roneos

Generated by Doxygen 1.8.2

Thu May 16 2013 18:12:26

Contents

Chapter 1

Main Page

1.1 Overview:

The r-one robots are designed by the Multi-Robotic Systems Lab at Rice University
http://mrsl.rice.edu/

1.1.1 Software stack

The code base is designed to be extensible, with a three-layer software stack:

- [Applications (i.e. `SensorTest`, `SuperDemo`)]
- [`ronelib` (basic behaviors that will be used to make other code)]
- [`roneos` (hardware, sensors, actuators, system-level code)]

1.2 Included in roneos:

- `Audio:`
- `InputOutput:`
- `NeighborListOps:`

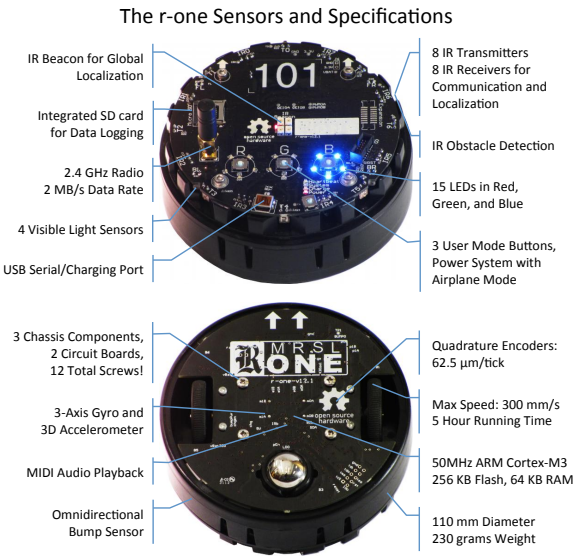

Figure 1.1: r-one robot specifications

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Module Documentation

5.1 nRF24L01 Register definitions

Macros

- #define NRF_ENAA_ENAA_P5 5
- #define NRF_ENAA_ENAA_P4 4
- #define NRF_ENAA_ENAA_P3 3
- #define NRF_ENAA_ENAA_P2 2
- #define NRF_ENAA_ENAA_P1 1
- #define NRF_ENAA_ENAA_P0 0
- #define NRF_DYNPD_DPL_P5 5
- #define NRF_DYNPD_DPL_P4 4
- #define NRF_DYNPD_DPL_P3 3
- #define NRF_DYNPD_DPL_P2 2
- #define NRF_DYNPD_DPL_P1 1
- #define NRF_DYNPD_DPL_P0 0
- #define NRF_FEATURE_EN_DPL 2
- #define NRF_FEATURE_EN_ACK_PAY 1
- #define NRF_FEATURE_EN_DYN_ACK 0

- Instruction Set -

- #define NRF_R_REGISTER 0x00
- #define NRF_W_REGISTER 0x20
- #define **REGISTER_MASK** 0x1F
- #define NRF_R_RX_PAYLOAD 0x61
- #define NRF_W_TX_PAYLOAD 0xA0
- #define NRF_FLUSH_TX 0xE1
- #define NRF_FLUSH_RX 0xE2
- #define NRF_REUSE_TX_PL 0xE3
- #define NRF_R_RX_PAYLOAD_WID 0x60
- #define NRF_W_ACK_PAYLOAD 0xA8
- #define NRF_W_TX_PAYLOAD_NOACK 0xB0
- #define NRF_NOP 0xFF
- #define NRF_LOCK_UNLOCK 0x50

- Register Memory Map -

- #define NRF_CONFIG 0x00
- #define NRF_EN_AA 0x01
- #define NRF_EN_RXADDR 0x02
- #define NRF_SETUP_AW 0x03
- #define NRF_SETUP_RETR 0x04
- #define NRF_RF_CH 0x05
- #define NRF_RF_SETUP 0x06
- #define NRF_STATUS 0x07
- #define NRF_OBSERVE_TX 0x08
- #define NRF_RPD 0x09
- #define NRF_RX_ADDR_P0 0x0A
- #define NRF_RX_ADDR_P1 0x0B
- #define NRF_RX_ADDR_P2 0x0C
- #define NRF_RX_ADDR_P3 0x0D
- #define NRF_RX_ADDR_P4 0x0E
- #define NRF_RX_ADDR_P5 0x0F
- #define NRF_TX_ADDR 0x10
- #define NRF_RX_PW_P0 0x11
- #define NRF_RX_PW_P1 0x12
- #define NRF_RX_PW_P2 0x13
- #define NRF_RX_PW_P3 0x14
- #define NRF_RX_PW_P4 0x15
- #define NRF_RX_PW_P5 0x16
- #define NRF_FIFO_STATUS 0x17
- #define NRF_DYNPD 0x1C
- #define NRF_FEATURE 0x1D

CONFIG register bit definitions

∗ Bit Mnemonics ∗/

- #define NRF_CONFIG_MASK_RX_DR 6
- #define NRF_CONFIG_MASK_TX_DS 5
- #define NRF_CONFIG_MASK_MAX_RT 4
- #define NRF_CONFIG_EN_CRC 3
- #define NRF_CONFIG_CRCO 2
- #define NRF_CONFIG_PWR_UP 1
- #define NRF_CONFIG_PRIM_RX 0

RF_SETUP register bit definitions

- #define NRF_SETUP_PLL_LOCK 4
- #define NRF_SETUP_RF_DR 3
- #define NRF_SETUP_RF_PWR1 2
- #define NRF_SETUP_RF_PWR0 1
- #define NRF_SETUP_LNA_HCURR 0

STATUS register bit definitions

- #define NRF_STATUS_RX_DR 6
- #define NRF_STATUS_TX_DS 5
- #define NRF_STATUS_MAX_RT 4
- #define NRF_STATUS_TX_FULL 0

FIFO_STATUS register bit definitions

- #define NRF_FIFOSTATUS_TX_REUSE 6
- #define NRF_FIFOSTATUS_TX_FIFO_FULL 5
- #define NRF_FIFOSTATUS_TX_EMPTY 4
- #define NRF_FIFOSTATUS_RX_FULL 1
- #define NRF_FIFOSTATUS_RX_EMPTY 0

5.1.1 Detailed Description

Header file defining register mapping with bit definitions. This file is radio-chip dependent, and are included with the hal_nrf.h

