`

direction of the used port on the microcontroller (writing from the UART2)

Definition at line 34 of file `uart_HDI.h`.

8.41.3 Function Documentation

8.41.3.1 `void Sys_Init_UART1_HDI (void)`

This function initialises UART1. < interrupt priority for the UART1 interrupt

< interrupt priority for the UART1 interrupt

< Baud rate for UART1

Definition at line 42 of file `uart_HDI.c`.

8.41.3.2 `void Sys_Init_UART2_HDI (void)`

This function initialises UART2. < Baud rate for UART2

< interrupt priority for the UART2 interrupt

< interrupt priority for the UART2 interrupt

Definition at line 79 of file uart_HDI.c.

8.41.3.3 void Sys_Read_UART1_ISR () [inline]

This function is executed at occurrence of the UART1 reading interrupt.

Definition at line 213 of file uart_HDI.c.

8.41.3.4 void Sys_Read_UART2_ISR () [inline]

This function is executed at occurrence of the UART2 reading interrupt.

Definition at line 271 of file uart_HDI.c.

8.41.3.5 void Sys_Start_UART1_HDI (void)

This function starts UART2.

Note

When executed this function, bytes can be received or transmitted at any time.

Definition at line 106 of file uart_HDI.c.

8.41.3.6 void Sys_Start_UART2_HDI (void)

This function starts UART2.

Note

When executed this function, bytes can be received or transmitted at any time.

Definition at line 122 of file uart_HDI.c.

8.41.3.7 void Sys_Write_UART1_ISR () [inline]

This function is executed at occurrence of the UART1 writing interrupt.

Definition at line 233 of file uart_HDI.c.

8.41.3.8 void Sys_Write_UART2_ISR () [inline]

This function is executed at occurrence of the UART1 writing interrupt.

Definition at line 291 of file uart_HDI.c.

8.41.4 Variable Documentation

8.41.4.1 uint byte_counter_uart1

Bytes that were written

Definition at line 34 of file uart_HDI.c.

8.41.4.2 uint byte_counter_uart2

Bytes that were written

Definition at line 35 of file uart_HDI.c.

8.41.4.3 pUART_reader read_uart_1

pointer to the functions that processes incoming bytes from UART1

Definition at line 28 of file uart_HDI.c.

8.41.4.4 pUART_reader read_uart_2

pointer to the functions that processes incoming bytes from UART2

Definition at line 29 of file uart_HDI.c.

8.41.4.5 sys_uart_txdata* sys_UART1_TX_data

Linked list of messages that need to be sent via UART1

Definition at line 31 of file uart_HDI.c.

8.41.4.6 sys_uart_txdata* sys_UART2_TX_data

Linked list of messages that need to be sent via UART2

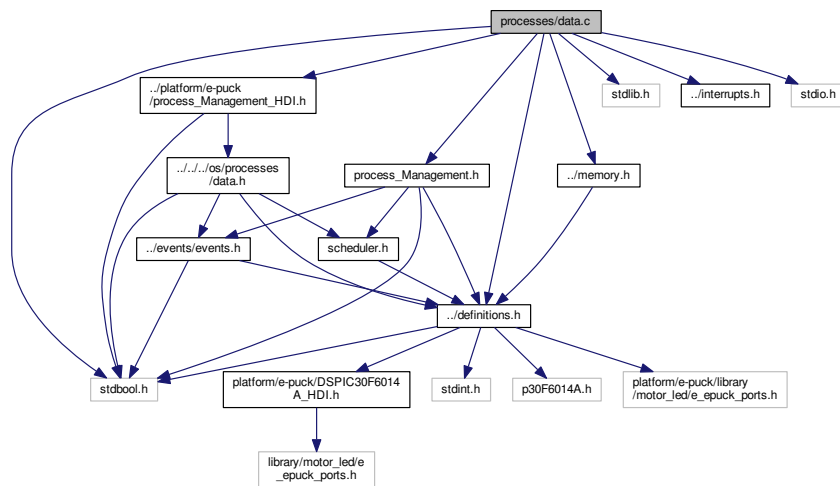
Definition at line 32 of file uart_HDI.c.

8.42 processes/data.c File Reference

It defines functions to manage process lists and related structs.

```
#include "process_Management.h"
#include "../platform/e-puck/process_Management_HDI.h"
#include <stdlib.h>
#include "../interrupts.h"
#include "../memory.h"
#include "../definitions.h"
#include <stdbool.h>
#include <stdio.h>
```

Include dependency graph for data.c:



Functions

- `sys_pcb_list_element * Sys_Remove_Process_from_List (uint pID, sys_pcb_list_element **list)`
- `sys_pcb_list_element * Sys_Find_Process (uint pid)`
- `sys_process_event_handler * Sys_Next_EventHandler (sys_process_event_handler *list, uint eventID)`
- `sys_process_event_handler * Sys_Remove_Event_from_EventRegister (uint eventID, pEventHandler↵ Function func, sys_process_event_handler **list)`

- void `Sys_Clear_EventData` (`sys_event_data **data`)
- void `Sys_Clear_EventRegister` (`sys_pcb_list_element *element`)
- void `Sys_Delete_Process` (`sys_pcb_list_element *element`)
- bool `Sys_Set_Defaults_PCB` (`sys_pcb *element`, `uint stacksize`)
- void `Sys_Insert_Process_to_List` (`sys_pcb_list_element *process`, `sys_pcb_list_element **list`)

Variables

- `sys_pcb_list_element * sys_ready_processes` = 0
- `sys_pcb_list_element * sys_running_process` = 0
- `sys_pcb_list_element * sys_blocked_processes` = 0
- `sys_pcb_list_element * sys_zombies` = 0
- `sys_occurred_event * sys_occurred_events` = 0

8.42.1 Detailed Description

It defines functions to manage process lists and related structs.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

08 July 2014

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.42.2 Function Documentation

8.42.2.1 void `Sys_Clear_EventData` (`sys_event_data ** data`) [inline]

This function removes and frees a list of `sys_event_data`

Parameters

<code>in, out</code>	<code>data</code>	pointer to the event_data (list)
----------------------	-------------------	----------------------------------

Returns

void

Definition at line 214 of file data.c.

8.42.2.2 void `Sys_Clear_EventRegister` (`sys_pcb_list_element * element`) [inline]

This function clears and frees all elements of a process. The process is also unsubscribed from any event, because and empty event register cannot handle any events.

Parameters

<i>in, out</i>	<i>element</i>	pointer to the pcb of the process
----------------	----------------	-----------------------------------

Returns

void

Definition at line 235 of file data.c.

8.42.2.3 void Sys_Delete_Process (sys_pcb_list_element * *element*)

This function deletes container elements. Warning: this function only deletes the process. All the elements which are linked with next are lost in memory, if you haven't take care of that on advance.

Parameters

<i>in</i>	<i>element</i>	pointer to the element which should be deleted
-----------	----------------	--

Returns

void

Definition at line 257 of file data.c.

8.42.2.4 sys_pcb_list_element* Sys_Find_Process (uint *pid*) [inline]

This function return the pointer to the PCB of process with pid

Parameters

<i>in</i>	<i>pid</i>	process ID
-----------	------------	------------

Returns

void

Definition at line 106 of file data.c.

8.42.2.5 void Sys_Insert_Process_to_List (sys_pcb_list_element * *process*, sys_pcb_list_element ** *list*)

This function inserts a process into a process list. Note: The elements are sorted (process ID)

Parameters

<i>in</i>	<i>process</i>	the process struct
<i>in, out</i>	<i>**list</i>	the process list which has to be seached

Returns

void

Definition at line 309 of file data.c.

8.42.2.6 sys_process_event_handler* Sys_Next_EventHandler (sys_process_event_handler * *list*, uint *eventID*) [inline]

This function searches (sequentially) all event handler for an event (eventID). The list contains a list of eventhandler and this function return the first occurrence of eventID. To search the list entirely, use the function on a list and after resulting an element use the same function on the next element (sublist).

Parameters

in	<i>list</i>	list of event handler
in	<i>eventID</i>	The Id of the event which can put the process (PID) back on the ready list

Returns

sys_process_event_handler * pointer to the next event handler for the event (eventID) in list (0 if not found)

Definition at line 142 of file data.c.

8.42.2.7 sys_process_event_handler* Sys_Remove_Event_from_EventRegister (uint eventID, pEventHandlerFunction func, sys_process_event_handler ** list) [inline]

This function removes subscribed handler function from event-handler list

Parameters

in	<i>eventID</i>	Identifier of the event that has to be removed
in	<i>func</i>	pointer to the subscribed handler function
in	<i>list</i>	list of event handlers

Returns

sys_process_event_handler * (New) top of the list (if changed)

< the value to indicate all event handler

< the value to indicate all event handler

Definition at line 170 of file data.c.

8.42.2.8 sys_pcb_list_element* Sys_Remove_Process_from_List (uint pID, sys_pcb_list_element ** list)

This function seaches all elements of a process list and removes the processs pID from it. Note: The element is not deleted. The pointer to it is returned.

Parameters

in	<i>pID</i>	the process identifier
in, out	<i>**list</i>	the process list which has to be seached

Returns

sys_pcb_list_element* the pointer to the removed element

Definition at line 52 of file data.c.

8.42.2.9 bool Sys_Set_Defaults_PCB (sys_pcb * element, uint stacksize) [inline]

This function sets the default values in a sys_process_control_block struct

Parameters

in, out	<i>element</i>	This is a pointer to a sys_process_control_block struct
in	<i>stacksize</i>	This is a uint which represents the size of the stack which should be allocated for this process. The default value (=0) is in DEFAULT_PROCESS_STACK↵_SIZE.

Returns

void

Definition at line 277 of file data.c.

8.42.3 Variable Documentation

8.42.3.1 `sys_pcb_list_element* sys_blocked_processes = 0`

pointer to the blocked process

Definition at line 34 of file data.c.

8.42.3.2 `sys_occurred_event* sys_occurred_events = 0`

pointer to the occurred events

Definition at line 36 of file data.c.

8.42.3.3 `sys_pcb_list_element* sys_ready_processes = 0`

pointer to the ready processes (linked list)

Definition at line 32 of file data.c.

8.42.3.4 `sys_pcb_list_element* sys_running_process = 0`

pointer to the running process

Definition at line 33 of file data.c.

8.42.3.5 `sys_pcb_list_element* sys_zombies = 0`

pointer to the zombie process

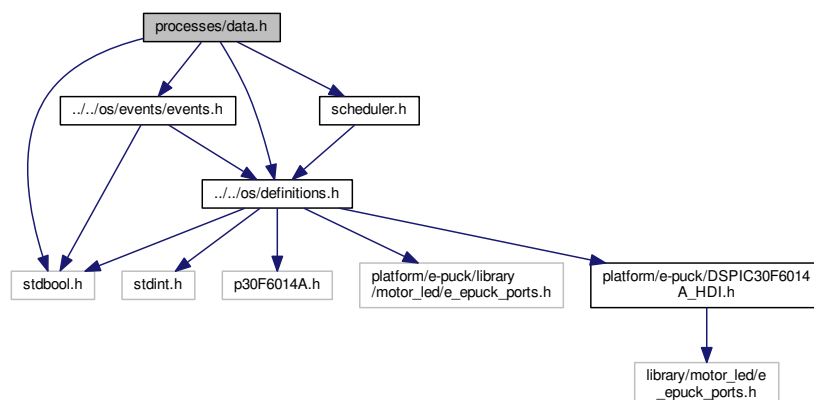
Definition at line 35 of file data.c.

8.43 `processes/data.h` File Reference

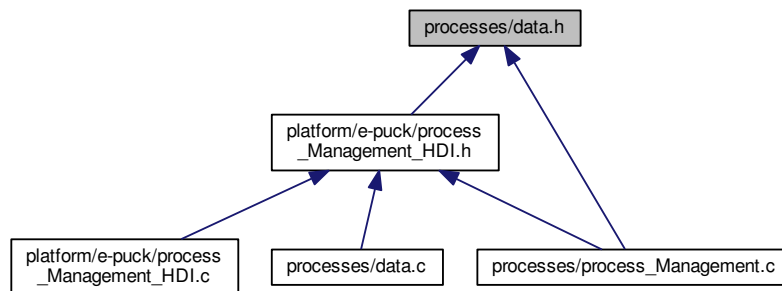
It declares functions to manage process lists and related structs.

```
#include <stdbool.h>
#include "../os/definitions.h"
#include "../os/events/events.h"
#include "scheduler.h"
```

Include dependency graph for data.h:



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



Data Structures

- struct [sys_occurred_event](#)
Linked list element containing an occurred events.
- struct [sys_peh](#)
Double linked list element of process event-handlers.
- struct [sys_pcb](#)
Process Control Block contains all data for a single process.
- struct [sys_pcb_list_element](#)
Double linked list element containing sys_process_control_block.

Functions

- [sys_pcb_list_element *](#) [Sys_Find_Process](#) ([uint16](#) pid)
- [sys_pcb_list_element *](#) [Sys_Remove_Process_from_List](#) ([uint16](#) pID, [sys_pcb_list_element **](#)list)
- void [Sys_Delete_Process](#) ([sys_pcb_list_element *](#)element)
- bool [Sys_Set_Defaults_PCB](#) ([sys_process_control_block *](#)element, [uint16](#) stacksize)
- void [Sys_Insert_Process_to_List](#) ([sys_pcb_list_element *](#)process, [sys_pcb_list_element **](#)list)
- [sys_process_event_handler *](#) [Sys_Next_EventHandler](#) ([sys_process_event_handler *](#)list, [uint16](#) eventID)
- void [Sys_Clear_EventRegister](#) ([sys_pcb_list_element *](#)element)
- void [Sys_Clear_EventData](#) ([sys_event_data **](#)data)
- [sys_process_event_handler *](#) [Sys_Find_EventHandler](#) ([sys_process_event_handler *](#)process, [uint16](#) eventID)
- [sys_process_event_handler *](#) [Sys_Remove_Event_from_EventRegister](#) ([uint16](#) eventID, [pEventHandler](#)↵
[Function](#) func, [sys_process_event_handler **](#)list)

Variables

- [sys_pcb_list_element *](#) [sys_ready_processes](#)
- [sys_pcb_list_element *](#) [sys_running_process](#)
- [sys_pcb_list_element *](#) [sys_blocked_processes](#)
- [sys_pcb_list_element *](#) [sys_zombies](#)
- [sys_occurred_event *](#) [sys_occurred_events](#)

8.43.1 Detailed Description

It declares functions to manage process lists and related structs.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

08 July 2014

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.43.2 Function Documentation

8.43.2.1 void Sys_Clear_EventData (sys_event_data ** data) [inline]

This function removes and frees a list of [sys_event_data](#)

Parameters

<i>in, out</i>	<i>data</i>	pointer to the event_data (list)
----------------	-------------	----------------------------------

Returns

void

Definition at line 214 of file data.c.

8.43.2.2 void Sys_Clear_EventRegister (sys_pcb_list_element * element) [inline]

This function clears and frees all elements of a process. The process is also unsubscribed from any event, because an empty event register cannot handle any events.

Parameters

<i>in, out</i>	<i>element</i>	pointer to the pcb of the process
----------------	----------------	-----------------------------------

Returns

void

Definition at line 235 of file data.c.

8.43.2.3 void Sys_Delete_Process (sys_pcb_list_element * element)

This function deletes container elements. Warning: this function only deletes the process. All the elements which are linked with next are lost in memory, if you haven't take care of that on advance.

Parameters

in	<i>element</i>	pointer to the element which should be deleted
----	----------------	--

Returns

void

Definition at line 257 of file data.c.

8.43.2.4 **sys_process_event_handler*** Sys_Find_EventHandler (**sys_process_event_handler** * *process*, **uint16** *eventID*)
[inline]

8.43.2.5 **sys_pcb_list_element*** Sys_Find_Process (**uint** *pid*) [inline]

This function return the pointer to the PCB of process with pid

Parameters

in	<i>pid</i>	process ID
----	------------	------------

Returns

void

Definition at line 106 of file data.c.