5.1.2 Macro Definition Documentation

5.1.2.1 #define NRF_CONFIG 0x00

nRF24L01 config register

5.1.2.2 #define NRF_CONFIG_CRCO 2

CONFIG register bit 2

5.1.2.3 #define NRF_CONFIG_EN_CRC 3

CONFIG register bit 3

5.1.2.4 #define NRF_CONFIG_MASK_MAX_RT 4

CONFIG register bit 4

5.1.2.5 #define NRF_CONFIG_MASK_RX_DR 6

CONFIG register bit 6

5.1.2.6 #define NRF_CONFIG_MASK_TX_DS 5

CONFIG register bit 5

5.1.2.7 #define NRF_CONFIG_PRIM_RX 0

CONFIG register bit 0

5.1.2.8 #define NRF_CONFIG_PWR_UP 1

CONFIG register bit 1

5.1.2.9 #define NRF_DYNPD 0x1C

nRF24L01 Dynamic payload setup

5.1.2.10 #define NRF_DYNPD_DPL_P0 0

dynamic payload enable

5.1.2.11 #define NRF_DYNPD_DPL_P1 1

dynamic payload enable

5.1.2.12 #define NRF_DYNPD_DPL_P2 2

dynamic payload enable

5.1.2.13 #define NRF_DYNPD_DPL_P3 3

dynamic payload enable

5.1.2.14 #define NRF_DYNPD_DPL_P4 4

dynamic payload enable

5.1.2.15 #define NRF_DYNPD_DPL_P5 5

dynamic payload enable

5.1.2.16 #define NRF_EN_AA 0x01

nRF24L01 enable Auto-Acknowledge register

5.1.2.17 #define NRF_EN_RXADDR 0x02

nRF24L01 enable RX addresses register

5.1.2.18 #define NRF_ENAA_ENAA_P0 0

dynamic payload enable

5.1.2.19 #define NRF_ENAA_ENAA_P1 1

dynamic payload enable

5.1.2.20 #define NRF_ENAA_ENAA_P2 2

dynamic payload enable

5.1.2.21 #define NRF_ENAA_ENAA_P3 3

dynamic payload enable

5.1.2.22 #define NRF_ENAA_ENAA_P4 4

dynamic payload enable

5.1.2.23 #define NRF_ENAA_ENAA_P5 5

dynamic payload enable

5.1.2.24 #define NRF_FEATURE 0x1D

nRF24L01 Exclusive feature setup

5.1.2.25 #define NRF_FEATURE_EN_ACK_PAY 1

dynamic payload enable

5.1.2.26 #define NRF_FEATURE_EN_DPL 2

dynamic payload enable

5.1.2.27 #define NRF_FEATURE_EN_DYN_ACK 0

dynamic payload enable

5.1.2.28 #define NRF_FIFO_STATUS 0x17

nRF24L01 FIFO status register

5.1.2.29 #define NRF_FIFOSTATUS_RX_EMPTY 0

FIFO_STATUS register bit 0

5.1.2.30 #define NRF_FIFOSTATUS_RX_FULL 1

FIFO_STATUS register bit 1

5.1.2.31 #define NRF_FIFOSTATUS_TX_EMPTY 4

FIFO_STATUS register bit 4

5.1.2.32 #define NRF_FIFOSTATUS_TX_FIFO_FULL 5

FIFO_STATUS register bit 5

5.1.2.33 #define NRF_FIFOSTATUS_TX_REUSE 6

FIFO_STATUS register bit 6

5.1.2.34 #define NRF_FLUSH_RX 0xE2

Flush RX register command

5.1.2.35 #define NRF_FLUSH_TX 0xE1

Flush TX register command

5.1.2.36 #define NRF_LOCK_UNLOCK 0x50

Lock/unlcok exclusive features

5.1.2.37 #define NRF_NOP 0xFF

No Operation command, used for reading status register

5.1.2.38 #define NRF_OBSERVE_TX 0x08

nRF24L01 transmit observe register

5.1.2.39 #define NRF_R_REGISTER 0x00

Register read command

5.1.2.40 #define NRF_R_RX_PAYLOAD 0x61

Read RX payload command

5.1.2.41 #define NRF_R_RX_PAYLOAD_WID 0x60

Read RX payload command

5.1.2.42 #define NRF_REUSE_TX_PL 0xE3

Reuse TX payload command

5.1.2.43 #define NRF_RF_CH 0x05

nRF24L01 RF channel register

5.1.2.44 #define NRF_RF_SETUP 0x06

nRF24L01 RF setup register

5.1.2.45 #define NRF_RPD 0x09

nRF24L01 receive power detect register

5.1.2.46 #define NRF_RX_ADDR_P0 0x0A

nRF24L01 receive address data pipe0

5.1.2.47 #define NRF_RX_ADDR_P1 0x0B

nRF24L01 receive address data pipe1

5.1.2.48 #define NRF_RX_ADDR_P2 0x0C

nRF24L01 receive address data pipe2

5.1.2.49 #define NRF_RX_ADDR_P3 0x0D

nRF24L01 receive address data pipe3

5.1.2.50 #define NRF_RX_ADDR_P4 0x0E

nRF24L01 receive address data pipe4

5.1.2.51 #define NRF_RX_ADDR_P5 0x0F

nRF24L01 receive address data pipe5

5.1.2.52 #define NRF_RX_PW_P0 0x11

nRF24L01 # of bytes in rx payload for pipe0

5.1.2.53 #define NRF_RX_PW_P1 0x12

nRF24L01 # of bytes in rx payload for pipe1

5.1.2.54 #define NRF_RX_PW_P2 0x13

nRF24L01 # of bytes in rx payload for pipe2

5.1.2.55 #define NRF_RX_PW_P3 0x14

nRF24L01 # of bytes in rx payload for pipe3

5.1.2.56 #define NRF_RX_PW_P4 0x15

nRF24L01 # of bytes in rx payload for pipe4

5.1.2.57 #define NRF_RX_PW_P5 0x16

nRF24L01 # of bytes in rx payload for pipe5

5.1.2.58 #define NRF_SETUP_AW 0x03

nRF24L01 setup of address width register

5.1.2.59 #define NRF_SETUP_LNA_HCURR 0

RF_SETUP register bit 0

5.1.2.60 #define NRF_SETUP_PLL_LOCK 4

RF_SETUP register bit 4

5.1.2.61 #define NRF_SETUP_RETR 0x04

nRF24L01 setup of automatic retransmission register

5.1.2.62 #define NRF_SETUP_RF_DR 3

RF_SETUP register bit 3

5.1.2.63 #define NRF_SETUP_RF_PWR0 1

RF_SETUP register bit 1

5.1.2.64 #define NRF_SETUP_RF_PWR1 2

RF_SETUP register bit 2

5.1.2.65 #define NRF_STATUS 0x07

nRF24L01 status register

5.1.2.66 #define NRF_STATUS_MAX_RT 4

STATUS register bit 4

5.1.2.67 #define NRF_STATUS_RX_DR 6

STATUS register bit 6

5.1.2.68 #define NRF_STATUS_TX_DS 5

STATUS register bit 5

5.1.2.69 #define NRF_STATUS_TX_FULL 0

STATUS register bit 0

5.1.2.70 #define NRF_TX_ADDR 0x10