8.43.2.6 **void** Sys_Insert_Process_to_List (**sys_pcb_list_element** * *process*, **sys_pcb_list_element** ** *list*)

This function inserts a process into a process list. Note: The elements are sorted (process ID)

Parameters

in	<i>process</i>	the process struct
in, out	** <i>list</i>	the process list which has to be seached

Returns

void

Definition at line 309 of file data.c.

8.43.2.7 **sys_process_event_handler*** Sys_Next_EventHandler (**sys_process_event_handler** * *list*, **uint** *eventID*)
[inline]

This function searches (sequentially) all event handler for an event (eventID). The list contains a list of eventhandler and this function return the first occurrence of eventID. To search the list entirely, use the function on a list and after resulting an element use the same function on the next element (sublist).

Parameters

in	<i>list</i>	list of event handler
in	<i>eventID</i>	The Id of the event which can put the process (PID) back on the ready list

Returns

sys_process_event_handler * pointer to the next event handler for the event (eventID) in list (0 if not found)

Definition at line 142 of file data.c.

8.43.2.8 **sys_process_event_handler*** Sys_Remove_Event_from_EventRegister (**uint** *eventID*, **pEventHandlerFunction** *func*, **sys_process_event_handler** ** *list*) [inline]

This function removes subscribed handler function from event-handler list

Parameters

in	<i>eventID</i>	Identifier of the event that has to be removed
in	<i>func</i>	pointer to the subscribed handler function
in	<i>list</i>	list of event handlers

Returns

sys_process_event_handler * (New) top of the list (if changed)

< the value to indicate all event handler

< the value to indicate all event handler

Definition at line 170 of file data.c.

8.43.2.9 sys_pcb_list_element* Sys_Remove_Process_from_List (uint *pID*, sys_pcb_list_element ** *list*)

This function searches all elements of a process list and removes the process *pID* from it. Note: The element is not deleted. The pointer to it is returned.

Parameters

in	<i>pID</i>	the process identifier
in, out	** <i>list</i>	the process list which has to be searched

Returns

sys_pcb_list_element* the pointer to the removed element

Definition at line 52 of file data.c.

8.43.2.10 bool Sys_Set_Defaults_PCB (sys_process_control_block * *element*, uint16 *stacksize*) [inline]

8.43.3 Variable Documentation

8.43.3.1 sys_pcb_list_element* sys_blocked_processes

pointer to the blocked process

Definition at line 34 of file data.c.

8.43.3.2 sys_occurred_event* sys_occurred_events

pointer to the occurred events

Definition at line 36 of file data.c.

8.43.3.3 sys_pcb_list_element* sys_ready_processes

pointer to the ready processes (linked list)

Definition at line 32 of file data.c.

8.43.3.4 sys_pcb_list_element* sys_running_process

pointer to the running process

Definition at line 33 of file data.c.

8.43.3.5 sys_pcb_list_element* sys_zombies

pointer to the zombie process

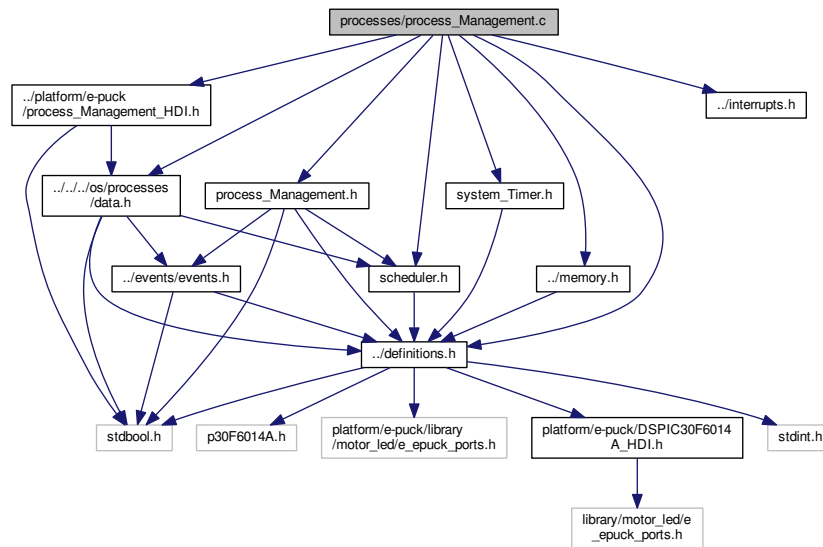
Definition at line 35 of file data.c.

8.44 processes/process_Management.c File Reference

It defines functions to manage processes (e.g. task creation, switching, termination)

```
#include "process_Management.h"
#include "../platform/e-puck/process_Management_HDI.h"
#include "data.h"
#include "scheduler.h"
#include "system_Timer.h"
#include "../interrupts.h"
#include "../memory.h"
#include "../definitions.h"
```

Include dependency graph for process_Management.c:



Functions

- void [Sys_Block_Process](#) (uint pid, uint eventID, pConditionFunction condition)
- bool [Sys_Continue_Pocess](#) (uint pid, uint eventID, sys_event_data *data)
- void [Sys_Set_Running_Process_to_Zombie](#) ()
- void [Sys_Init_Process_Management](#) ()
- unsigned short [Sys_Get_Number_Processes](#) ()
- bool [Sys_Start_Process](#) (pFunction function)
- void [Sys_Kill_Process](#) (uint pid)
- void [Sys_Kill_Zombies](#) ()
- void [Sys_Switch_Process](#) (uint pid)
- void [Sys_Switch_to_next_Process](#) ()
- void [Sys_Start_CriticalSection](#) (void)
- void [Sys_End_CriticalSection](#) (void)
- bool [Sys_Add_Event_Subscription](#) (uint pid, uint eventID, pEventHandlerFunction func, pConditionFunction cond)
- void [Sys_Add_Event_to_Process](#) (uint pid, uint eventID, void *data, uint length)
- void [Sys_Execute_Events_in_ProcessList](#) (uint eventID, sys_pcb_list_element *elements)
- void [Sys_Execute_All_EventHandler](#) ()
- void [Sys_Interprocess_EventHandling](#) ()
- void [Sys_Remove_All_Event_Subscriptions](#) (uint eventID)

- void `Sys_Remove_Event_Subscription` (uint *pid*, uint *eventID*, `pEventHandlerFunction` *func*)
- `sys_event_data` * `Sys_Wait_For_Condition` (uint *eventID*, `pConditionFunction` *function*)
- `sys_event_data` * `Sys_Wait_For_Event` (uint *eventID*)
- void `Sys_Yield` ()

8.44.1 Detailed Description

It defines functions to manage processes (e.g. task creation, switching, termination)

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

{08 July 2014}

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.44.2 Function Documentation

8.44.2.1 `bool Sys_Add_Event_Subscription (uint pid, uint eventID, pEventHandlerFunction func, pConditionFunction cond)`

Puts a process on the blocked list and stops its execution (if it's executed)

Parameters

in	<i>pid</i>	Process ID
in	<i>eventID</i>	The Id of the event which can put the process (PID) back on the ready list
in	<i>func</i>	The function that handles the event
in	<i>cond</i>	The condition under which the handler is executed

Returns

bool Was the event-handler successfully added?

Definition at line 323 of file `process_Management.c`.

8.44.2.2 `void Sys_Add_Event_to_Process (uint pid, uint eventID, void * data, uint length)`

This function adds the event-data to the local list of the process (*pid*).

Parameters

in	<i>pid</i>	process identifier
in	<i>eventID</i>	event identifier

in	<i>data</i>	memory that contains the value of the occurred event
in	<i>length</i>	length of the data (bytes)

Definition at line 376 of file process_Management.c.

8.44.2.3 void Sys_Block_Process (uint *pid*, uint *eventID*, pConditionFunction *condition*)

Puts a process on the blocked list and stops its execution (if it's executed)

Parameters

in	<i>pid</i>	Process ID
in	<i>eventID</i>	The Id of the event which can put the process (PID) back on the ready list
in	<i>condition</i>	the condition under which the process is released

Definition at line 249 of file process_Management.c.

8.44.2.4 bool Sys_Continue_Pocess (uint *pid*, uint *eventID*, sys_event_data * *data*)

Puts a process with the process ID (PID) back on the ready list. Consequently the process can be executed again.

Parameters

in	<i>pid</i>	Process ID
in	<i>eventID</i>	Event ID
in	<i>data</i>	pointer to the data of the event

Definition at line 278 of file process_Management.c.

8.44.2.5 void Sys_End_CriticalSection (void) [inline]

This Function ends a critical section. The task-scheduling can now occure. Note: if a critical section was started once or more, it only takes a single call of this function to end all critical sections.

Definition at line 232 of file process_Management.c.

8.44.2.6 void Sys_Execute_All_EventHandler () [inline]

This function executes all event handlers and processes stored event data. First it checks the list of occurred events and then it executes all event handlers of these events

Definition at line 491 of file process_Management.c.

8.44.2.7 void Sys_Execute_Events_in_ProcessList (uint *eventID*, sys_pcb_list_element * *elements*) [inline]

This function executes all event handlers and processes stored event data. First it checks the list of occurred events and then it executes all event handlers of these events

Parameters

in	<i>eventID</i>	event identifier
in	<i>elements</i>	list of processes

Definition at line 465 of file process_Management.c.

8.44.2.8 unsigned short Sys_Get_Number_Processes (void) [inline]

This function counts the number of process

Definition at line 63 of file process_Management.c.

8.44.2.9 void Sys_Init_Process_Management (void) [inline]

This function initialises the process management and creates the first elements in the linked list

Definition at line 54 of file process_Management.c.

8.44.2.10 void Sys_Interprocess_EventHandling ()

This function starts the execution of the event handler and resets the execution time of the process

Definition at line 514 of file process_Management.c.

8.44.2.11 void Sys_Kill_Process (uint *pid*) [inline]

This function deletes the `sys_process_control_block` element and stops a process

Parameters

in	<i>pid</i>	This argument is the process identifier
----	------------	---

< State to indicate that a process is about to be deleted

< State to indicate that a process is about to be deleted

Definition at line 99 of file process_Management.c.

8.44.2.12 void Sys_Kill_Zombies (void) [inline]

This function deletes all processes which are marked as zombies.

Definition at line 174 of file process_Management.c.

8.44.2.13 void Sys_Remove_All_Event_Subscriptions (uint *eventID*)

This function removes all subscriptions of any process to event (*eventID*)

Parameters

in	<i>eventID</i>	Identifier of the event that has to be removed
----	----------------	--

< the value to indicate all event handler

Definition at line 529 of file process_Management.c.

8.44.2.14 void Sys_Remove_Event_Subscription (uint *pid*, uint *eventID*, pEventHandlerFunction *func*)

This function removes subscribed handler function for process (*pid*) to event (*eventID*)

Parameters

in	<i>pid</i>	Identifier of the process
in	<i>eventID</i>	Identifier of the event that has to be removed
in	<i>func</i>	pointer to the subscribed handler function

Definition at line 546 of file process_Management.c.

8.44.2.15 void Sys_Set_Running_Process_to_Zombie ()

This function puts the running process in the zombie list and switches content to the next ready process < State to indicate that a process is about to be deleted

Definition at line 132 of file process_Management.c.

8.44.2.16 void Sys_Start_CriticalSection (void) [inline]

This Function starts a critical section to prevent the task-scheduling during its execution

Definition at line 223 of file process_Management.c.

8.44.2.17 bool Sys_Start_Process (pFunction *function*) [inline]

This function creates a new `sys_process_control_block` (in a `sys_process_control_block_list_element`) which contains all information which is used to execute this process.

Parameters

<i>in</i>	<i>function</i>	This argument points to a function in memory which should be executed as an new task
-----------	-----------------	--

Definition at line 89 of file process_Management.c.

8.44.2.18 void Sys_Switch_Process (uint *pid*)

This function loads all values into the registers of a process with the PID

Parameters

<i>in</i>	<i>pid</i>	process id
-----------	------------	------------

Definition at line 192 of file process_Management.c.

8.44.2.19 void Sys_Switch_to_next_Process (void)

This function loads all values into the registers of the process which is next in the list.

Definition at line 210 of file process_Management.c.

8.44.2.20 sys_event_data* Sys_Wait_For_Condition (uint *eventID*, pConditionFunction *function*)

This function blocks the current process while waiting for an event that sends data which meet the condition.

Parameters

<i>in</i>	<i>eventID</i>	Identifier of the event that need to occur
<i>in</i>	<i>function</i>	Pointer to the function that represents the condition function (return true if condition is met and continues the process). If function = 0 .. condition is always met.

Returns

[sys_event_data](#) * Pointer to the event data struct that contains the values carried by the event

Definition at line 569 of file process_Management.c.

8.44.2.21 sys_event_data* Sys_Wait_For_Event (uint *eventID*) [inline]

This function blocks the current process and waits for the occurrence of event (eventID).

Parameters

<i>eventID</i>	ID of the event
----------------	-----------------

Returns

[sys_event_data](#) * Pointer to the event data struct that contains the values carried by the event

Definition at line 605 of file process_Management.c.

8.44.2.22 void Sys_Yield (void)

This function blocks the current process and let the next process be executed.

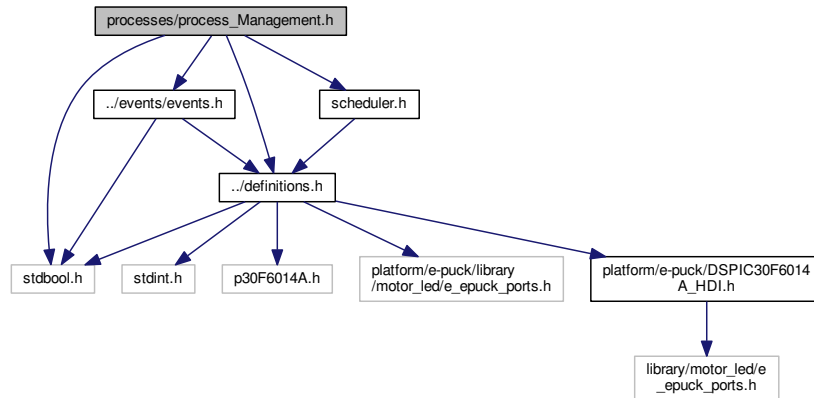
Definition at line 614 of file process_Management.c.

8.45 processes/process_Management.h File Reference

It declares functions to manage processes (e.g. task creation, switching, termination)

```
#include <stdbool.h>
#include "../definitions.h"
#include "../events/events.h"
#include "scheduler.h"
```

Include dependency graph for process_Management.h:



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



Macros

- `#define` `DEFAULT_PROCESS_STACK_SIZE` 200

Functions

- void `Sys_Switch_Process` (uint pid)
- void `Sys_Switch_to_next_Process` (void)
- bool `Sys_Start_Process` (pFunction function)
- void `Sys_Kill_Process` (uint pid)
- void `Sys_Kill_Zombies` (void)
- void `Sys_Yield` (void)
- void `Sys_Init_Process_Management` (void)
- unsigned short `Sys_Get_Number_Processes` (void)
- void `Sys_Start_CriticalSection` (void)
- void `Sys_End_CriticalSection` (void)
- bool `Sys_Add_Event_Subscription` (uint pid, uint eventID, pEventHandlerFunction func, pConditionFunction cond)
- void `Sys_Remove_Event_Subscription` (uint pid, uint eventID, pEventHandlerFunction func)
- void `Sys_Remove_All_Event_Subscriptions` (uint eventID)
- void `Sys_Add_Event_to_Process` (uint pid, uint eventID, void *data, uint length)
- void `Sys_Execute_All_EventHandler` ()
- void `Sys_Clear_EventData` (sys_event_data **data)
- sys_event_data * `Sys_Wait_For_Event` (uint eventID)
- sys_event_data * `Sys_Wait_For_Condition` (uint eventID, pConditionFunction function)

8.45.1 Detailed Description

It declares functions to manage processes (e.g. task creation, switching, termination)

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

08 July 2014

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.45.2 Macro Definition Documentation

8.45.2.1 #define DEFAULT_PROCESS_STACK_SIZE 200

process default stack size

Definition at line 110 of file process_Management.h.