nRF24L01 transmit address

5.1.2.71 #define NRF_W_ACK_PAYLOAD 0xA8

Write ACK payload command

5.1.2.72 #define NRF_W_REGISTER 0x20

Register write command

5.1.2.73 #define NRF_W_TX_PAYLOAD 0xA0

Write TX payload command

5.1.2.74 #define NRF_W_TX_PAYLOAD_NOACK 0xB0

Write ACK payload command

5.2 System>

Functions

- void neighborsDisable (void)

  Disable neighbor xmit/recv.
- void neighborsXmitEnable (boolean neighbor_xmit_enable_arg)

  Enable neighbor to transmit messages.
- void neighborsInit (uint32 neighbor_period_arg)

  Initialize neighbors and start neighbors task.
- void neighborsSetPeriod (uint32 neighbor_period_arg)

  Set neighbor period, neighbor timeout, and obstacle timeout proportional to argument.
- void neighborsSetTimeoutRounds (uint8 timeoutRounds, uint8 minActive, uint8 maxInactive)

  Set neighbor period, neighbor timeout, and obstacle timeout proportional to argument.
- void neighborsIgnore (uint8 neighborID)

  Tries add neighborID to list of neighbors to ignore.
- uint32 neighborsGetPeriod (void)

  Get neighbor period.
- void nbrPrint (Nbr *nbrPtr)

  Print information on neighbor (and information of neighbor's neighbors).
- void obstaclePrint (void)

  Print the obstacle data from the IR sensors.
- void nbrPrintData (Nbr *nbrPtr, uint32 round)

  Print header and neighbor data.
- uint8 irObstaclesGetBits (void)

  Get IR obstacle bits.
- uint8 * irObstaclesGetBitMatrix (void)

  Get IR obstacle bit matrix.
- int16 irObstaclesGetBearing (void)

  Get IR obstacle bearing.
- void neighborsGetMutex (void)

  Get neighbors mutex.
- void neighborsPutMutex (void)

  Put neighbors mutex.
- void neighborsTask (void *parameters)

  The neighbor update system task.
- uint32 neighborsGetRound (void)

  Get neighbor round from neighbor data.
- boolean neighborsNewRoundCheck (uint32 *roundOldPtr)

  Check to see if there is a new neighbor round. Updates the variable at the pointer.
- boolean nbrIsBeacon (Nbr *nbrPtr)

  Returns true if this neighbor is a beacon.
- boolean nbrIsRobot (Nbr *nbrPtr)

  Returns true if this neighbor is a beacon.
- uint8 nbrGetID (Nbr *nbrPtr)

  Get neighbor ID.
- int32 nbrGetBearing (Nbr *nbrPtr)

  Get neighbor bearing.
- int32 nbrGetOrientation (Nbr *nbrPtr)

  Get neighbor orientation.
- int32 nbrGetRange (Nbr *nbrPtr)

Get neighbor range.

- boolean nbrGetOrientationValid (Nbr *nbrPtr)

  Get neighbor orientation valid.

- uint8 nbrGetRangeBits (Nbr *nbrPtr)

  Get neighbor range bits.

- uint8 nbrGetReceiverBits (Nbr *nbrPtr)

  Get neighbor receiver bits.

- uint8 nbrGetTransmitterBits (Nbr *nbrPtr)

  Get neighbor transmitter bits.

- uint32 nbrGetUpdateTime (Nbr *nbrPtr)

  Get neighbor update time.

- uint32 nbrGetUpdateRound (Nbr *nbrPtr)

  Get neighbor update round.

5.2.1 Detailed Description

```
Neighbor system allows a robot to communicate with its neighbors.\n
Function neighborsTask is performed constantly at every WHAT.\n
```

}@

5.2.2 Function Documentation

5.2.2.1 int16 irObstaclesGetBearing (void)

Get IR obstacle bearing.

This function returns the bearing of an IR obsracle. It looks for contiguous obstacle bits, to select te largest (nearest) obstacle. This could cause dithering for 1-bit sizd obtsacles.

Returns

the bump sensor bearing. returns 0 if there is no obstacle

5.2.2.2 uint8* irObstaclesGetBitMatrix (void)

Get IR obstacle bit matrix.

Returns

IR obstacle bit matrix

5.2.2.3 uint8 irObstaclesGetBits (void)

Get IR obstacle bits.

Returns

IR obstacle bits

5.2.2.4 int32 nbrGetBearing (Nbr * *nbrPtr*)

Get neighbor bearing.

Parameters

nbrPtr	neighbor pointer

Returns

returns bearing. ranges within [-pi, pi)

5.2.2.5 uint8 nbrGetID (Nbr * *nbrPtr*)

Get neighbor ID.

Parameters

nbrPtr	neighbor pointer

Returns

ID

5.2.2.6 int32 nbrGetOrientation (Nbr * *nbrPtr*)

Get neighbor orientation.

Parameters

nbrPtr	neighbor pointer

Returns

orientation. ranges within [-pi, pi)

5.2.2.7 boolean nbrGetOrientationValid (Nbr * *nbrPtr*)

Get neighbor orientation valid.

Parameters

nbrPtr	neighbor pointer

Returns

whether orientation is valid

5.2.2.8 int32 nbrGetRange (Nbr * *nbrPtr*)

Get neighbor range.

Parameters

nbrPtr	neighbor pointer

Returns

orientation. ranges within [-pi, pi)

5.2.2.9 uint8 nbrGetRangeBits (Nbr * *nbrPtr*)

Get neighbor range bits.

Range bits are recieverBitCount + orientationBitCount

Parameters

nbrPtr	neighbor pointer

Returns

range bits

5.2.2.10 uint8 nbrGetReceiverBits (Nbr * *nbrPtr*)

Get neighbor receiver bits.

Receiver bits are the actual receivers the message was received on

Parameters

nbrPtr	neighbor pointer

Returns

receiver bits

5.2.2.11 uint8 nbrGetTransmitterBits (Nbr * *nbrPtr*)

Get neighbor transmitter bits.

Receiver bits are the actual transmitter the message was received from

Parameters

nbrPtr	neighbor pointer

Returns

transmitter bits

5.2.2.12 uint32 nbrGetUpdateRound (Nbr * *nbrPtr*)

Get neighbor update round.

Parameters

nbrPtr	neighbor pointer

Returns

the last round this neighbor was heard from

5.2.2.13 uint32 nbrGetUpdateTime (Nbr * *nbrPtr*)

Get neighbor update time.

Parameters

nbrPtr	neighbor pointer

Returns

update time

5.2.2.14 boolean nbrIsBeacon (Nbr * *nbrPtr*)

Returns true if this neighbor is a beacon.

Parameters

nbrPtr	neighbor pointer

Returns

true if the neighbor is a IR beacon

5.2.2.15 boolean nbrIsRobot (Nbr * *nbrPtr*)

Returns true if this neighbor is a beacon.

Parameters

nbrPtr	neighbor pointer

Returns

true if the neighbor is a IR beacon

5.2.2.16 void nbrPrint (Nbr * *nbrPtr*)

Print information on neighbor (and information of neighbor's neighbors).

Print roneID and neighbor's ID, bear, orientation, orientation valid Print name and value of each neighbor message.

Parameters

nbrPtr	neighbor pointer

Returns

void

5.2.2.17 void nbrPrintData (Nbr * *nbrPtr,* uint32 *round*)

Print header and neighbor data.

Print header once. Print id, time, round; neighbor's ID, bearing, update time; neighbor's neighbor's ID, bearing, update time.

Parameters

nbrPtr	neighbor pointer
round	the round number

Returns

void

5.2.2.18 void neighborsDisable (void)

Disable neighbor xmit/recv.

Returns

void

5.2.2.19 void neighborsGetMutex (void)

Get neighbors mutex.

Returns

void

5.2.2.20 uint32 neighborsGetPeriod (void)

Get neighbor period.

Returns

void

5.2.2.21 uint32 neighborsGetRound (void)

Get neighbor round from neighbor data.

Returns

neighbor round

5.2.2.22 void neighborsIgnore (uint8 *neighborID*)

Tries add neighborID to list of neighbors to ignore.

Parameters

neighborID	the neighbor we want to ignore (no longer monitor)

Returns

void

5.2.2.23 void neighborsInit (uint32 *neighbor_period_arg*)

Initialize neighbors and start neighbors task.

Initializes neighbor period, neighbor timeout, obstacle timeout. Initialize neighborData Sets message length. Puts 7-bit roneID in message. Semaphore implementing neighborsMutex created.

Parameters

neighbor_period_arg	the neighbor period in rounds

Returns

void

5.2.2.24 boolean neighborsNewRoundCheck (uint32 * *roundOldPtr*)

Check to see if there is a new neighbor round. Updates the variable at the pointer.

Returns

TRUE if the neighbor round has changed

5.2.2.25 void neighborsPutMutex (void)

Put neighbors mutex.

Returns

void

5.2.2.26 void neighborsSetPeriod (uint32 *neighbor_period_arg*)

Set neighbor period, neighbor timeout, and obstacle timeout proportional to argument.

Parameters

neighbor_period_arg	the neighbor period length in rounds

Returns

void

5.2.2.27 void neighborsSetTimeoutRounds (uint8 *timeoutRounds,* uint8 *minActive,* uint8 *maxInactive*)

Set neighbor period, neighbor timeout, and obstacle timeout proportional to argument.

Parameters

neighbor_period_arg	the neighbor period length in rounds

Returns

void

5.2.2.28 void neighborsTask (void * *parameters*)

The neighbor update system task.

Returns

void

5.2.2.29 void neighborsXmitEnable (boolean *neighbor_xmit_enable_arg*)

Enable neighbor to transmit messages.

Parameters

neighbor_xmit_enable_arg	a boolean that allows enable or not

Returns

void

5.2.2.30 void obstaclePrint (void)

Print the obstacle data from the IR sensors.

Returns

void

Chapter 6

Data Structure Documentation

6.1 errorMsg Struct Reference

Error message includes information to track error.

6.1.1 Detailed Description

Error message includes information to track error.

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

- System/system.c

6.2 guiCmdData Struct Reference

Commands from the GUI.

6.2.1 Detailed Description

Commands from the GUI.

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

- SerialIO/systemCommands.c

6.3 irCommsMessage Struct Reference

message received over IR ring (contains the sending transmitter ID and receiving ID

`#include <ir_comms.h>`

6.3.1 Detailed Description

message received over IR ring (contains the sending transmitter ID and receiving ID

NOTE! If you are using the neighbor system, you have 7 less bits than this says.

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

- IRComms/ir_comms.h

6.4 IRRangeData Struct Reference

Data from IR signal.

`#include <neighbors.h>`

6.4.1 Detailed Description

Data from IR signal.

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

- IRComms/neighbors.h

6.5 motorVelocityData Struct Reference

contains information on the robot's state used for smooth motor control

6.5.1 Detailed Description

contains information on the robot's state used for smooth motor control

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

- Motors/motor.c

6.6 Nbr Struct Reference

information stored on a network neighbor

`#include <neighbors.h>`

6.6.1 Detailed Description

information stored on a network neighbor

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

- IRComms/neighbors.h

6.7 NbrData Struct Reference

linked list of data on messages

`#include <neighbors.h>`

6.7.1 Detailed Description

linked list of data on messages

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

- IRComms/neighbors.h

6.8 NbrDatabase Struct Reference

Array containing data on all the network neighbors.

`#include <neighbors.h>`

6.8.1 Detailed Description

Array containing data on all the network neighbors.

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

- IRComms/neighbors.h

6.9 NbrList Struct Reference

Array containing data on all the network neighbors??

`#include <neighbors.h>`

6.9.1 Detailed Description

Array containing data on all the network neighbors??

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

- IRComms/neighbors.h

6.10 NbrMsgRadio Struct Reference

linked list a neighbor's messages

`#include <neighbors.h>`

6.10.1 Detailed Description

linked list a neighbor's messages

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

- IRComms/neighbors.h

6.11 NbrMsgRadioNbrData Struct Reference

IR message a neighbor has.

`#include <neighbors.h>`

6.11.1 Detailed Description

IR message a neighbor has.

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

- IRComms/neighbors.h

6.12 Pose Struct Reference

The pose of a robot, it's position and orientation.