8.45.3 Function Documentation

8.45.3.1 bool Sys_Add_Event_Subscription (uint *pid*, uint *eventID*, pEventHandlerFunction *func*, pConditionFunction *cond*)

Puts a process on the blocked list and stops its execution (if it's executed)

Parameters

in	<i>pid</i>	Process ID
in	<i>eventID</i>	The Id of the event which can put the process (PID) back on the ready list
in	<i>func</i>	The function that handles the event
in	<i>cond</i>	The condition under which the handler is executed

Returns

bool Was the event-handler successfully added?

Definition at line 323 of file process_Management.c.

8.45.3.2 void Sys_Add_Event_to_Process (uint *pid*, uint *eventID*, void * *data*, uint *length*)

This function adds the event-data to the local list of the process (pid).

Parameters

in	<i>pid</i>	process identifier
in	<i>eventID</i>	event identifier
in	<i>data</i>	memory that contains the value of the occurred event
in	<i>length</i>	length of the data (bytes)

Definition at line 376 of file process_Management.c.

8.45.3.3 `void Sys_Clear_EventData (sys_event_data ** data) [inline]`

This function removes and frees a list of [sys_event_data](#)

Parameters

in, out	<i>data</i>	pointer to the event_data (list)
---------	-------------	----------------------------------

Returns

void

Definition at line 214 of file data.c.

8.45.3.4 `void Sys_End_CriticalSection (void) [inline]`

This Function ends a critical section. The task-scheduling can now occur. Note: if a critical section was started once or more, it only takes a single call of this function to end all critical sections.

Definition at line 232 of file process_Management.c.

8.45.3.5 `void Sys_Execute_All_EventHandler () [inline]`

This function executes all event handlers and processes stored event data. First it checks the list of occurred events and then it executes all event handlers of these events

Definition at line 491 of file process_Management.c.

8.45.3.6 `unsigned short Sys_Get_Number_Processes (void) [inline]`

This function counts the number of process

Definition at line 63 of file process_Management.c.

8.45.3.7 `void Sys_Init_Process_Management (void) [inline]`

This function initialises the process management and creates the first elements in the linked list

Definition at line 54 of file process_Management.c.

8.45.3.8 `void Sys_Kill_Process (uint pid) [inline]`

This function deletes the `sys_process_control_block` element and stops a process

Parameters

in	<i>pid</i>	This argument is the process identifier
----	------------	---

< State to indicate that a process is about to be deleted

< State to indicate that a process is about to be deleted

Definition at line 99 of file process_Management.c.

8.45.3.9 `void Sys_Kill_Zombies (void) [inline]`

This function deletes all processes which are marked as zombies.

Definition at line 174 of file process_Management.c.

8.45.3.10 void Sys_Remove_All_Event_Subscriptions (uint *eventID*)

This function removes all subscriptions of any process to event (eventID)

Parameters

in	<i>eventID</i>	Identifier of the event that has to be removed
----	----------------	--

< the value to indicate all event handler

Definition at line 529 of file process_Management.c.

8.45.3.11 void Sys_Remove_Event_Subscription (uint *pid*, uint *eventID*, pEventHandlerFunction *func*)

This function removes subscribed handler function for process (pid) to event (eventID)

Parameters

in	<i>pid</i>	Identifier of the process
in	<i>eventID</i>	Identifier of the event that has to be removed
in	<i>func</i>	pointer to the subscribed handler function

Definition at line 546 of file process_Management.c.

8.45.3.12 void Sys_Start_CriticalSection (void) [inline]

This Function starts a critical section to prevent the task-scheduling during its execution

Definition at line 223 of file process_Management.c.

8.45.3.13 bool Sys_Start_Process (pFunction *function*) [inline]

This function creates a new sys_process_control_block (in a sys_process_control_block_list_element) which contains all information which is used to execute this process.

Parameters

in	<i>function</i>	This argument points to a function in memory which should be executed as a new task
----	-----------------	---

Definition at line 89 of file process_Management.c.

8.45.3.14 void Sys_Switch_Process (uint *pid*)

This function loads all values into the registers of a process with the PID

Parameters

in	<i>pid</i>	process id
----	------------	------------

Definition at line 192 of file process_Management.c.

8.45.3.15 void Sys_Switch_to_next_Process (void)

This function loads all values into the registers of the process which is next in the list.

Definition at line 210 of file process_Management.c.

8.45.3.16 sys_event_data* Sys_Wait_For_Condition (uint *eventID*, pConditionFunction *function*)

This function blocks the current process while waiting for an event that sends data which meet the condition.

Parameters

in	<i>eventID</i>	Identifier of the event that need to occur
in	<i>function</i>	Pointer to the function that represents the condition function (return true if condition is met and continues the process). If function = 0 .. condition is always met.

Returns

`sys_event_data` * Pointer to the event data struct that contains the values carried by the event

Definition at line 569 of file process_Management.c.

8.45.3.17 `sys_event_data* Sys_Wait_For_Event (uint eventID) [inline]`

This function blocks the current process and waits for the occurrence of event (eventID).

Parameters

<i>eventID</i>	ID of the event
----------------	-----------------

Returns

`sys_event_data`* Pointer to the event data struct that contains the values carried by the event

Definition at line 605 of file process_Management.c.

8.45.3.18 `void Sys_Yield (void)`

This function blocks the current process and let the next process be executed.

Definition at line 614 of file process_Management.c.

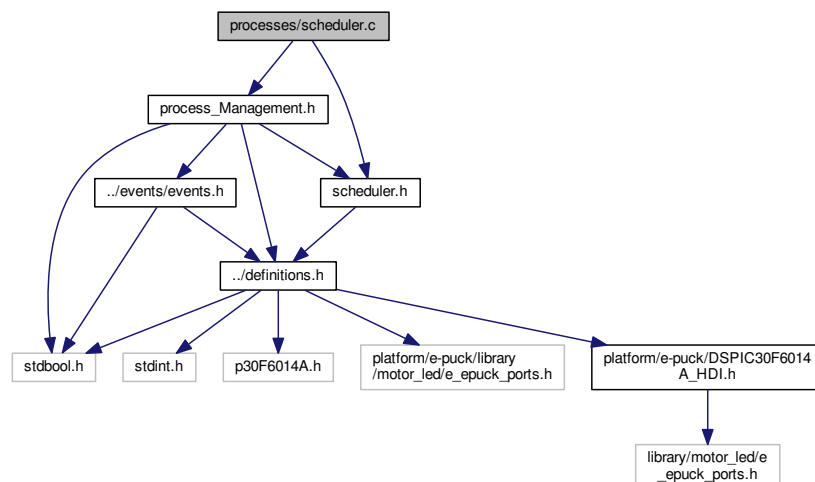
8.46 processes/scheduler.c File Reference

It defines functions to specify a scheduling algorithm.

```
#include "scheduler.h"
```

```
#include "process_Management.h"
```

Include dependency graph for scheduler.c:



Functions

- void [Sys_Scheduler_RoundRobin](#) (void)
- void [Sys_Set_Defaults_Info](#) ([sys_scheduler_info](#) *sct)

8.46.1 Detailed Description

It defines functions to specify a scheduling algorithm.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

{07 July 2014}

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.46.2 Function Documentation

8.46.2.1 void Sys_Scheduler_RoundRobin (void)

This function shows the implementation of the RoundRobin Scheduling algorithm

Definition at line 24 of file scheduler.c.

8.46.2.2 void Sys_Set_Defaults_Info ([sys_scheduler_info](#) * sct) [inline]

This function sets the default values in a [sys_scheduler_info](#) struct

Parameters

in, out	sct	This is a pointer to a sys_scheduler_info struct
-------------------------	---------------------	--

< State to indicate that a process is waiting to be executed

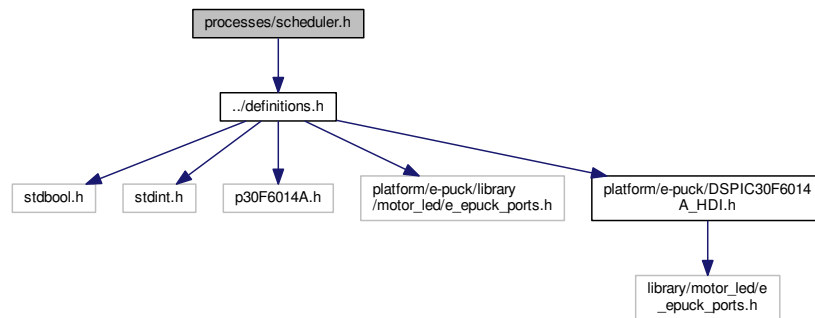
Definition at line 45 of file scheduler.c.

8.47 processes/scheduler.h File Reference

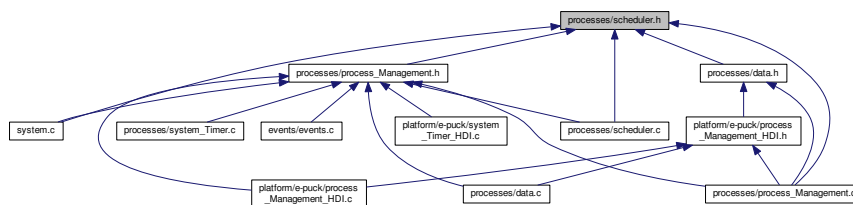
It declares functions to specify a scheduling algorithm.

```
#include "../definitions.h"
```

Include dependency graph for scheduler.h:



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



Data Structures

- struct [sys_scheduler_info](#)
The scheduling information for each process.

Macros

- #define [SYS_PROCESS_STATE_BABY](#) 0xBABE
- #define [SYS_PROCESS_STATE_RUNNING](#) 0xFFFF
- #define [SYS_PROCESS_STATE_BLOCKED](#) 0xBCED
- #define [SYS_PROCESS_STATE_WAITING](#) 0x5555
- #define [SYS_PROCESS_STATE_ZOMBIE](#) 0xDEAD
- #define [SYS_PROCESS_PRIORITY_SYSTEM](#) 0xFFFF
- #define [SYS_PROCESS_PRIORITY_HIGH](#) 0x0FFF
- #define [SYS_PROCESS_PRIORITY_NORMAL](#) 0x00FF
- #define [SYS_PROCESS_PRIORITY_LOW](#) 0x000F

Functions

- void [Sys_Scheduler_RoundRobin](#) (void)
- void [Sys_Set_Defaults_Info](#) (sys_scheduler_info *Sct)

8.47.1 Detailed Description

It declares functions to specify a scheduling algorithm.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

{07 July 2014}

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.47.2 Macro Definition Documentation

8.47.2.1 #define SYS_PROCESS_PRIORITY_HIGH 0x0FFF

Definition at line 31 of file scheduler.h.

8.47.2.2 #define SYS_PROCESS_PRIORITY_LOW 0x000F

Definition at line 33 of file scheduler.h.

8.47.2.3 #define SYS_PROCESS_PRIORITY_NORMAL 0x00FF

Definition at line 32 of file scheduler.h.

8.47.2.4 #define SYS_PROCESS_PRIORITY_SYSTEM 0xFFFF

process priority values Process priority: System = highest

Definition at line 30 of file scheduler.h.

8.47.2.5 #define SYS_PROCESS_STATE_BABY 0xBABE

process state values State to indicate that a process is created but not yet ready to be executed

Definition at line 23 of file scheduler.h.

8.47.2.6 #define SYS_PROCESS_STATE_BLOCKED 0xBCED

State to indicate that a process is blocked and waits till an event occurs

Definition at line 25 of file scheduler.h.

8.47.2.7 #define SYS_PROCESS_STATE_RUNNING 0xFFFF

State to indicate that a process is executed

Definition at line 24 of file scheduler.h.

8.47.2.8 #define SYS_PROCESS_STATE_WAITING 0x5555

State to indicate that a process is waiting to be executed

Definition at line 26 of file scheduler.h.

8.47.2.9 #define SYS_PROCESS_STATE_ZOMBIE 0xDEAD

State to indicate that a process is about to be deleted

Definition at line 27 of file scheduler.h.

8.47.3 Function Documentation

8.47.3.1 void Sys_Scheduler_RoundRobin (void)

This function shows the implementation of the RoundRobin Scheduling algorithm

Definition at line 24 of file scheduler.c.

8.47.3.2 void Sys_Set_Defaults_Info (sys_scheduler_info * sct) [inline]

This function sets the default values in a [sys_scheduler_info](#) struct

Parameters

in, out	sct	This is a pointer to a sys_scheduler_info struct
---------	-----	--

< State to indicate that a process is waiting to be executed

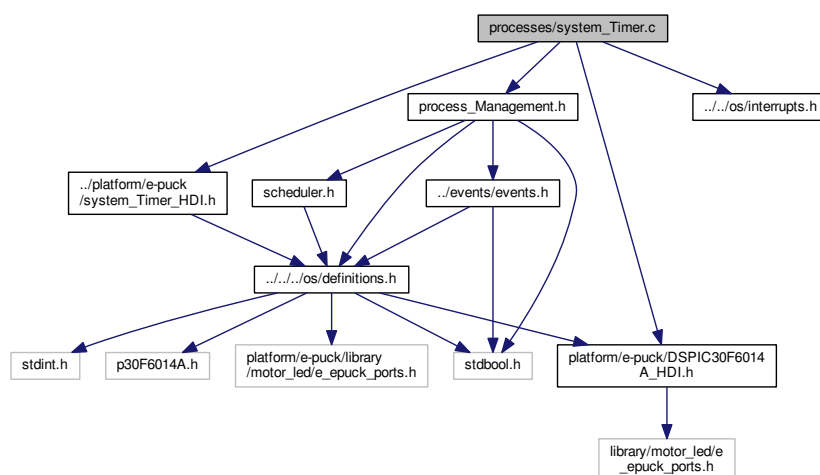
Definition at line 45 of file scheduler.c.

8.48 processes/system_Timer.c File Reference

It defines functions to initialise, configure and run the system Timer.

```
#include "../platform/e-puck/system_Timer_HDI.h"
#include "process_Management.h"
#include "../platform/e-puck/DSPIC30F6014A_HDI.h"
#include "../../os/interrupts.h"
```

Include dependency graph for system_Timer.c:



Functions

- void [Sys_todo_SystemTimer](#) ()
- void [Sys_Init_SystemTimer](#) (pFunction scheduler)

- void [Sys_Start_SystemTimer](#) ()
- void [Sys_Stop_SystemTimer](#) ()
- void [Sys_Continue_SystemTimer](#) ()
- void [Sys_Reset_SystemTimer](#) ()
- void [Sys_Disable_TimerInterrupt](#) (void)
- void [Sys_Enable_TimerInterrupt](#) (void)
- void [Sys_Force_TimerInterrupt](#) (void)

8.48.1 Detailed Description

It defines functions to initialise, configure and run the system Timer.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

{07 July 2014}

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.48.2 Function Documentation

8.48.2.1 void [Sys_Continue_SystemTimer](#) () [inline]

This Function deactivated the Timer1 Interrupt

Definition at line 55 of file system_Timer.c.

8.48.2.2 void [Sys_Disable_TimerInterrupt](#) (void) [inline]

Disables the Timer1 interrupt and sets the interrupt flag to 0

Definition at line 94 of file system_Timer.c.

8.48.2.3 void [Sys_Enable_TimerInterrupt](#) (void) [inline]

Enables the Timer1 interrupt and leaves the interrupt flag to its value. If the flag was set, the Timer1 interrupt will be emitted after executing this function.

Definition at line 103 of file system_Timer.c.

8.48.2.4 void [Sys_Force_TimerInterrupt](#) (void) [inline]

Enables the Timer1 interrupt and leaves the interrupt flag to its value. If the flag was set, the Timer1 interrupt will be emitted after executing this function.

Definition at line 112 of file system_Timer.c.

8.48.2.5 void [Sys_Init_SystemTimer](#) (pFunction *scheduler*) [inline]

This Function sets the Timer0 of the DSPIC 30F6014A for timer intervals of 10 ms. The timer will be startet with [Start_SystemTimer_HDI\(\)](#)

Parameters

<code>in, out</code>	<code>scheduler</code>	This is a pointer to a callback function, which should be called whenever a timer interrupt is emitted.
----------------------	------------------------	---

Definition at line 27 of file `system_Timer.c`.