`#include <intMath.h>`

Data Fields

- int32 y
- int32 theta

6.12.1 Detailed Description

The pose of a robot, it's position and orientation.

6.12.2 Field Documentation

6.12.2.1 int32 Pose::theta

typically in milli-meters

6.12.2.2 int32 Pose::y

typically in milli-meters

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

- System/intMath.h

6.13 RadioCmd Struct Reference

A radio command is a linked list of the radio commands received.

`#include <radioCommand.h>`

6.13.1 Detailed Description

A radio command is a linked list of the radio commands received.

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

- InputOutput/radioCommand.h

6.14 RadioMessage Union Reference

radio message and metrics union

`#include <radio.h>`

6.14.1 Detailed Description

radio message and metrics union

This is the union type for raw messages and commands

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

- InputOutput/radio.h

6.15 RadioMessageCommand Struct Reference

radio message and metrics

`#include <radio.h>`

6.15.1 Detailed Description

radio message and metrics

This is the radio message command. This is processed by the threads and filtered for type and subnet. Note: this must be the same length as the RadioMessageRaw for the union to work properly

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

- InputOutput/radio.h

6.16 RadioMessageRaw Struct Reference

radio message and metrics

`#include <radio.h>`

6.16.1 Detailed Description

radio message and metrics

This it the raw radio message. It should only be used at the low level, or for network snooping Note: this must be the same length as the RadioMessageCommand for the union to work properly

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

- InputOutput/radio.h

6.17 robotName Struct Reference

robot has a name and a numeric ID

6.17.1 Detailed Description

robot has a name and a numeric ID

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

- System/robot_names.c

6.18 SerialCmd Struct Reference

Serial commands are a linked list of commands containing the message and name.

`#include <serialCommand.h>`

6.18.1 Detailed Description

Serial commands are a linked list of commands containing the message and name.

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

- SerialIO/serialCommand.h

6.19 warningMessage Struct Reference

Warning message includes information to track warning.

6.19.1 Detailed Description

Warning message includes information to track warning.

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

- System/system.c

Chapter 7

File Documentation

7.1 Audio/audio.h File Reference

Files for controlling audio speaker and MIDI files.

```
#include "VS1053_PatchTable.h"
```

Functions

- void audioInit (void)

  Initialize audio chip.
- void audioNoteOn (uint8 instrument, uint8 key, uint8 velocity, uint32 duration)

  Plays given note for given time. This will schedule a note off message for the sound chip.
- void audioNoteOff (uint8 instrument, uint8 key)

  Turns off given note.
- void audioNoteOffAll (void)

  Turns off all notes.
- void audioVolume (uint8 volume)

  Set the volume level.
- void playMajorChord (uint8 instrument, uint8 baseNote, uint8 volume, uint32 duration)

  Plays a three note major chord.
- void playMinorChord (uint8 instrument, uint8 baseNote, uint8 volume, uint32 duration)

  Plays a three note minor chord.

7.1.1 Detailed Description

Files for controlling audio speaker and MIDI files.

Since

Jul 6, 2011

Author

Sunny Kim

7.1.2 Function Documentation

7.1.2.1 void audioInit (void)

Initialize audio chip.

Returns

void

7.1.2.2 void audioNoteOff (uint8 *instrument,* uint8 *key*)

Turns off given note.

Parameters

instrument	Instrument to stop
key	Note to turn off

Returns

void

7.1.2.3 void audioNoteOffAll (void)

Turns off all notes.

Returns

void

7.1.2.4 void audioNoteOn (uint8 *instrument,* uint8 *key,* uint8 *velocity,* uint32 *duration*)

Plays given note for given time. This will schedule a note off message for the sound chip.

Parameters

instrument	Instrument to play
key	Note to play
velocity	the velocity (volume) to play. range is from 0-127
duration	Duration of the note (ms)

7.1.2.5 void audioVolume (uint8 *volume*)

Set the volume level.

Parameters

volume	the output level for left and right channels. 0 = total silence, 255 = max volume

Returns

void

7.1.2.6 void playMajorChord (uint8 *instrument,* uint8 *baseNote,* uint8 *volume,* uint32 *duration*)

Plays a three note major chord.

Parameters

volume	the output level for left and right channels. 0 = total silence, 255 = max volume
baseNote	root of the chord (played at baseNote + middleC)
volume	the output level for left and right channels. 0 = total silence, 255 = max volume
duration	Duration of the note (ms)

Returns

void

7.1.2.7 void playMinorChord (uint8 *instrument,* uint8 *baseNote,* uint8 *volume,* uint32 *duration*)

Plays a three note minor chord.

Parameters

volume	the output level for left and right channels. 0 = total silence, 255 = max volume
baseNote	root of the chord (played at baseNote + middleC)
volume	the output level for left and right channels. 0 = total silence, 255 = max volume
duration	Duration of the note (ms)

Returns

void

7.2 Audio/Midi.c File Reference

functions for playing MIDI files on the robot

7.2.1 Detailed Description

functions for playing MIDI files on the robot

Since

Jul 20, 2011

Author

Sunny Kim

Warning

Many functions are not commented

7.3 Audio/MIDIFilesOS.h File Reference

```
#include "roneos.h"
```

7.4 InputOutput/blinky_led.c File Reference

Functions that control the blinky_led {heartbeat, System, Charge, Power}.

```
#include "inc/lm3s8962.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "roneos.h"
```

Functions

- void blinky_led_init (void)

  Initializes blinky.
- void blinkyLedSet (uint32 state)

  Sets the blinky on or off.
- void blinky_led_flash (uint32 delay)

  Flashes the blinky once with delay.
- void blinkyUpdate (void)

  Updates the blinky.
- void blinkyUpdateFast (void)

  Updates the blinky with a fast rate.
- void blinkySystemBuildMessage (uint8 *msg)

  Build message for blinky.
- void blinkySystemUpdate (void)

  Update blinky system.

7.4.1 Detailed Description

Functions that control the blinky_led {heartbeat, System, Charge, Power}.

7.4.2 Function Documentation

7.4.2.1 void blinky_led_flash (uint32 *delay*)

Flashes the blinky once with delay.

Flashes the blinky once (turns it on and then off) with a specified delay in between).

Parameters

delay	determines how long the delay is

Returns

void

7.4.2.2 void blinky_led_init (void)

Initializes blinky.

Initializes blinky with port B, pin 6 as output. Blinky is turned off with initialization.

Returns

void

7.4.2.3 void blinkyLedSet (uint32 *state*)

Sets the blinky on or off.

Parameters

state	determines whether the pin should be on or off (send 1 to turn on, 0 to turn off)

Returns

void

7.4.2.4 void blinkySystemBuildMessage (uint8 * *msg*)

Build message for blinky.

Initializes message to specific brightness.

Returns

void

7.4.2.5 void blinkySystemUpdate (void)

Update blinky system.

Update both the blinky system timer and the brightness.

Returns

void

7.4.2.6 void blinkyUpdate (void)

Updates the blinky.

Updates the blinky timer and turns blinky on or off depending on the timer.

Returns

void

7.4.2.7 void blinkyUpdateFast (void)

Updates the blinky with a fast rate.

Updates the blinky timer and turns blinky on or off depending on the timer.

Returns

void

7.5 InputOutput/buttons.c File Reference

functions for 3 buttons on the top of the robot (R,G,B) that can be programmed by the user.

```
#include "inc/lm3s8962.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "roneos.h"
```

Functions

- void buttons_init (void)

  Initializes the buttons.
- uint32 buttons_get (uint32 button)

  Gets the state of the specified button.
- void buttonsBuildMessage (uint8 *msg)

  Build a message containing the states of each button.

7.5.1 Detailed Description

functions for 3 buttons on the top of the robot (R,G,B) that can be programmed by the user.

Since

Jul 21, 2010

Author

jamesm

7.5.2 Function Documentation

7.5.2.1 uint32 buttons_get (uint32 *button*)

Gets the state of the specified button.

Tells you whether the specified button is on or off

Parameters

button	specifies which button to check (BUTTON_RED, BUTTON_BLUE, or BUTTON_GREEN)

Returns

void

7.5.2.2 void buttons_init (void)

Initializes the buttons.

Initializes the red, green, and blue buttons as input.

Returns

void

7.5.2.3 void buttonsBuildMessage (uint8 * *msg*)

Build a message containing the states of each button.

3 bits of message used. Blue, green, red.

Parameters

msg	the address where we place the message

Returns

void

7.6 InputOutput/hal_nrf_reg.h File Reference

Register definitions for the nRF HAL module.

Macros

- #define NRF_ENAA_ENAA_P5 5
- #define NRF_ENAA_ENAA_P4 4
- #define NRF_ENAA_ENAA_P3 3
- #define NRF_ENAA_ENAA_P2 2
- #define NRF_ENAA_ENAA_P1 1
- #define NRF_ENAA_ENAA_P0 0
- #define NRF_DYNPD_DPL_P5 5
- #define NRF_DYNPD_DPL_P4 4
- #define NRF_DYNPD_DPL_P3 3
- #define NRF_DYNPD_DPL_P2 2
- #define NRF_DYNPD_DPL_P1 1
- #define NRF_DYNPD_DPL_P0 0
- #define NRF_FEATURE_EN_DPL 2
- #define NRF_FEATURE_EN_ACK_PAY 1
- #define NRF_FEATURE_EN_DYN_ACK 0

- Instruction Set -

- #define NRF_R_REGISTER 0x00
- #define NRF_W_REGISTER 0x20
- #define **REGISTER_MASK** 0x1F
- #define NRF_R_RX_PAYLOAD 0x61
- #define NRF_W_TX_PAYLOAD 0xA0
- #define NRF_FLUSH_TX 0xE1
- #define NRF_FLUSH_RX 0xE2
- #define NRF_REUSE_TX_PL 0xE3
- #define NRF_R_RX_PAYLOAD_WID 0x60
- #define NRF_W_ACK_PAYLOAD 0xA8
- #define NRF_W_TX_PAYLOAD_NOACK 0xB0

- #define NRF_NOP 0xFF
- #define NRF_LOCK_UNLOCK 0x50

- Register Memory Map -

- #define NRF_CONFIG 0x00
- #define NRF_EN_AA 0x01
- #define NRF_EN_RXADDR 0x02
- #define NRF_SETUP_AW 0x03
- #define NRF_SETUP_RETR 0x04
- #define NRF_RF_CH 0x05
- #define NRF_RF_SETUP 0x06
- #define NRF_STATUS 0x07
- #define NRF_OBSERVE_TX 0x08
- #define NRF_RPD 0x09
- #define NRF_RX_ADDR_P0 0x0A
- #define NRF_RX_ADDR_P1 0x0B
- #define NRF_RX_ADDR_P2 0x0C
- #define NRF_RX_ADDR_P3 0x0D
- #define NRF_RX_ADDR_P4 0x0E
- #define NRF_RX_ADDR_P5 0x0F
- #define NRF_TX_ADDR 0x10
- #define NRF_RX_PW_P0 0x11
- #define NRF_RX_PW_P1 0x12
- #define NRF_RX_PW_P2 0x13
- #define NRF_RX_PW_P3 0x14
- #define NRF_RX_PW_P4 0x15
- #define NRF_RX_PW_P5 0x16
- #define NRF_FIFO_STATUS 0x17
- #define NRF_DYNPD 0x1C
- #define NRF_FEATURE 0x1D

CONFIG register bit definitions

- *Bit Mnemonics ∗/*

- #define NRF_CONFIG_MASK_RX_DR 6
- #define NRF_CONFIG_MASK_TX_DS 5
- #define NRF_CONFIG_MASK_MAX_RT 4
- #define NRF_CONFIG_EN_CRC 3
- #define NRF_CONFIG_CRCO 2
- #define NRF_CONFIG_PWR_UP 1
- #define NRF_CONFIG_PRIM_RX 0

RF_SETUP register bit definitions

- #define NRF_SETUP_PLL_LOCK 4
- #define NRF_SETUP_RF_DR 3
- #define NRF_SETUP_RF_PWR1 2
- #define NRF_SETUP_RF_PWR0 1
- #define NRF_SETUP_LNA_HCURR 0

STATUS register bit definitions

- #define NRF_STATUS_RX_DR 6
- #define NRF_STATUS_TX_DS 5
- #define NRF_STATUS_MAX_RT 4
- #define NRF_STATUS_TX_FULL 0

FIFO_STATUS register bit definitions

- #define NRF_FIFOSTATUS_TX_REUSE 6
- #define NRF_FIFOSTATUS_TX_FIFO_FULL 5
- #define NRF_FIFOSTATUS_TX_EMPTY 4
- #define NRF_FIFOSTATUS_RX_FULL 1
- #define NRF_FIFOSTATUS_RX_EMPTY 0

7.6.1 Detailed Description

Register definitions for the nRF HAL module.