8.48.2.6 `void Sys_Reset_SystemTimer () [inline]`

This Function resets the Timer1 value

Definition at line 64 of file `system_Timer.c`.

8.48.2.7 `void Sys_Start_SystemTimer (void) [inline]`

This Function starts the Timer0 of the DSPIC 30F6014A for timer intervals of 10 ms. The MUST be initialised first with `Init_SystemTimer_HDI()`

Definition at line 37 of file `system_Timer.c`.

8.48.2.8 `void Sys_Stop_SystemTimer () [inline]`

This Function activated the Timer1 Interrupt

Definition at line 46 of file `system_Timer.c`.

8.48.2.9 `void Sys_todo_SystemTimer () [inline]`

This function is executed periodically by the system timer interrupt. It kills all zombies, executes event handlers and executes the scheduling algorithm.

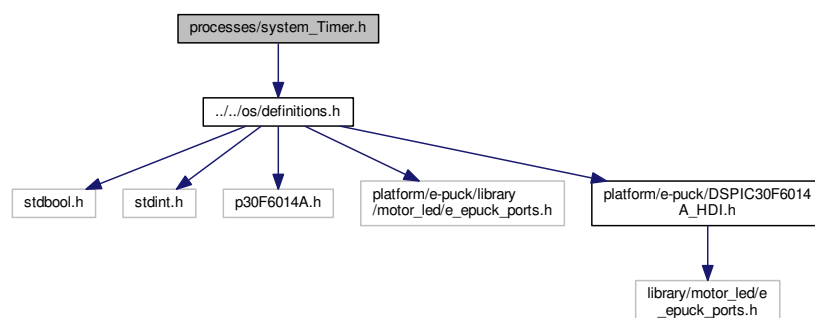
Definition at line 73 of file `system_Timer.c`.

8.49 `processes/system_Timer.h` File Reference

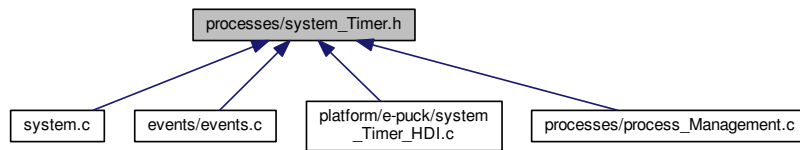
It declares functions to initialise, configure and run the system Timer.

```
#include "../os/definitions.h"
```

Include dependency graph for `system_Timer.h`:



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



Functions

- void [Sys_Init_SystemTimer](#) (pFunction)
- void [Sys_Start_SystemTimer](#) (void)
- void [Sys_Stop_SystemTimer](#) ()
- void [Sys_Continue_SystemTimer](#) ()
- void [Sys_Disable_TimerInterrupt](#) (void)
- void [Sys_Enable_TimerInterrupt](#) (void)
- void [Sys_Force_TimerInterrupt](#) (void)
- void [Sys_Reset_SystemTimer](#) ()
- void [Sys_todo_SystemTimer](#) ()

8.49.1 Detailed Description

It declares functions to initialise, configure and run the system Timer.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

{07 July 2014}

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

8.49.2 Function Documentation

8.49.2.1 void Sys_Continue_SystemTimer () [inline]

This Function deactivated the Timer1 Interrupt

Definition at line 55 of file system_Timer.c.

8.49.2.2 void Sys_Disable_TimerInterrupt (void) [inline]

Disables the Timer1 interrupt and sets the interrupt flag to 0

Definition at line 94 of file system_Timer.c.

8.49.2.3 void Sys_Enable_TimerInterrupt(void) [inline]

Enables the Timer1 interrupt and leaves the interrupt flag to its value. If the flag was set, the Timer1 interrupt will be emitted after executing this function.

Definition at line 103 of file system_Timer.c.

8.49.2.4 void Sys_Force_TimerInterrupt(void) [inline]

Enables the Timer1 interrupt and leaves the interrupt flag to its value. If the flag was set, the Timer1 interrupt will be emitted after executing this function.

Definition at line 112 of file system_Timer.c.

8.49.2.5 void Sys_Init_SystemTimer(pFunction scheduler) [inline]

This Function sets the Timer0 of the DSPIC 30F6014A for timer intervals of 10 ms. The timer will be startet with Start_SystemTimer_HDI()

Parameters

in, out	<i>scheduler</i>	This is a pointer t an callback function, which schuld becalled whenever a timer interrupt is emmitted.
---------	------------------	---

Definition at line 27 of file system_Timer.c.

8.49.2.6 void Sys_Reset_SystemTimer() [inline]

This Function resets the Timer1 value

Definition at line 64 of file system_Timer.c.

8.49.2.7 void Sys_Start_SystemTimer(void) [inline]

This Function starts the Timer0 of the DSPIC 30F6014A for timer intervals of 10 ms. The MUST be initialised first with Init_SystemTimer_HDI()

Definition at line 37 of file system_Timer.c.

8.49.2.8 void Sys_Stop_SystemTimer() [inline]

This Function activated the Timer1 Interrupt

Definition at line 46 of file system_Timer.c.

8.49.2.9 void Sys_todo_SystemTimer() [inline]

This function is executed periodically by the system timer interrupt. It kills all zombies, executes event handlers and executes the scheduling algorithm.

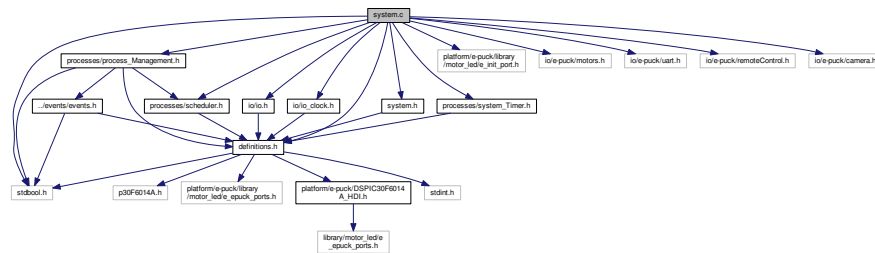
Definition at line 73 of file system_Timer.c.

8.50 system.c File Reference

defines functions to initialise and start OpenSwarm.

```
#include "definitions.h"
#include "system.h"
#include "processes/system_Timer.h"
#include "processes/scheduler.h"
#include "processes/process_Management.h"
#include "platform/e-puck/library/motor_led/e_init_port.h"
#include "io/io.h"
#include "io/io_clock.h"
#include "io/e-puck/motors.h"
#include "io/e-puck/uart.h"
#include "io/e-puck/remoteControl.h"
#include "io/e-puck/camera.h"
```

Include dependency graph for system.c:



Functions

- void [Sys_Init_Kernel](#) ()
- void [Sys_Start_Kernel](#) (void)

8.50.1 Detailed Description

defines functions to initialise and start OpenSwarm.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

2015

Copyright

adapted FreeBSD License (see <http://openswarm.org/license>)

In short, Openswarm can be executed as shown in the following example

```
#include "os/system.h"

int main(void) {
    //initialise some global or local variables
```

```

Sys_Init_Kernel();

//do some preperation before executing OpenSwarm and user applications

Sys_Start_Kernel();
while(1){
    //do nothing
}
}

```

8.50.2 Function Documentation

8.50.2.1 void Sys_Init_Kernel (void)

This Function sets the system Timer (Timer0) and sets an scheduling algorithm. It also intitalises I/O devices (e.g. if e-puck is used: motor, UART, remote control, and camera)

Postcondition

To start OpenSwarm, [Sys_Start_Kernel\(\)](#) must be executed after the initialisation.

Remarks

Code can be executed between initialisation and start of the kernel. But, note that you can only execute code that does not depend on an active OpenSwarm.

< ID of the event that signal a general termination event

Definition at line 63 of file system.c.

8.50.2.2 void Sys_Start_Kernel (void)

This Function starts all functions of the operating system. The system MUST HAVE BEEN INITIALISED before.

Precondition

System must be initialised with [Sys_Init_Kernel\(\)](#).

Remarks

Code can be executed between initialisation and start of the kernel. But, note that you can only execute code that does not depend on an active OpenSwarm.

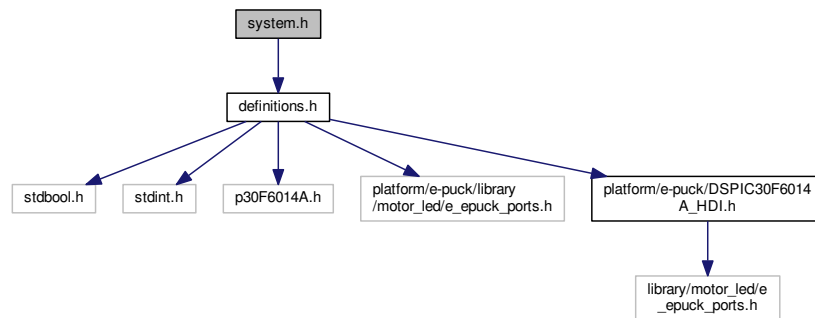
Definition at line 102 of file system.c.

8.51 system.h File Reference

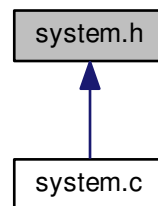
It declares functions to initialise and start OpenSwarm.

```
#include "definitions.h"
```

Include dependency graph for system.h:



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



Macros

- `#define SYS_MOTOR_USED`
- `#define SYS_UART1_USED`
- `#define SYS_REMOTECONTROL_USED`
- `#define SYS_CAMERA_USED`

Functions

- `void Sys_Init_Kernel (void)`
- `void Sys_Start_Kernel (void)`

8.51.1 Detailed Description

It declares functions to initialise and start OpenSwarm.

Author

Stefan M. Trenkwalder s.trenkwalder@openswarm.org

Version

1.0

Date

{07 July 2014}

Copyrightadapted FreeBSD License (see <http://openswarm.org/license>)**8.51.2 Macro Definition Documentation****8.51.2.1 #define SYS_CAMERA_USED**

Define this preprocessor symbol to use the camera

Definition at line 85 of file system.h.

8.51.2.2 #define SYS_MOTOR_USED

Define this preprocessor symbol to use motors

Definition at line 82 of file system.h.

8.51.2.3 #define SYS_REMOTECONTROL_USED

Define this preprocessor symbol to receive remote control signals

Definition at line 84 of file system.h.

8.51.2.4 #define SYS_UART1_USED

Define this preprocessor symbol to use UART1

Definition at line 83 of file system.h.

8.51.3 Function Documentation**8.51.3.1 void Sys_Init_Kernel (void)**

This Function sets the system Timer (Timer0) and sets an scheduling algorithm. It also intitalises I/O devices (e.g. if e-puck is used: motor, UART, remote control, and camera)

PostconditionTo start OpenSwarm, [Sys_Start_Kernel\(\)](#) mast be executed after the initialisation.**Remarks**

Code can be executed between initialisation and start of the kernel. But, note that you can only execute code that does not depend on an active OpenSwarm.

< ID of the event that signal a general termination event

Definition at line 63 of file system.c.

8.51.3.2 void Sys_Start_Kernel (void)

This Function starts all functions of the operating system. The system MUST HAVE BEEN INITIALISED before.

Precondition

System must be initialised with [Sys_Init_Kernel\(\)](#).

Remarks

Code can be executed between initialisation and start of the kernel. But, note that you can only execute code that does not depend on an active OpenSwarm.

Definition at line 102 of file system.c.

Index

- [__attribute__](#)
 - [io_HDI.c, 94](#)
 - [remoteControl_HDI.c, 123](#)
 - [system_Timer_HDI.c, 128](#)
 - [traps.c, 132](#)
 - [uart_HDI.c, 140](#)
- [ADDRESS_AITV_ADDRESS_ERROR](#)
 - [DSPIC30F6014A_HDI.h, 77](#)
- [ADDRESS_AITV_MATH_ERROR](#)
 - [DSPIC30F6014A_HDI.h, 77](#)
- [ADDRESS_AITV_OSC_FAIL](#)
 - [DSPIC30F6014A_HDI.h, 77](#)
- [ADDRESS_AITV_STACK_ERROR](#)
 - [DSPIC30F6014A_HDI.h, 77](#)
- [ADDRESS_AIVT](#)
 - [DSPIC30F6014A_HDI.h, 77](#)
- [ADDRESS_AIVT_T1](#)
 - [DSPIC30F6014A_HDI.h, 77](#)
- [ADDRESS_ITV_ADDRESS_ERROR](#)
 - [DSPIC30F6014A_HDI.h, 77](#)
- [ADDRESS_ITV_MATH_ERROR](#)
 - [DSPIC30F6014A_HDI.h, 77](#)
- [ADDRESS_ITV_OSC_FAIL](#)
 - [DSPIC30F6014A_HDI.h, 77](#)
- [ADDRESS_ITV_STACK_ERROR](#)
 - [DSPIC30F6014A_HDI.h, 78](#)
- [ADDRESS_IVT](#)
 - [DSPIC30F6014A_HDI.h, 78](#)
- [ADDRESS_IVT_T1](#)
 - [DSPIC30F6014A_HDI.h, 78](#)
- [ALL_FUNCTIONS](#)
 - [definitions.h, 38](#)
- [BLACK](#)
 - [definitions.h, 41](#)
- [BLUE](#)
 - [definitions.h, 41](#)
- [BLUE_MAX](#)
 - [camera.c, 65](#)
- [BLUE_THRESHOLD](#)
 - [camera.c, 65](#)
- [blue](#)
 - [sys_rgb_pixel, 34](#)
- [brushedColorFromRGB565](#)
 - [camera_processing.c, 74](#)
 - [camera_processing.h, 75](#)
- [buffered_data](#)
 - [sys_peh, 31](#)
- [byte_counter_uart1](#)
 - [uart_HDI.c, 141](#)
 - [uart_HDI.h, 145](#)
- [byte_counter_uart2](#)
 - [uart_HDI.c, 141](#)
 - [uart_HDI.h, 145](#)
- [CAM_H_SIZE](#)
 - [camera.c, 65](#)
- [CAM_HEIGHT](#)
 - [camera.c, 65](#)
- [CAM_W_SIZE](#)
 - [camera.c, 65](#)
- [CAM_WIDTH](#)
 - [camera.c, 66](#)
- [CAM_ZOOM_X](#)
 - [camera.c, 66](#)
- [CAM_ZOOM_Y](#)
 - [camera.c, 66](#)
- [CAMERA_I2C_ADDRESS](#)
 - [camera.c, 66](#)
- [CBP_BI](#)
 - [camera_processing.c, 73](#)
- [CBP_DI](#)
 - [camera_processing.c, 73](#)
- [CBP_GI](#)
 - [camera_processing.c, 73](#)
- [CBP_RI](#)
 - [camera_processing.c, 73](#)
- [CBP_WI](#)
 - [camera_processing.c, 73](#)
- [COLOUR_THRESHOLD](#)
 - [camera.c, 66](#)
- [CP_BI](#)
 - [camera.c, 66](#)
 - [camera_processing.c, 73](#)
- [CP_GI](#)
 - [camera.c, 66](#)
 - [camera_processing.c, 73](#)
- [CP_RI](#)
 - [camera.c, 66](#)
 - [camera_processing.c, 73](#)
- [CP_WGB_I](#)
 - [camera_processing.c, 73](#)
- [CP_WI](#)
 - [camera.c, 66](#)
 - [camera_processing.c, 73](#)
- [CYAN](#)
 - [definitions.h, 41](#)
- [Camera Module, 12](#)
- [camera.c](#)
 - [BLUE_MAX, 65](#)
 - [BLUE_THRESHOLD, 65](#)
 - [CAM_H_SIZE, 65](#)
 - [CAM_HEIGHT, 65](#)
 - [CAM_W_SIZE, 65](#)
 - [CAM_WIDTH, 66](#)
 - [CAM_ZOOM_X, 66](#)
 - [CAM_ZOOM_Y, 66](#)
 - [CAMERA_I2C_ADDRESS, 66](#)
 - [COLOUR_THRESHOLD, 66](#)
 - [CP_BI, 66](#)