7.7 InputOutput/ir_beacon.c File Reference

This code controls the 4 IR LEDS on the top center of the rone robot (these are the IR_beacons, and an IR sensitive camera can use these to track the robots).

```
#include <string.h>
#include "inc/lm3s8962.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/ssi.h"
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"
#include "driverlib/debug.h"
#include "driverlib/pwm.h"
#include "roneos.h"
#include "swarmCamLookupTable.h"
```

Functions

- void IRBeaconPreinit (void)

  Initializes IRBeacon.
- void IRBeaconInit (void)

  Initializes IRBeacon interrupt.
- void IRBeaconIntEnable ()

  Enables ir_beacon clock.
- void IRBeaconIntDisable ()

  Disables ir_beacon clock.
- void IRBeaconSetData (uint32 data)

  Sets the data in IRBeacon.
- void IRBeaconDisable (void)

  Disables IRBeacon.

7.7.1 Detailed Description

This code controls the 4 IR LEDS on the top center of the rone robot (these are the IR_beacons, and an IR sensitive camera can use these to track the robots). The init and preinit functions are typically called by functions in system.c

TODO: list the functions (and the calling order) that must be included in any main file in order to use the IRbeacons.

Since

Jul 22, 2010

Author

jamesm

7.7.2 Function Documentation

7.7.2.1 void IRBeaconDisable (void)

Disables IRBeacon.

Turns off IRBeacon LED and sets the timer to 0.

Returns

void

7.7.2.2 void IRBeaconInit (void)

Initializes IRBeacon interrupt.

Enables the 60hz IRBeacon interrupt.

Returns

void

7.7.2.3 void IRBeaconIntDisable ( )

Disables ir_beacon clock.

Returns

void

Currently unused

7.7.2.4 void IRBeaconIntEnable ( )

Enables ir_beacon clock.

Returns

void

Currently unused

7.7.2.5 void IRBeaconPreinit (void)

Initializes IRBeacon.

Enables the IRBeacon pin as an output. Turns IRBeacon off in the process.

Returns

void

7.7.2.6 void IRBeaconSetData (uint32 *data*)

Sets the data in IRBeacon.

Sets what the IRBeacon is going to output; also sets the timer for IRBeacon to 60.

This function, when called "IRBeaconSetData(roneID);" gives each robot a unique ID. TODO: this function should be called every second to get continuous localization?

Parameters

data	the output data (32 bit unsigned int)

Returns

void

7.8 InputOutput/leds.c File Reference

interface functions for LEDs on robot

```
#include "inc/lm3s8962.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/ssi.h"
#include "roneos.h"
```

7.8.1 Detailed Description

interface functions for LEDs on robot

7.9 InputOutput/Logger/diskio.c File Reference

Interfaces the FatFS file-system SD card function calls to the SD card implementation for roneos.

```
#include "roneos.h"
```

Functions

- DSTATUS disk_initialize (BYTE drive)

  Initialized the disk drive (the SD Card).
- DSTATUS disk_status (BYTE drive)

  Gets the current status of the disk (SD card).
- DWORD get_fattime (void)

7.9.1 Detailed Description

Interfaces the FatFS file-system SD card function calls to the SD card implementation for roneos.

Author

Jeremy Hunt

7.9.2 Function Documentation

7.9.2.1 DSTATUS disk_initialize (BYTE *drive*)

Initialized the disk drive (the SD Card).

Parameters

drive	The drive which is to be initialized (Must be 0).

Returns

DSTATUS The status of the drive as flags.

7.9.2.2 DSTATUS disk_status (BYTE *drive*)

Gets the current status of the disk (SD card).

Parameters

drive	The drive which is to be read (Must be 0).

Returns

DSTATUS The status of the drive as flags

7.9.2.3 DWORD get_fattime (void)

Returns the current time (currently a fake time) as a packed 32 bit value. Used in the FatFs File system.

7.10 InputOutput/Logger/sd_card.c File Reference

Functions that control the SD card.

```
#include "inc/lm3s8962.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "inc/hw_qei.h"
#include "driverlib/flash.h"
#include "driverlib/gpio.h"
#include "driverlib/pwm.h"
#include "driverlib/qei.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/systick.h"
#include "driverlib/uart.h"
#include "driverlib/ssi.h"
#include "driverlib/interrupt.h"
#include "roneos.h"
```

7.10.1 Detailed Description

Functions that control the SD card.

Authors

Jeremy Hunt and Nathan Alison

7.11 InputOutput/radio.c File Reference

turns WiFi radio on or off, sends and receives radio messages, radio interrupts

```
#include <stdio.h>
#include <string.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "inc/hw_qei.h"
#include "driverlib/flash.h"
#include "driverlib/gpio.h"
#include "driverlib/pwm.h"
#include "driverlib/qei.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/systick.h"
#include "driverlib/uart.h"
#include "driverlib/ssi.h"
#include "driverlib/interrupt.h"
#include "hal_nrf_reg.h"
#include "roneos.h"
```

Macros

- #define RADIO_IRQ_PORT GPIO_PORTC_BASE

Functions

- void radioIntEnable (void)

  Enables radio interrupt.
- void radioIntDisable (void)

  Disables radio interrupt.
- void radioIntHandler (void)

  Handles radio interrupt.
- void radioInit (void)

  Initializes the radio.
- void radioSendMessage (RadioMessage *messagePtr)

  Sends a message through the radio.
- void radioGetMessageBlocking (RadioMessage *messagePtr)

  Checks whether there is a message with blocking.

7.11.1 Detailed Description

turns WiFi radio on or off, sends and receives radio messages, radio interrupts

Since

Jul 9, 2010

Authors

sjb2, edited by lyncas

7.11.2 Macro Definition Documentation

7.11.2.1 #define RADIO_IRQ_PORT GPIO_PORTC_BASE

V6 Pin Definitions - Radio SS=PA7, SLP_TR=PA6, RST=PG0 , IRQ=PC5 V11 Pin Definitions - Radio SS=PA7, SLP_TR=PA6, RST=PG0 , IRQ=PC5

7.11.3 Function Documentation

7.11.3.1 void radioGetMessageBlocking (RadioMessage * *messagePtr*)

Checks whether there is a message with blocking.

Parameters

messagePtr	pointer to the message to be checked

Returns

void

7.11.3.2 void radioInit (void)

Initializes the radio.

Returns

void

7.11.3.3 void radioIntDisable (void)

Disables radio interrupt.