- CP_GI, [66](#)
- CP_RI, [66](#)
- CP_WI, [66](#)
- FRAME_HEIGHT, [66](#)
- FRAME_WIDTH, [66](#)
- GREEN_MAX, [67](#)
- GREEN_THRESHOLD, [67](#)
- RED_MAX, [67](#)
- RED_THRESHOLD, [67](#)
- Sys_Camera_PreProcessor, [67](#)
- Sys_Init_Camera, [67](#)
- Sys_Process_newFrame, [67](#)
- Sys_Process_newLine, [67](#)
- Sys_Process_newPixel, [68](#)
- Sys_Set_Preprocessing, [68](#)
- Sys_Start_Camera, [68](#)
- camera.h
 - getFinishedFrame, [70](#)
 - isNewFrameAvailable, [70](#)
 - pCameraPreProcessor, [70](#)
 - SYS_MAX_BLUE, [70](#)
 - SYS_MAX_GREEN, [70](#)
 - SYS_MAX_RED, [70](#)
 - Sys_Init_Camera, [70](#)
 - Sys_Set_Preprocessing, [70](#)
 - Sys_Start_Camera, [72](#)
- camera_processing.c
 - brushedColorFromRGB565, [74](#)
 - CBP_BI, [73](#)
 - CBP_DI, [73](#)
 - CBP_GI, [73](#)
 - CBP_RI, [73](#)
 - CBP_WI, [73](#)
 - CP_BI, [73](#)
 - CP_GI, [73](#)
 - CP_RI, [73](#)
 - CP_WGB_I, [73](#)
 - CP_WI, [73](#)
 - colorBrushedPositions, [74](#)
 - colorPositions, [74](#)
 - convertRGB565ToRGB888, [74](#)
 - getBrushedColorAt, [74](#)
 - getRGB565at, [74](#)
 - getRGB888at, [74](#)
 - nearestNeighborRGB, [74](#)
 - powerTbl, [74](#)
- camera_processing.h
 - brushedColorFromRGB565, [75](#)
 - convertRGB565ToRGB888, [75](#)
 - getBrushedColorAt, [75](#)
 - getRGB565at, [76](#)
 - getRGB888at, [76](#)
 - nearestNeighborRGB, [76](#)
- colorBrushedPositions
 - camera_processing.c, [74](#)
- colorPositions
 - camera_processing.c, [74](#)
- condition
 - sys_peh, [31](#)
- convertRGB565ToRGB888
 - camera_processing.c, [74](#)
 - camera_processing.h, [75](#)
- DEFAULT_PROCESS_STACK_SIZE
 - process_Management.h, [163](#)
- DSPIC30F6014A_HDI.h
 - ADDRESS_AITV_ADDRESS_ERROR, [77](#)
 - ADDRESS_AITV_MATH_ERROR, [77](#)
 - ADDRESS_AITV_OSC_FAIL, [77](#)
 - ADDRESS_AITV_STACK_ERROR, [77](#)
 - ADDRESS_AIVT, [77](#)
 - ADDRESS_AIVT_T1, [77](#)
 - ADDRESS_ITV_ADDRESS_ERROR, [77](#)
 - ADDRESS_ITV_MATH_ERROR, [77](#)
 - ADDRESS_ITV_OSC_FAIL, [77](#)
 - ADDRESS_ITV_STACK_ERROR, [78](#)
 - ADDRESS_IVT, [78](#)
 - ADDRESS_IVT_T1, [78](#)
- data
 - sys_i2c_msg, [27](#)
 - sys_uart_txdata, [36](#)
- data.c
 - Sys_Clear_EventData, [147](#)
 - Sys_Clear_EventRegister, [147](#)
 - Sys_Delete_Process, [148](#)
 - Sys_Find_Process, [148](#)
 - Sys_Insert_Process_to_List, [148](#)
 - Sys_Next_EventHandler, [148](#)
 - Sys_Remove_Event_from_EventRegister, [149](#)
 - Sys_Remove_Process_from_List, [149](#)
 - Sys_Set_Defaults_PCB, [149](#)
 - sys_blocked_processes, [150](#)
 - sys_occurred_events, [150](#)
 - sys_ready_processes, [150](#)
 - sys_running_process, [150](#)
 - sys_zombies, [150](#)
- data.h
 - Sys_Clear_EventData, [152](#)
 - Sys_Clear_EventRegister, [152](#)
 - Sys_Delete_Process, [152](#)
 - Sys_Find_EventHandler, [153](#)
 - Sys_Find_Process, [153](#)
 - Sys_Insert_Process_to_List, [153](#)
 - Sys_Next_EventHandler, [153](#)
 - Sys_Remove_Event_from_EventRegister, [153](#)
 - Sys_Remove_Process_from_List, [154](#)
 - Sys_Set_Defaults_PCB, [154](#)
 - sys_blocked_processes, [154](#)
 - sys_occurred_events, [154](#)
 - sys_ready_processes, [154](#)
 - sys_running_process, [154](#)
 - sys_zombies, [154](#)
- definitions.h, [36](#)
 - ALL_FUNCTIONS, [38](#)
 - BLACK, [41](#)
 - BLUE, [41](#)
 - CYAN, [41](#)

- EPUCK_USED, 38
- GREEN, 41
- MAGENTA, 41
- pByteFunction, 39
- pFunction, 39
- pUART_reader, 40
- RED, 41
- SYS_EVENT_1ms_CLOCK, 38
- SYS_EVENT_IO_CAMERA, 38
- SYS_EVENT_IO_MOTOR_LEFT, 38
- SYS_EVENT_IO_MOTOR_RIGHT, 38
- SYS_EVENT_IO_REMOECONTROL, 39
- SYS_EVENT_IO_TO_BLUETOOTH, 39
- SYS_EVENT_TERMINATION, 39
- sint, 40
- sint16, 40
- sint32, 40
- sint8, 40
- sys_colour, 40
- UART1_RX, 39
- UART1_RX_DIR, 39
- UART1_TX, 39
- UART1_TX_DIR, 39
- UART2_RX, 39
- UART2_RX_DIR, 39
- UART2_TX, 39
- UART2_TX_DIR, 39
- uint, 40
- uint16, 40
- uint32, 40
- uint8, 40
- WHITE, 41
- YELLOW, 41
- e-puck specific modules, 15
- EPUCK_USED
 - definitions.h, 38
- Event Management, 8
- event_register
 - sys_pcb, 29
- eventID
 - sys_occurred_event, 28
 - sys_peh, 31
 - sys_registered_event, 33
- events.c
 - registered_events, 44
 - Sys_Find_Event, 42
 - Sys_IsEventRegistered, 42
 - Sys_Register_Event, 43
 - Sys_Send_Event, 43
 - Sys_Send_IntEvent, 43
 - Sys_Subscribe_to_Event, 43
 - Sys_Unregister_Event, 44
 - Sys_Unsubscribe_Handler_from_Event, 44
 - Sys_Unsubscribe_Process, 44
 - Sys_Unsubscribe_from_Event, 44
- events.h
 - pConditionFunction, 46
 - pEventHandlerFunction, 46
 - Sys_IsEventRegistered, 46
 - Sys_Register_Event, 46
 - Sys_Send_Event, 47
 - Sys_Send_IntEvent, 47
 - Sys_Subscribe_to_Event, 47
 - Sys_Unregister_Event, 47
 - Sys_Unsubscribe_Process, 48
 - Sys_Unsubscribe_from_Event, 48
- events/events.c, 41
- events/events.h, 45
- FRAME_HEIGHT
 - camera.c, 66
- FRAME_WIDTH
 - camera.c, 66
- framePointer
 - sys_pcb, 29
- function
 - sys_pIOHandler, 32
- GREEN
 - definitions.h, 41
- GREEN_MAX
 - camera.c, 67
- GREEN_THRESHOLD
 - camera.c, 67
- getBrushedColorAt
 - camera_processing.c, 74
 - camera_processing.h, 75
- getFinishedFrame
 - camera.h, 70
- getRGB565at
 - camera_processing.c, 74
 - camera_processing.h, 76
- getRGB888at
 - camera_processing.c, 74
 - camera_processing.h, 76
- green
 - sys_rgb_pixel, 34
- handler
 - sys_i2c_msg, 27
 - sys_peh, 32
- I/O Management, 10
- I2C interface, 18
- I2C_ACKNOWLEDGED
 - i2c_data.h, 87
- I2C_ERROR
 - i2c_data.h, 87
- I2C_ERROR_MODE
 - i2c_data.h, 86
- I2C_IDLE
 - i2c_data.h, 87
- I2C_IDLE_MODE
 - i2c_data.h, 86
- I2C_IS_READING
 - i2c_data.h, 87
- I2C_IS_SENDING

- i2c_data.h, 87
- I2C_IS_STARTING
 - i2c_data.h, 87
- I2C_IS_STOPPING
 - i2c_data.h, 87
- I2C_READING_BYTES_MODE
 - i2c_data.h, 86
- I2C_SENT
 - i2c_data.h, 87
- I2C_STARTED
 - i2c_data.h, 87
- I2C_WRITING_ADDRESS_MODE
 - i2c_data.h, 86
- I2C_WRITING_BYTES_MODE
 - i2c_data.h, 86
- i2c.c
 - Sys_Contine_I2C, 79
 - Sys_I2C_Read, 79
 - Sys_I2C_ReadByte, 79
 - Sys_I2C_Send_ACK, 79
 - Sys_I2C_Send_NACK, 79
 - Sys_I2C_Send_Restart, 80
 - Sys_I2C_Send_Start, 80
 - Sys_I2C_Send_Stop, 80
 - Sys_I2C_SentBytes, 80
 - Sys_I2C_Start_Reading, 80
 - Sys_I2C_WriteByte, 80
 - Sys_Init_I2C, 80
 - Sys_Pause_I2C, 80
 - Sys_Start_I2C, 80
 - Sys_Stop_I2C, 81
- i2c.h
 - Sys_Contine_I2C, 82
 - Sys_I2C_Read, 82
 - Sys_I2C_SentBytes, 82
 - Sys_Init_I2C, 83
 - Sys_Pause_I2C, 83
 - Sys_Start_I2C, 83
 - Sys_Stop_I2C, 83
- i2c_HDI.c
 - Sys_Contine_I2C_HDI, 89
 - Sys_I2C_ReadByte_HDI, 89
 - Sys_I2C_Send_ACK_HDI, 89
 - Sys_I2C_Send_NACK_HDI, 89
 - Sys_I2C_Send_Restart_HDI, 89
 - Sys_I2C_Send_Start_HDI, 89
 - Sys_I2C_Send_Stop_HDI, 89
 - Sys_I2C_Start_Reading_HDI, 89
 - Sys_I2C_WriteByte_HDI, 89
 - Sys_Init_I2C_HDI, 89
 - Sys_Pause_I2C_HDI, 90
 - Sys_Start_I2C_HDI, 90
 - Sys_Stop_I2C_HDI, 90
- i2c_HDI.h
 - Sys_Contine_I2C_HDI, 92
 - Sys_I2C_ReadByte_HDI, 92
 - Sys_I2C_Send_ACK_HDI, 92
 - Sys_I2C_Send_NACK_HDI, 92
- Sys_I2C_Send_Restart_HDI, 92
- Sys_I2C_Send_Start_HDI, 92
- Sys_I2C_Send_Stop_HDI, 92
- Sys_I2C_Start_Reading_HDI, 92
- Sys_I2C_WriteByte_HDI, 92
- Sys_Init_I2C_HDI, 92
- Sys_Pause_I2C_HDI, 92
- Sys_Start_I2C_HDI, 93
- Sys_Stop_I2C_HDI, 93
- i2c_data.c
 - Sys_I2C_AppendMessages, 84
 - Sys_I2C_FreeMessages, 84
 - Sys_I2C_RemoveOldestMessage, 84
 - sys_i2c_msgs, 85
- i2c_data.h
 - I2C_ACKNOWLEDGED, 87
 - I2C_ERROR, 87
 - I2C_ERROR_MODE, 86
 - I2C_IDLE, 87
 - I2C_IDLE_MODE, 86
 - I2C_IS_READING, 87
 - I2C_IS_SENDING, 87
 - I2C_IS_STARTING, 87
 - I2C_IS_STOPPING, 87
 - I2C_READING_BYTES_MODE, 86
 - I2C_SENT, 87
 - I2C_STARTED, 87
 - I2C_WRITING_ADDRESS_MODE, 86
 - I2C_WRITING_BYTES_MODE, 86
 - Sys_I2C_AppendMessages, 87
 - Sys_I2C_FreeMessages, 87
 - Sys_I2C_RemoveOldestMessage, 87
 - sys_i2c_mode, 86
 - sys_i2c_state, 86
 - sys_i2c_msgs, 87
- i2c_device_address
 - sys_i2c_msg, 27
- interrupts.c, 48
 - Sys_End_AtomicSection, 49
 - Sys_Start_AtomicSection, 49
- interrupts.h, 50
 - SYS_IRQP_CAMERA_FRAME, 50
 - SYS_IRQP_CAMERA_LINE, 50
 - SYS_IRQP_CAMERA_PIXEL, 51
 - SYS_IRQP_I2C, 51
 - SYS_IRQP_IO_TIMER, 51
 - SYS_IRQP_MAX, 51
 - SYS_IRQP_REMOTECONTROL, 51
 - SYS_IRQP_SYSTEM_TIMER, 51
 - SYS_IRQP_UART1, 51
 - SYS_IRQP_UART2, 51
 - Sys_End_AtomicSection, 51
 - Sys_Start_AtomicSection, 51
- io.c
 - Sys_Continue_IOTimer, 53
 - Sys_Disable_IOTimerInterrupt, 53
 - Sys_Enable_IOTimerInterrupt, 53
 - Sys_Force_IOTimerInterrupt, 53

- Sys_Init_IOManagement, [53](#)
- Sys_Register_IOHandler, [53](#)
- Sys_Reset_IOTimer, [54](#)
- Sys_Start_IOManagement, [54](#)
- Sys_Stop_IOManagement, [54](#)
- Sys_Stop_IOTimer, [54](#)
- Sys_Unregister_IOHandler, [54](#)

io.h

- Sys_Continue_IOTimer, [56](#)
- Sys_Disable_IOTimerInterrupt, [56](#)
- Sys_Enable_IOTimerInterrupt, [56](#)
- Sys_Force_IOTimerInterrupt, [56](#)
- Sys_Init_IOManagement, [56](#)
- Sys_Register_IOHandler, [56](#)
- Sys_Reset_IOTimer, [56](#)
- Sys_Start_IOManagement, [56](#)
- Sys_Stop_IOManagement, [56](#)
- Sys_Stop_IOTimer, [57](#)
- Sys_Unregister_IOHandler, [57](#)

io/io.c, [52](#)

io/io.h, [54](#)

io/io_clock.c, [57](#)

io/io_clock.h, [59](#)

io_HDI.c

- __attribute__, [94](#)
- Sys_Continue_IOTimer_HDI, [95](#)
- Sys_Disable_IOTimerInterrupt_HDI, [95](#)
- Sys_Enable_IOTimerInterrupt_HDI, [95](#)
- Sys_Force_IOTimerInterrupt_HDI, [95](#)
- Sys_IOTimer_code_HDI, [95](#)
- Sys_Init_IOTimer_HDI, [95](#)
- Sys_Reset_IOTimer_HDI, [95](#)
- Sys_Start_IOTimer_HDI, [95](#)
- Sys_Stop_IOTimer_HDI, [95](#)
- sys_iohandlers, [96](#)

io_HDI.h

- STEPS_PER_MILLISECOND, [97](#)
- STEPS_PER_SECOND, [97](#)
- Sys_Continue_IOTimer_HDI, [98](#)
- Sys_Disable_IOTimerInterrupt_HDI, [98](#)
- Sys_Enable_IOTimerInterrupt_HDI, [98](#)
- Sys_Force_IOTimerInterrupt_HDI, [98](#)
- Sys_IOTimer_code_HDI, [98](#)
- Sys_Init_IOTimer_HDI, [98](#)
- Sys_Reset_IOTimer_HDI, [98](#)
- Sys_Start_IOTimer_HDI, [98](#)
- Sys_Stop_IOTimer_HDI, [98](#)
- sys_iohandlers, [98](#)

io_clock.c

- Sys_Get_SystemClock, [58](#)
- Sys_Get_SystemTime, [58](#)
- Sys_Init_Clock, [58](#)
- Sys_Init_SystemTime, [58](#)
- Sys_SystemClock_Counter, [58](#)

io_clock.h

- Sys_Get_SystemClock, [60](#)
- Sys_Get_SystemTime, [60](#)
- Sys_Init_Clock, [60](#)

isNewDataAvailable

- remoteControl_HDI.c, [123](#)
- remoteControl_HDI.h, [126](#)

isNewFrameAvailable

- camera.h, [70](#)

Kernel, [6](#)

length

- sys_i2c_msg, [27](#)
- sys_uart_txdata, [36](#)

MAGENTA

- definitions.h, [41](#)

MAX_WHEEL_SPEED

- motors.c, [100](#)

MAX_WHEEL_SPEED_MM_S

- motors.h, [103](#)

MOTORPHASE_RESET

- motors_HDI.h, [107](#)

memory.c, [60](#)

- Sys_Free, [61](#)
- Sys_Malloc, [62](#)
- Sys_Memcpy, [62](#)

memory.h, [62](#)

- Sys_Free, [63](#)
- Sys_Malloc, [63](#)
- Sys_Memcpy, [63](#)

message_arriving

- remoteControl_HDI.c, [123](#)
- remoteControl_HDI.h, [126](#)

Motor Control, [20](#)

motors.c

- MAX_WHEEL_SPEED, [100](#)
- POWER_SAVE_WAIT, [100](#)
- Sys_Get_LeftWheelSpeed, [100](#)
- Sys_Get_RightWheelSpeed, [100](#)
- Sys_Init_Motors, [100](#)
- Sys_LeftMotor_Controller, [101](#)
- Sys_LeftMotor_EventHandler, [101](#)
- Sys_LeftMotor_Reset, [101](#)
- Sys_RightMotor_Controller, [101](#)
- Sys_RightMotor_EventHandler, [101](#)
- Sys_RightMotor_Reset, [101](#)
- Sys_Set_LeftWheelSpeed, [102](#)
- Sys_Set_RightWheelSpeed, [102](#)

motors.h

- MAX_WHEEL_SPEED_MM_S, [103](#)
- Sys_Get_LeftWheelSpeed, [104](#)
- Sys_Get_RightWheelSpeed, [104](#)
- Sys_Init_Motors, [104](#)
- Sys_Set_LeftWheelSpeed, [104](#)
- Sys_Set_RightWheelSpeed, [104](#)

motors_HDI.c

- Sys_LeftMotor_SetPhase_HDI, [105](#)
- Sys_RightMotor_SetPhase_HDI, [106](#)

motors_HDI.h

- MOTORPHASE_RESET, [107](#)

- Sys_LeftMotor_SetPhase_HDI, 107
- Sys_RightMotor_SetPhase_HDI, 107
- nearestNeighborRGB
 - camera_processing.c, 74
 - camera_processing.h, 76
- next
 - sys_event_data, 26
 - sys_i2c_msg, 27
 - sys_occurred_event, 28
 - sys_pIOHandler, 32
 - sys_pcb_list_element, 30
 - sys_peh, 32
 - sys_registered_event, 33
 - sys_subscribed_process, 35
 - sys_uart_txdata, 36
- pByteFunction
 - definitions.h, 39
- pCameraPreProcessor
 - camera.h, 70
- pConditionFunction
 - events.h, 46
- pEventHandlerFunction
 - events.h, 46
- pFunction
 - definitions.h, 39
- POWER_SAVE_WAIT
 - motors.c, 100
- pUART_reader
 - definitions.h, 40
- pcb
 - sys_pcb_list_element, 30
- pid
 - sys_subscribed_process, 35
- platform/e-puck/DSPIC30F6014A_HDI.h, 76
- platform/e-puck/camera.c, 64
- platform/e-puck/camera.h, 68
- platform/e-puck/camera_processing.c, 72
- platform/e-puck/camera_processing.h, 75
- platform/e-puck/i2c.c, 78
- platform/e-puck/i2c.h, 81
- platform/e-puck/i2c_HDI.c, 87
- platform/e-puck/i2c_HDI.h, 90
- platform/e-puck/i2c_data.c, 83
- platform/e-puck/i2c_data.h, 85
- platform/e-puck/io_HDI.c, 93
- platform/e-puck/io_HDI.h, 96
- platform/e-puck/motors.c, 99
- platform/e-puck/motors.h, 102
- platform/e-puck/motors_HDI.c, 105
- platform/e-puck/motors_HDI.h, 106
- platform/e-puck/process_Management_HDI.c, 108
- platform/e-puck/process_Management_HDI.h, 110
- platform/e-puck/remoteControl.c, 112
- platform/e-puck/remoteControl.h, 114
- platform/e-puck/remoteControl_HDI.c, 121
- platform/e-puck/remoteControl_HDI.h, 124
- platform/e-puck/system_Timer_HDI.c, 126
- platform/e-puck/system_Timer_HDI.h, 129
- platform/e-puck/traps.c, 131
- platform/e-puck/uart.c, 132
- platform/e-puck/uart.h, 135
- platform/e-puck/uart_HDI.c, 138
- platform/e-puck/uart_HDI.h, 142
- powerTbl
 - camera_processing.c, 74
- previous
 - sys_pcb_list_element, 30
 - sys_peh, 32
- priority
 - sys_scheduler_info, 35
- Process Management, 23
- process_ID
 - sys_pcb, 29
- process_Management.c
 - Sys_Add_Event_Subscription, 156
 - Sys_Add_Event_to_Process, 156
 - Sys_Block_Process, 156
 - Sys_Continue_Pocess, 157
 - Sys_End_CriticalSection, 157
 - Sys_Execute_All_EventHandler, 157
 - Sys_Execute_Events_in_ProcessList, 157
 - Sys_Get_Number_Processes, 157
 - Sys_Init_Process_Management, 157
 - Sys_Interprocess_EventHandling, 157
 - Sys_Kill_Process, 157
 - Sys_Kill_Zombies, 159
 - Sys_Remove_All_Event_Subscriptions, 159
 - Sys_Remove_Event_Subscription, 159
 - Sys_Set_Running_Process_to_Zombie, 159
 - Sys_Start_CriticalSection, 159
 - Sys_Start_Process, 159
 - Sys_Switch_Process, 159
 - Sys_Switch_to_next_Process, 161
 - Sys_Wait_For_Condition, 161
 - Sys_Wait_For_Event, 161
 - Sys_Yield, 161
- process_Management.h
 - DEFAULT_PROCESS_STACK_SIZE, 163
 - Sys_Add_Event_Subscription, 163
 - Sys_Add_Event_to_Process, 163
 - Sys_Clear_EventData, 163
 - Sys_End_CriticalSection, 165
 - Sys_Execute_All_EventHandler, 165
 - Sys_Get_Number_Processes, 165
 - Sys_Init_Process_Management, 165
 - Sys_Kill_Process, 165
 - Sys_Kill_Zombies, 165
 - Sys_Remove_All_Event_Subscriptions, 165
 - Sys_Remove_Event_Subscription, 165
 - Sys_Start_CriticalSection, 167
 - Sys_Start_Process, 167
 - Sys_Switch_Process, 167
 - Sys_Switch_to_next_Process, 167
 - Sys_Wait_For_Condition, 167
 - Sys_Wait_For_Event, 167

- Sys_Yield, 168
- process_Management_HDI.c
 - Sys_Change_Stack_HDI, 109
 - Sys_Init_Process_Management_HDI, 109
 - Sys_Save_Running_Process_HDI, 109
 - Sys_Start_Process_HDI, 109
 - Sys_Switch_Process_HDI, 109
- process_Management_HDI.h
 - Sys_Change_Stack_HDI, 111
 - Sys_Init_Process_Management_HDI, 111
 - Sys_Save_Running_Process_HDI, 111
 - Sys_Start_Process_HDI, 111
 - Sys_Switch_Process_HDI, 111
- process_stack
 - sys_pcb, 29
- processes/data.c, 146
- processes/data.h, 150
- processes/process_Management.c, 155
- processes/process_Management.h, 161
- processes/scheduler.c, 168
- processes/scheduler.h, 169
- processes/system_Timer.c, 172
- processes/system_Timer.h, 174
- RC_BUTTON_0
 - remoteControl.h, 116
- RC_BUTTON_1
 - remoteControl.h, 116
- RC_BUTTON_2
 - remoteControl.h, 116
- RC_BUTTON_3
 - remoteControl.h, 117
- RC_BUTTON_4
 - remoteControl.h, 117
- RC_BUTTON_5
 - remoteControl.h, 117
- RC_BUTTON_6
 - remoteControl.h, 117
- RC_BUTTON_7
 - remoteControl.h, 117
- RC_BUTTON_8
 - remoteControl.h, 117
- RC_BUTTON_9
 - remoteControl.h, 117
- RC_BUTTON_BACK
 - remoteControl.h, 117
- RC_BUTTON_BLUE
 - remoteControl.h, 117
- RC_BUTTON_CHANNEL_DOWN
 - remoteControl.h, 117
- RC_BUTTON_CHANNEL_UP
 - remoteControl.h, 117
- RC_BUTTON_CURSOR_DOWN
 - remoteControl.h, 118
- RC_BUTTON_CURSOR_LEFT
 - remoteControl.h, 118
- RC_BUTTON_CURSOR_RIGHT
 - remoteControl.h, 118
- RC_BUTTON_CURSOR_UP
 - remoteControl.h, 118
- RC_BUTTON_EPG
 - remoteControl.h, 118
- RC_BUTTON_FAV
 - remoteControl.h, 118
- RC_BUTTON_GREEN
 - remoteControl.h, 118
- RC_BUTTON_INFO
 - remoteControl.h, 118
- RC_BUTTON_INTERNET
 - remoteControl.h, 118
- RC_BUTTON_LANG
 - remoteControl.h, 118
- RC_BUTTON_MENU
 - remoteControl.h, 118
- RC_BUTTON_MUTE
 - remoteControl.h, 119
- RC_BUTTON_OK
 - remoteControl.h, 119
- RC_BUTTON_PAUSE
 - remoteControl.h, 119
- RC_BUTTON_PLAY
 - remoteControl.h, 119
- RC_BUTTON_PRESETS
 - remoteControl.h, 119
- RC_BUTTON_RECORD
 - remoteControl.h, 119
- RC_BUTTON_RED
 - remoteControl.h, 119
- RC_BUTTON_REWIND
 - remoteControl.h, 119
- RC_BUTTON_SCREEN
 - remoteControl.h, 119
- RC_BUTTON_SLEEP
 - remoteControl.h, 119
- RC_BUTTON_SOURCE
 - remoteControl.h, 119
- RC_BUTTON_STANDBY
 - remoteControl.h, 120
- RC_BUTTON_STOP
 - remoteControl.h, 120
- RC_BUTTON_SUBTTL
 - remoteControl.h, 120
- RC_BUTTON_SWAP
 - remoteControl.h, 120
- RC_BUTTON_TELE_TEXT
 - remoteControl.h, 120
- RC_BUTTON_VOLUME_DOWN
 - remoteControl.h, 120
- RC_BUTTON_VOLUME_UP
 - remoteControl.h, 120
- RC_BUTTON_WIND
 - remoteControl.h, 120
- RC_BUTTON_YELLOW
 - remoteControl.h, 120
- RC_NOT_STARTED
 - remoteControl_HDI.h, 125
- RC_WAIT_FOR_BIT

- remoteControl_HDI.h, 125
- RC_WAIT_FOR_HALFBIT
 - remoteControl_HDI.h, 125
- RC_WAIT_FOR_QUARTERBIT
 - remoteControl_HDI.h, 125
- RC_WAIT_INITIALLY
 - remoteControl_HDI.h, 125
- RED
 - definitions.h, 41
- RED_MAX
 - camera.c, 67
- RED_THRESHOLD
 - camera.c, 67
- read_uart_1
 - uart_HDI.c, 141
 - uart_HDI.h, 145
- read_uart_2
 - uart_HDI.c, 141
 - uart_HDI.h, 145
- receiving_bit
 - remoteControl_HDI.c, 123
 - remoteControl_HDI.h, 126
- red
 - sys_rgb_pixel, 34
- registered_events
 - events.c, 44
- Remote Control, 21
- remoteControl.c
 - Sys_HasRemoteC_Sent_New_Data, 113
 - Sys_Init_RemoteControl, 113
 - Sys_Receive_RemoteControl_Msg, 113
 - Sys_RemoteC_Get_Address, 113
 - Sys_RemoteC_Get_CheckBit, 113
 - Sys_RemoteC_Get_Data, 114
 - Sys_Start_RemoteControl, 114
- remoteControl.h
 - RC_BUTTON_0, 116
 - RC_BUTTON_1, 116
 - RC_BUTTON_2, 116
 - RC_BUTTON_3, 117
 - RC_BUTTON_4, 117
 - RC_BUTTON_5, 117
 - RC_BUTTON_6, 117
 - RC_BUTTON_7, 117
 - RC_BUTTON_8, 117
 - RC_BUTTON_9, 117
 - RC_BUTTON_BACK, 117
 - RC_BUTTON_BLUE, 117
 - RC_BUTTON_CHANNEL_DOWN, 117
 - RC_BUTTON_CHANNEL_UP, 117
 - RC_BUTTON_CURSOR_DOWN, 118
 - RC_BUTTON_CURSOR_LEFT, 118
 - RC_BUTTON_CURSOR_RIGHT, 118
 - RC_BUTTON_CURSOR_UP, 118
 - RC_BUTTON_EPG, 118
 - RC_BUTTON_FAV, 118
 - RC_BUTTON_GREEN, 118
 - RC_BUTTON_INFO, 118
 - RC_BUTTON_INTERNET, 118
 - RC_BUTTON_LANG, 118
 - RC_BUTTON_MENU, 118
 - RC_BUTTON_MUTE, 119
 - RC_BUTTON_OK, 119
 - RC_BUTTON_PAUSE, 119
 - RC_BUTTON_PLAY, 119
 - RC_BUTTON_PRESETS, 119
 - RC_BUTTON_RECORD, 119
 - RC_BUTTON_RED, 119
 - RC_BUTTON_REWIND, 119
 - RC_BUTTON_SCREEN, 119
 - RC_BUTTON_SLEEP, 119
 - RC_BUTTON_SOURCE, 119
 - RC_BUTTON_STANDBY, 120
 - RC_BUTTON_STOP, 120
 - RC_BUTTON_SUBTTL, 120
 - RC_BUTTON_SWAP, 120
 - RC_BUTTON_TELE_TEXT, 120
 - RC_BUTTON_VOLUME_DOWN, 120
 - RC_BUTTON_VOLUME_UP, 120
 - RC_BUTTON_WIND, 120
 - RC_BUTTON_YELLOW, 120
 - Sys_Init_RemoteControl, 120
 - Sys_Receive_RemoteControl_Msg, 121
 - Sys_RemoteC_Get_Address, 121
 - Sys_RemoteC_Get_CheckBit, 121
 - Sys_RemoteC_Get_Data, 121
 - Sys_RemoteC_Received_New_Data, 121
 - Sys_Start_RemoteControl, 121
- remoteControl_HDI.c
 - __attribute__, 123
 - isNewDataAvailable, 123
 - message_arriving, 123
 - receiving_bit, 123
 - rx_buffer, 123
 - Sys_Init_RemoteControl_HDI, 123
 - Sys_Start_RemoteControl_HDI, 123
 - waiting_cycles, 123
- remoteControl_HDI.h
 - isNewDataAvailable, 126
 - message_arriving, 126
 - RC_NOT_STARTED, 125
 - RC_WAIT_FOR_BIT, 125
 - RC_WAIT_FOR_HALFBIT, 125
 - RC_WAIT_FOR_QUARTERBIT, 125
 - RC_WAIT_INITIALLY, 125
 - receiving_bit, 126
 - rx_buffer, 126
 - Sys_Init_RemoteControl_HDI, 126
 - Sys_Start_RemoteControl_HDI, 126
 - waiting_cycles, 126
- rx_buffer
 - remoteControl_HDI.c, 123
 - remoteControl_HDI.h, 126
- STEPS_PER_MILLISECOND
 - io_HDI.h, 97
- STEPS_PER_SECOND

- io_HDL.h, [97](#)
- SYS_CAMERA_USED
 - system.h, [179](#)
- SYS_EVENT_1ms_CLOCK
 - definitions.h, [38](#)
- SYS_EVENT_IO_CAMERA
 - definitions.h, [38](#)
- SYS_EVENT_IO_MOTOR_LEFT
 - definitions.h, [38](#)
- SYS_EVENT_IO_MOTOR_RIGHT
 - definitions.h, [38](#)
- SYS_EVENT_IO_REMOECONTROL
 - definitions.h, [39](#)
- SYS_EVENT_IO_TO_BLUETOOTH
 - definitions.h, [39](#)
- SYS_EVENT_TERMINATION
 - definitions.h, [39](#)
- SYS_IRQP_CAMERA_FRAME
 - interrupts.h, [50](#)
- SYS_IRQP_CAMERA_LINE
 - interrupts.h, [50](#)
- SYS_IRQP_CAMERA_PIXEL
 - interrupts.h, [51](#)
- SYS_IRQP_I2C
 - interrupts.h, [51](#)
- SYS_IRQP_IO_TIMER
 - interrupts.h, [51](#)
- SYS_IRQP_MAX
 - interrupts.h, [51](#)
- SYS_IRQP_REMOTECONTROL
 - interrupts.h, [51](#)
- SYS_IRQP_SYSTEM_TIMER
 - interrupts.h, [51](#)
- SYS_IRQP_UART1
 - interrupts.h, [51](#)
- SYS_IRQP_UART2
 - interrupts.h, [51](#)
- SYS_MAX_BLUE
 - camera.h, [70](#)
- SYS_MAX_GREEN
 - camera.h, [70](#)
- SYS_MAX_RED
 - camera.h, [70](#)
- SYS_MOTOR_USED
 - system.h, [179](#)
- SYS_PROCESS_PRIORITY_HIGH
 - scheduler.h, [171](#)
- SYS_PROCESS_PRIORITY_LOW
 - scheduler.h, [171](#)
- SYS_PROCESS_PRIORITY_NORMAL
 - scheduler.h, [171](#)
- SYS_PROCESS_PRIORITY_SYSTEM
 - scheduler.h, [171](#)
- SYS_PROCESS_STATE_BABY
 - scheduler.h, [171](#)
- SYS_PROCESS_STATE_BLOCKED
 - scheduler.h, [171](#)
- SYS_PROCESS_STATE_RUNNING
 - scheduler.h, [171](#)
- SYS_PROCESS_STATE_WAITING
 - scheduler.h, [171](#)
- SYS_PROCESS_STATE_ZOMBIE
 - scheduler.h, [171](#)
- SYS_REMOTECONTROL_USED
 - system.h, [179](#)
- SYS_UART1_BAUDRATE
 - uart.c, [134](#)
 - uart_HDL.h, [143](#)
- SYS_UART1_USED
 - system.h, [179](#)
- SYS_UART2_BAUDRATE
 - uart.c, [134](#)
 - uart_HDL.h, [143](#)
- scheduler.c
 - Sys_Scheduler_RoundRobin, [169](#)
 - Sys_Set_Defaults_Info, [169](#)
- scheduler.h
 - SYS_PROCESS_PRIORITY_HIGH, [171](#)
 - SYS_PROCESS_PRIORITY_LOW, [171](#)
 - SYS_PROCESS_PRIORITY_NORMAL, [171](#)
 - SYS_PROCESS_PRIORITY_SYSTEM, [171](#)
 - SYS_PROCESS_STATE_BABY, [171](#)
 - SYS_PROCESS_STATE_BLOCKED, [171](#)
 - SYS_PROCESS_STATE_RUNNING, [171](#)
 - SYS_PROCESS_STATE_WAITING, [171](#)
 - SYS_PROCESS_STATE_ZOMBIE, [171](#)
 - Sys_Scheduler_RoundRobin, [171](#)
 - Sys_Set_Defaults_Info, [171](#)
- sheduler_info
 - sys_pcb, [29](#)
- Shefpuck, [14](#)
- sint
 - definitions.h, [40](#)
- sint16
 - definitions.h, [40](#)
- sint32
 - definitions.h, [40](#)
- sint8
 - definitions.h, [40](#)
- size
 - sys_event_data, [26](#)
- speed
 - sys_motors, [28](#)
- stackPointer
 - sys_pcb, [30](#)
- stackPointerLimit
 - sys_pcb, [30](#)
- state
 - sys_scheduler_info, [35](#)
- subscribers
 - sys_registered_event, [33](#)
- Sys_Add_Event_Subscription
 - process_Management.c, [156](#)
 - process_Management.h, [163](#)
- Sys_Add_Event_to_Process
 - process_Management.c, [156](#)

- process_Management.h, 163
- Sys_Block_Process
 - process_Management.c, 156
- Sys_Camera_PreProcessor
 - camera.c, 67
- Sys_Change_Stack_HDI
 - process_Management_HDI.c, 109
 - process_Management_HDI.h, 111
- Sys_Clear_EventData
 - data.c, 147
 - data.h, 152
 - process_Management.h, 163
- Sys_Clear_EventRegister
 - data.c, 147
 - data.h, 152
- Sys_Contine_I2C
 - i2c.c, 79
 - i2c.h, 82
- Sys_Contine_I2C_HDI
 - i2c_HDI.c, 89
 - i2c_HDI.h, 92
- Sys_Continue_IOTimer
 - io.c, 53
 - io.h, 56
- Sys_Continue_IOTimer_HDI
 - io_HDI.c, 95
 - io_HDI.h, 98
- Sys_Continue_Pocess
 - process_Management.c, 157
- Sys_Continue_SystemTimer
 - system_Timer.c, 173
 - system_Timer.h, 175
- Sys_Continue_SystemTimer_HDI
 - system_Timer_HDI.c, 128
 - system_Timer_HDI.h, 130
- Sys_Delete_Process
 - data.c, 148
 - data.h, 152
- Sys_Disable_IOTimerInterrupt
 - io.c, 53
 - io.h, 56
- Sys_Disable_IOTimerInterrupt_HDI
 - io_HDI.c, 95
 - io_HDI.h, 98
- Sys_Disable_TimerInterrupt
 - system_Timer.c, 173
 - system_Timer.h, 175
- Sys_Disable_TimerInterrupt_HDI
 - system_Timer_HDI.c, 128
 - system_Timer_HDI.h, 130
- Sys_Enable_IOTimerInterrupt
 - io.c, 53
 - io.h, 56
- Sys_Enable_IOTimerInterrupt_HDI
 - io_HDI.c, 95
 - io_HDI.h, 98
- Sys_Enable_TimerInterrupt
 - system_Timer.c, 173
- system_Timer.h, 175
- Sys_Enable_TimerInterrupt_HDI
 - system_Timer_HDI.c, 128
 - system_Timer_HDI.h, 130
- Sys_End_AtomicSection
 - interrupts.c, 49
 - interrupts.h, 51
- Sys_End_CriticalSection
 - process_Management.c, 157
 - process_Management.h, 165
- Sys_Execute_All_EventHandler
 - process_Management.c, 157
 - process_Management.h, 165
- Sys_Execute_Events_in_ProcessList
 - process_Management.c, 157
- Sys_Find_Event
 - events.c, 42
- Sys_Find_EventHandler
 - data.h, 153
- Sys_Find_Process
 - data.c, 148
 - data.h, 153
- Sys_Force_IOTimerInterrupt
 - io.c, 53
 - io.h, 56
- Sys_Force_IOTimerInterrupt_HDI
 - io_HDI.c, 95
 - io_HDI.h, 98
- Sys_Force_TimerInterrupt
 - system_Timer.c, 173
 - system_Timer.h, 175
- Sys_Force_TimerInterrupt_HDI
 - system_Timer_HDI.c, 128
 - system_Timer_HDI.h, 130
- Sys_Free
 - memory.c, 61
 - memory.h, 63
- Sys_Get_LeftWheelSpeed
 - motors.c, 100
 - motors.h, 104
- Sys_Get_Number_Processes
 - process_Management.c, 157
 - process_Management.h, 165
- Sys_Get_RightWheelSpeed
 - motors.c, 100
 - motors.h, 104
- Sys_Get_SystemClock
 - io_clock.c, 58
 - io_clock.h, 60
- Sys_Get_SystemTime
 - io_clock.c, 58
 - io_clock.h, 60
- Sys_HasRemoteC_Sent_New_Data
 - remoteControl.c, 113
- Sys_I2C_AppendMessages
 - i2c_data.c, 84
 - i2c_data.h, 87
- Sys_I2C_FreeMessages

- i2c_data.c, [84](#)
 - i2c_data.h, [87](#)
- Sys_I2C_Read
 - i2c.c, [79](#)
 - i2c.h, [82](#)
- Sys_I2C_ReadByte
 - i2c.c, [79](#)
- Sys_I2C_ReadByte_HDI
 - i2c_HDI.c, [89](#)
 - i2c_HDI.h, [92](#)
- Sys_I2C_RemoveOldestMessage
 - i2c_data.c, [84](#)
 - i2c_data.h, [87](#)
- Sys_I2C_Send_ACK
 - i2c.c, [79](#)
- Sys_I2C_Send_ACK_HDI
 - i2c_HDI.c, [89](#)
 - i2c_HDI.h, [92](#)
- Sys_I2C_Send_NACK
 - i2c.c, [79](#)
- Sys_I2C_Send_NACK_HDI
 - i2c_HDI.c, [89](#)
 - i2c_HDI.h, [92](#)
- Sys_I2C_Send_Restart
 - i2c.c, [80](#)
- Sys_I2C_Send_Restart_HDI
 - i2c_HDI.c, [89](#)
 - i2c_HDI.h, [92](#)
- Sys_I2C_Send_Start
 - i2c.c, [80](#)
- Sys_I2C_Send_Start_HDI
 - i2c_HDI.c, [89](#)
 - i2c_HDI.h, [92](#)
- Sys_I2C_Send_Stop
 - i2c.c, [80](#)
- Sys_I2C_Send_Stop_HDI
 - i2c_HDI.c, [89](#)
 - i2c_HDI.h, [92](#)
- Sys_I2C_SentBytes
 - i2c.c, [80](#)
 - i2c.h, [82](#)
- Sys_I2C_Start_Reading
 - i2c.c, [80](#)
- Sys_I2C_Start_Reading_HDI
 - i2c_HDI.c, [89](#)
 - i2c_HDI.h, [92](#)
- Sys_I2C_WriteByte
 - i2c.c, [80](#)
- Sys_I2C_WriteByte_HDI
 - i2c_HDI.c, [89](#)
 - i2c_HDI.h, [92](#)
- sys_I2C_mode
 - i2c_data.h, [86](#)
- sys_I2C_state
 - i2c_data.h, [86](#)
- Sys_IOTimer_code_HDI
 - io_HDI.c, [95](#)
 - io_HDI.h, [98](#)
- Sys_Init_Camera
 - camera.c, [67](#)
 - camera.h, [70](#)
- Sys_Init_Clock
 - io_clock.c, [58](#)
 - io_clock.h, [60](#)
- Sys_Init_I2C
 - i2c.c, [80](#)
 - i2c.h, [83](#)
- Sys_Init_I2C_HDI
 - i2c_HDI.c, [89](#)
 - i2c_HDI.h, [92](#)
- Sys_Init_IOManagement
 - io.c, [53](#)
 - io.h, [56](#)
- Sys_Init_IOTimer_HDI
 - io_HDI.c, [95](#)
 - io_HDI.h, [98](#)
- Sys_Init_Kernel
 - system.c, [177](#)
 - system.h, [179](#)
- Sys_Init_Motors
 - motors.c, [100](#)
 - motors.h, [104](#)
- Sys_Init_Process_Management
 - process_Management.c, [157](#)
 - process_Management.h, [165](#)
- Sys_Init_Process_Management_HDI
 - process_Management_HDI.c, [109](#)
 - process_Management_HDI.h, [111](#)
- Sys_Init_RemoteControl
 - remoteControl.c, [113](#)
 - remoteControl.h, [120](#)
- Sys_Init_RemoteControl_HDI
 - remoteControl_HDI.c, [123](#)
 - remoteControl_HDI.h, [126](#)
- Sys_Init_SystemTime
 - io_clock.c, [58](#)
 - io_clock.h, [60](#)
- Sys_Init_SystemTimer
 - system_Timer.c, [173](#)
 - system_Timer.h, [175](#)
- Sys_Init_SystemTimer_HDI
 - system_Timer_HDI.c, [128](#)
 - system_Timer_HDI.h, [130](#)
- Sys_Init_UART1
 - uart.c, [134](#)
 - uart.h, [137](#)
- Sys_Init_UART1_HDI
 - uart_HDI.c, [140](#)
 - uart_HDI.h, [144](#)
- Sys_Init_UART2
 - uart.c, [134](#)
 - uart.h, [137](#)
- Sys_Init_UART2_HDI
 - uart_HDI.c, [140](#)
 - uart_HDI.h, [144](#)
- Sys_Insert_Process_to_List

- data.c, 148
- data.h, 153
- Sys_Interprocess_EventHandling
 - process_Management.c, 157
- Sys_IsEventRegistered
 - events.c, 42
 - events.h, 46
- Sys_Kill_Process
 - process_Management.c, 157
 - process_Management.h, 165
- Sys_Kill_Zombies
 - process_Management.c, 159
 - process_Management.h, 165
- Sys_LeftMotor_Controller
 - motors.c, 101
- Sys_LeftMotor_EventHandler
 - motors.c, 101
- Sys_LeftMotor_Reset
 - motors.c, 101
- Sys_LeftMotor_SetPhase_HDI
 - motors_HDI.c, 105
 - motors_HDI.h, 107
- Sys_Malloc
 - memory.c, 62
 - memory.h, 63
- Sys_Memcpy
 - memory.c, 62
 - memory.h, 63
- Sys_Next_EventHandler
 - data.c, 148
 - data.h, 153
- Sys_Pause_I2C
 - i2c.c, 80
 - i2c.h, 83
- Sys_Pause_I2C_HDI
 - i2c_HDI.c, 90
 - i2c_HDI.h, 92
- Sys_Process_newFrame
 - camera.c, 67
- Sys_Process_newLine
 - camera.c, 67
- Sys_Process_newPixel
 - camera.c, 68
- Sys_Read_UART1_ISR
 - uart_HDI.c, 140
 - uart_HDI.h, 145
- Sys_Read_UART2_ISR
 - uart_HDI.c, 140
 - uart_HDI.h, 145
- Sys_Receive_RemoteControl_Msg
 - remoteControl.c, 113
 - remoteControl.h, 121
- Sys_Register_Event
 - events.c, 43
 - events.h, 46
- Sys_Register_IOHandler
 - io.c, 53
 - io.h, 56
- Sys_RemoteC_Get_Address
 - remoteControl.c, 113
 - remoteControl.h, 121
- Sys_RemoteC_Get_CheckBit
 - remoteControl.c, 113
 - remoteControl.h, 121
- Sys_RemoteC_Get_Data
 - remoteControl.c, 114
 - remoteControl.h, 121
- Sys_RemoteC_Received_New_Data
 - remoteControl.h, 121
- Sys_Remove_All_Event_Subscriptions
 - process_Management.c, 159
 - process_Management.h, 165
- Sys_Remove_Event_Subscription
 - process_Management.c, 159
 - process_Management.h, 165
- Sys_Remove_Event_from_EventRegister
 - data.c, 149
 - data.h, 153
- Sys_Remove_Process_from_List
 - data.c, 149
 - data.h, 154
- Sys_Reset_IOTimer
 - io.c, 54
 - io.h, 56
- Sys_Reset_IOTimer_HDI
 - io_HDI.c, 95
 - io_HDI.h, 98
- Sys_Reset_SystemTimer
 - system_Timer.c, 173
 - system_Timer.h, 175
- Sys_Reset_SystemTimer_HDI
 - system_Timer_HDI.c, 128
 - system_Timer_HDI.h, 130
- Sys_RightMotor_Controller
 - motors.c, 101
- Sys_RightMotor_EventHandler
 - motors.c, 101
- Sys_RightMotor_Reset
 - motors.c, 101
- Sys_RightMotor_SetPhase_HDI
 - motors_HDI.c, 106
 - motors_HDI.h, 107
- Sys_Save_Running_Process_HDI
 - process_Management_HDI.c, 109
 - process_Management_HDI.h, 111
- Sys_Scheduler_RoundRobin
 - scheduler.c, 169
 - scheduler.h, 171
- Sys_Send_Event
 - events.c, 43
 - events.h, 47
- Sys_Send_IntEvent
 - events.c, 43
 - events.h, 47
- Sys_Set_Defaults_Info
 - scheduler.c, 169

- [scheduler.h](#), [171](#)
- [Sys_Set_Defaults_PCB](#)
 - [data.c](#), [149](#)
 - [data.h](#), [154](#)
- [Sys_Set_LeftWheelSpeed](#)
 - [motors.c](#), [102](#)
 - [motors.h](#), [104](#)
- [Sys_Set_Preprocessing](#)
 - [camera.c](#), [68](#)
 - [camera.h](#), [70](#)
- [Sys_Set_RightWheelSpeed](#)
 - [motors.c](#), [102](#)
 - [motors.h](#), [104](#)
- [Sys_Set_Running_Process_to_Zombie](#)
 - [process_Management.c](#), [159](#)
- [Sys_SetReadingFunction_UART1](#)
 - [uart.c](#), [134](#)
 - [uart.h](#), [137](#)
- [Sys_SetReadingFunction_UART2](#)
 - [uart.c](#), [134](#)
 - [uart.h](#), [137](#)
- [Sys_Start_AtomicSection](#)
 - [interrupts.c](#), [49](#)
 - [interrupts.h](#), [51](#)
- [Sys_Start_Camera](#)
 - [camera.c](#), [68](#)
 - [camera.h](#), [72](#)
- [Sys_Start_CriticalSection](#)
 - [process_Management.c](#), [159](#)
 - [process_Management.h](#), [167](#)
- [Sys_Start_I2C](#)
 - [i2c.c](#), [80](#)
 - [i2c.h](#), [83](#)
- [Sys_Start_I2C_HDI](#)
 - [i2c_HDI.c](#), [90](#)
 - [i2c_HDI.h](#), [93](#)
- [Sys_Start_IOManagement](#)
 - [io.c](#), [54](#)
 - [io.h](#), [56](#)
- [Sys_Start_IOTimer_HDI](#)
 - [io_HDI.c](#), [95](#)
 - [io_HDI.h](#), [98](#)
- [Sys_Start_Kernel](#)
 - [system.c](#), [177](#)
 - [system.h](#), [179](#)
- [Sys_Start_Process](#)
 - [process_Management.c](#), [159](#)
 - [process_Management.h](#), [167](#)
- [Sys_Start_Process_HDI](#)
 - [process_Management_HDI.c](#), [109](#)
 - [process_Management_HDI.h](#), [111](#)
- [Sys_Start_RemoteControl](#)
 - [remoteControl.c](#), [114](#)
 - [remoteControl.h](#), [121](#)
- [Sys_Start_RemoteControl_HDI](#)
 - [remoteControl_HDI.c](#), [123](#)
 - [remoteControl_HDI.h](#), [126](#)
- [Sys_Start_SystemTimer](#)
 - [system_Timer.c](#), [173](#)
 - [system_Timer.h](#), [175](#)
- [Sys_Start_SystemTimer_HDI](#)
 - [system_Timer_HDI.c](#), [128](#)
 - [system_Timer_HDI.h](#), [130](#)
- [Sys_Start_UART1](#)
 - [uart.c](#), [134](#)
 - [uart.h](#), [137](#)
- [Sys_Start_UART1_HDI](#)
 - [uart_HDI.c](#), [140](#)
 - [uart_HDI.h](#), [145](#)
- [Sys_Start_UART2](#)
 - [uart.c](#), [135](#)
 - [uart.h](#), [137](#)
- [Sys_Start_UART2_HDI](#)
 - [uart_HDI.c](#), [141](#)
 - [uart_HDI.h](#), [145](#)
- [Sys_Stop_I2C](#)
 - [i2c.c](#), [81](#)
 - [i2c.h](#), [83](#)
- [Sys_Stop_I2C_HDI](#)
 - [i2c_HDI.c](#), [90](#)
 - [i2c_HDI.h](#), [93](#)
- [Sys_Stop_IOManagement](#)
 - [io.c](#), [54](#)
 - [io.h](#), [56](#)
- [Sys_Stop_IOTimer](#)
 - [io.c](#), [54](#)
 - [io.h](#), [57](#)
- [Sys_Stop_IOTimer_HDI](#)
 - [io_HDI.c](#), [95](#)
 - [io_HDI.h](#), [98](#)
- [Sys_Stop_SystemTimer](#)
 - [system_Timer.c](#), [173](#)
 - [system_Timer.h](#), [176](#)
- [Sys_Stop_SystemTimer_HDI](#)
 - [system_Timer_HDI.c](#), [128](#)
 - [system_Timer_HDI.h](#), [131](#)
- [Sys_Subscribe_to_Event](#)
 - [events.c](#), [43](#)
 - [events.h](#), [47](#)
- [Sys_Switch_Process](#)
 - [process_Management.c](#), [159](#)
 - [process_Management.h](#), [167](#)
- [Sys_Switch_Process_HDI](#)
 - [process_Management_HDI.c](#), [109](#)
 - [process_Management_HDI.h](#), [111](#)
- [Sys_Switch_to_next_Process](#)
 - [process_Management.c](#), [161](#)
 - [process_Management.h](#), [167](#)
- [Sys_SystemClock_Counter](#)
 - [io_clock.c](#), [58](#)
- [sys_UART1_TX_data](#)
 - [uart_HDI.c](#), [141](#)
 - [uart_HDI.h](#), [146](#)
- [sys_UART2_TX_data](#)
 - [uart_HDI.c](#), [141](#)
 - [uart_HDI.h](#), [146](#)

- events.c, [44](#)
 - events.h, [47](#)
- io.c, [54](#)
 - io.h, [57](#)
- events.c, [44](#)
- events.c, [44](#)
 - events.h, [48](#)
- events.c, [44](#)
 - events.h, [48](#)
- process_Management.c, [161](#)
 - process_Management.h, [167](#)
- process_Management.c, [161](#)
 - process_Management.h, [167](#)
- uart_HDI.c, [141](#)
 - uart_HDI.h, [145](#)
- uart_HDI.c, [141](#)
 - uart_HDI.h, [145](#)
- uart.c, [135](#)
 - uart.h, [138](#)
- uart.c, [135](#)
 - uart.h, [138](#)
- process_Management.c, [161](#)
 - process_Management.h, [168](#)
- data.c, [150](#)
 - data.h, [154](#)
- definitions.h, [40](#)
- next, [26](#)
 - size, [26](#)
 - value, [26](#)
- data, [27](#)
 - handler, [27](#)
 - i2c_device_address, [27](#)
 - length, [27](#)
 - next, [27](#)
 - write, [27](#)
- i2c_data.c, [85](#)
 - i2c_data.h, [87](#)
- io_HDI.c, [96](#)
 - io_HDI.h, [98](#)
- speed, [28](#)
- eventID, [28](#)
 - next, [28](#)
- data.c, [150](#)
 - data.h, [154](#)
- function, [32](#)
 - next, [32](#)
- event_register, [29](#)
 - framePointer, [29](#)
 - process_ID, [29](#)
 - process_stack, [29](#)
 - sheduler_info, [29](#)
 - stackPointer, [30](#)
 - stackPointerLimit, [30](#)
- next, [30](#)
 - pcb, [30](#)
 - previous, [30](#)
- buffered_data, [31](#)
 - condition, [31](#)
 - eventID, [31](#)
 - handler, [32](#)
 - next, [32](#)
 - previous, [32](#)
- system_Timer_HDI.c, [128](#)
 - system_Timer_HDI.h, [131](#)
- data.c, [150](#)
 - data.h, [154](#)
- eventID, [33](#)
 - next, [33](#)
 - subscribers, [33](#)
- blue, [34](#)
 - green, [34](#)
 - red, [34](#)
- data.c, [150](#)
 - data.h, [154](#)
- priority, [35](#)
 - state, [35](#)
- next, [35](#)
 - pid, [35](#)
- system_Timer.c, [174](#)
 - system_Timer.h, [176](#)
 - system_Timer_HDI.h, [131](#)
- data, [36](#)

- length, 36
- next, 36
- sys_zombies
 - data.c, 150
 - data.h, 154
- system.c, 176
 - Sys_Init_Kernel, 177
 - Sys_Start_Kernel, 177
- system.h, 178
 - SYS_CAMERA_USED, 179
 - SYS_MOTOR_USED, 179
 - SYS_REMOTECONTROL_USED, 179
 - SYS_UART1_USED, 179
 - Sys_Init_Kernel, 179
 - Sys_Start_Kernel, 179
- system_Timer.c
 - Sys_Continue_SystemTimer, 173
 - Sys_Disable_TimerInterrupt, 173
 - Sys_Enable_TimerInterrupt, 173
 - Sys_Force_TimerInterrupt, 173
 - Sys_Init_SystemTimer, 173
 - Sys_Reset_SystemTimer, 173
 - Sys_Start_SystemTimer, 173
 - Sys_Stop_SystemTimer, 173
 - Sys_todo_SystemTimer, 174
- system_Timer.h
 - Sys_Continue_SystemTimer, 175
 - Sys_Disable_TimerInterrupt, 175
 - Sys_Enable_TimerInterrupt, 175
 - Sys_Force_TimerInterrupt, 175
 - Sys_Init_SystemTimer, 175
 - Sys_Reset_SystemTimer, 175
 - Sys_Start_SystemTimer, 175
 - Sys_Stop_SystemTimer, 176
 - Sys_todo_SystemTimer, 176
- system_Timer_HDI.c
 - __attribute__, 128
 - Sys_Continue_SystemTimer_HDI, 128
 - Sys_Disable_TimerInterrupt_HDI, 128
 - Sys_Enable_TimerInterrupt_HDI, 128
 - Sys_Force_TimerInterrupt_HDI, 128
 - Sys_Init_SystemTimer_HDI, 128
 - Sys_Reset_SystemTimer_HDI, 128
 - Sys_Start_SystemTimer_HDI, 128
 - Sys_Stop_SystemTimer_HDI, 128
 - sys_process_scheduler, 128
- system_Timer_HDI.h
 - Sys_Continue_SystemTimer_HDI, 130
 - Sys_Disable_TimerInterrupt_HDI, 130
 - Sys_Enable_TimerInterrupt_HDI, 130
 - Sys_Force_TimerInterrupt_HDI, 130
 - Sys_Init_SystemTimer_HDI, 130
 - Sys_Reset_SystemTimer_HDI, 130
 - Sys_Start_SystemTimer_HDI, 130
 - Sys_Stop_SystemTimer_HDI, 131
 - sys_process_scheduler, 131
 - Sys_todo_SystemTimer, 131
- traps.c
 - __attribute__, 132
- UART 1&2, 22
- UART1_RX
 - definitions.h, 39
 - uart_HDI.h, 143
- UART1_RX_DIR
 - definitions.h, 39
 - uart_HDI.h, 144
- UART1_TX
 - definitions.h, 39
 - uart_HDI.h, 144
- UART1_TX_DIR
 - definitions.h, 39
 - uart_HDI.h, 144
- UART2_RX
 - definitions.h, 39
 - uart_HDI.h, 144
- UART2_RX_DIR
 - definitions.h, 39
 - uart_HDI.h, 144
- UART2_TX
 - definitions.h, 39
 - uart_HDI.h, 144
- UART2_TX_DIR
 - definitions.h, 39
 - uart_HDI.h, 144
- uart.c
 - SYS_UART1_BAUDRATE, 134
 - SYS_UART2_BAUDRATE, 134
 - Sys_Init_UART1, 134
 - Sys_Init_UART2, 134
 - Sys_SetReadingFunction_UART1, 134
 - Sys_SetReadingFunction_UART2, 134
 - Sys_Start_UART1, 134
 - Sys_Start_UART2, 135
 - Sys_Writeto_UART1, 135
 - Sys_Writeto_UART2, 135
- uart.h
 - Sys_Init_UART1, 137
 - Sys_Init_UART2, 137
 - Sys_SetReadingFunction_UART1, 137
 - Sys_SetReadingFunction_UART2, 137
 - Sys_Start_UART1, 137
 - Sys_Start_UART2, 137
 - Sys_Writeto_UART1, 138
 - Sys_Writeto_UART2, 138
- uart_HDI.c
 - __attribute__, 140
 - byte_counter_uart1, 141
 - byte_counter_uart2, 141
 - read_uart_1, 141
 - read_uart_2, 141
 - Sys_Init_UART1_HDI, 140
 - Sys_Init_UART2_HDI, 140
 - Sys_Read_UART1_ISR, 140
 - Sys_Read_UART2_ISR, 140
 - Sys_Start_UART1_HDI, 140
 - Sys_Start_UART2_HDI, 141

- [sys_UART1_TX_data](#), [141](#)
 - [sys_UART2_TX_data](#), [141](#)
 - [Sys_Write_UART1_ISR](#), [141](#)
 - [Sys_Write_UART2_ISR](#), [141](#)
- [uart_HDI.h](#)
 - [byte_counter_uart1](#), [145](#)
 - [byte_counter_uart2](#), [145](#)
 - [read_uart_1](#), [145](#)
 - [read_uart_2](#), [145](#)
 - [SYS_UART1_BAUDRATE](#), [143](#)
 - [SYS_UART2_BAUDRATE](#), [143](#)
 - [Sys_Init_UART1_HDI](#), [144](#)
 - [Sys_Init_UART2_HDI](#), [144](#)
 - [Sys_Read_UART1_ISR](#), [145](#)
 - [Sys_Read_UART2_ISR](#), [145](#)
 - [Sys_Start_UART1_HDI](#), [145](#)
 - [Sys_Start_UART2_HDI](#), [145](#)
 - [sys_UART1_TX_data](#), [146](#)
 - [sys_UART2_TX_data](#), [146](#)
 - [Sys_Write_UART1_ISR](#), [145](#)
 - [Sys_Write_UART2_ISR](#), [145](#)
 - [UART1_RX](#), [143](#)
 - [UART1_RX_DIR](#), [144](#)
 - [UART1_TX](#), [144](#)
 - [UART1_TX_DIR](#), [144](#)
 - [UART2_RX](#), [144](#)
 - [UART2_RX_DIR](#), [144](#)
 - [UART2_TX](#), [144](#)
 - [UART2_TX_DIR](#), [144](#)
- [uint](#)
 - [definitions.h](#), [40](#)
- [uint16](#)
 - [definitions.h](#), [40](#)
- [uint32](#)
 - [definitions.h](#), [40](#)
- [uint8](#)
 - [definitions.h](#), [40](#)
- [value](#)
 - [sys_event_data](#), [26](#)
- [WHITE](#)
 - [definitions.h](#), [41](#)
- [waiting_cycles](#)
 - [remoteControl_HDI.c](#), [123](#)
 - [remoteControl_HDI.h](#), [126](#)
- [write](#)
 - [sys_i2c_msg](#), [27](#)
- [YELLOW](#)
 - [definitions.h](#), [41](#)