Returns

void

7.11.3.4 void radioIntEnable (void)

Enables radio interrupt.

Returns

void

7.11.3.5 void radioIntHandler (void)

Handles radio interrupt.

Returns

void

7.11.3.6 void radioSendMessage (RadioMessage * *messagePtr*)

Sends a message through the radio.

Parameters

messagePtr	pointer to the message to be sent

Can't call this function from within an ISR

7.12 IRComms/nbrData.h File Reference

Functions

- void nbrDataCreate (NbrData *nbrDataPtr, const char *name, uint8 size, uint8 value)

  Create a neighbor data. This is transmitted using the radio each neighbor round.
- void nbrDataCreateIR (NbrData *nbrDataPtr, const char *name, uint8 size, uint8 value)

  Create a IR neighbor data. This is transmitted via IR. The neighbor system makes one 7-bit data by default for the robot ID. Only data that need to share the fate of an IR message should go here. The rest should use radio (See nbrDataCreate())
- void nbrDataSet (NbrData *nbrDataPtr, uint8 value)

  Set the value of the neighbor data.
- uint8 nbrDataGet (NbrData *nbrDataPtr)

Get value in a neighbor message.

- uint8 nbrDataGetNbr (NbrData *nbrDataPtr, Nbr *nbrPtr)

  Get value of the local nbrData.
- const char * nbrDataGetName (NbrData *nbrDataPtr)

  Get name of neighbor message.
- uint8 nbrDataGetSize (NbrData *nbrDataPtr)

  Get size of neighbor message.
- void nbrDataPrintNbr (Nbr *nbrPtr)

  Print all the neighbor messages of a given neighbor.
- void nbrDataPrintNbrVerbose (Nbr *nbrPtr)

  Print all the neighbor messages of the given neighbor. Prints in verbose format, with names on separate lines.
- void nbrDataPrintHeaders (void)

  Print the nmes of all the nbr data in a header suitable for data logging.
- uint8 nbrDataCount (void)

  Get the total number of nbr data that have been created.

7.12.1 Detailed Description

Created on: Feb 12, 2013 Author: jamesm

7.12.2 Function Documentation

7.12.2.1 uint8 nbrDataCount (void)

Get the total number of nbr data that have been created.

Returns

the count of nbrData objects

7.12.2.2 void nbrDataCreate (NbrData * *nbrDataPtr*, const char * *name*, uint8 *size*, uint8 *value*)

Create a neighbor data. This is transmitted using the radio each neighbor round.

This function is called only once to init each neighbor data

Parameters

nbrMsgPtr	the pointer to the neighbor data
name	the name for the neighbor data
size	the size of the neighbor data
size	the value of the neighbor data

Returns

void

7.12.2.3 void nbrDataCreateIR (NbrData * *nbrDataPtr*, const char * *name*, uint8 *size*, uint8 *value*)

Create a IR neighbor data. This is transmitted via IR. The neighbor system makes one 7-bit data by default for the robot ID. Only data that need to share the fate of an IR message should go here. The rest should use radio (See nbrDataCreate())

This function is called only once to init each neighbor data

Parameters

nbrMsgPtr	the pointer to the neighbor data
name	the name for the neighbor data
size	the size of the neighbor data
size	the value of the neighbor data

Returns

void

7.12.2.4 uint8 nbrDataGet (NbrData * *nbrDataPtr*)

Get value in a neighbor message.

Parameters

nbrDataPtr	pointer to neighbor message

Returns

if nbrDataPtr is valid, value; else, 0

7.12.2.5 const char* nbrDataGetName (NbrData * *nbrDataPtr*)

Get name of neighbor message.

As specified in NbrMsgCreate()

Parameters

nbrMsgPtr	pointer to neighbor message

Returns

if nbrMsgPtr is valid, name; else, 0

7.12.2.6 uint8 nbrDataGetNbr (NbrData * *nbrDataPtr,* Nbr * *nbrPtr*)

Get value of the local nbrData.

Parameters

nbrDataPtr	pointer to neighbor message
nbrPtr	neighbor pointer

Returns

if nbrPtr and nbrMsgPtr are valid, value; else, 0

7.12.2.7 uint8 nbrDataGetSize (NbrData * *nbrDataPtr*)

Get size of neighbor message.

As specified in NbrMsgCreate()

Parameters

nbrMsgPtr	pointer to neighbor message

Returns

if nbrMsgPtr is valid, size; else, 0

7.12.2.8 void nbrDataPrintHeaders (void)

Print the nmes of all the nbr data in a header suitable for data logging.

Parameters

nbrPtr	neighbor pointer

Returns

void

7.12.2.9 void nbrDataPrintNbr (Nbr * *nbrPtr*)

Print all the neighbor messages of a given neighbor.

Parameters

nbrPtr	neighbor pointer

Returns

void

7.12.2.10 void nbrDataPrintNbrVerbose (Nbr * *nbrPtr*)

Print all the neighbor messages of the given neighbor. Prints in verbose format, with names on separate lines.

Parameters

nbrPtr	neighbor pointer

Returns

void

7.12.2.11 void nbrDataSet (NbrData * *nbrDataPtr,* uint8 *value*)

Set the value of the neighbor data.

Parameters

nbrMsgPtr	the point to the nbrMsgPtr to be set. The size of the pointer must be 8 or smaller.
value	the value the nbrMsgPtr should be set to

Returns

void

7.13 IRComms/neighbors.c File Reference

used to maintain information about network neighbors, sets up data storage and callbacks

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "roneos.h"
#include "neighborsInternal.h"
```

Functions

- void neighborsDisable (void)

  Disable neighbor xmit/recv.
- void neighborsXmitEnable (boolean neighbor_xmit_enable_arg)

  Enable neighbor to transmit messages.
- void neighborsInit (uint32 neighbor_period_arg)

  Initialize neighbors and start neighbors task.
- void neighborsSetPeriod (uint32 neighbor_period_arg)

  Set neighbor period, neighbor timeout, and obstacle timeout proportional to argument.
- void neighborsSetTimeoutRounds (uint8 timeoutRounds, uint8 minActive, uint8 maxInactive)

  Set neighbor period, neighbor timeout, and obstacle timeout proportional to argument.
- void neighborsIgnore (uint8 neighborID)

  Tries add neighborID to list of neighbors to ignore.
- uint32 neighborsGetPeriod (void)

  Get neighbor period.
- void nbrPrint (Nbr *nbrPtr)

  Print information on neighbor (and information of neighbor's neighbors).
- void obstaclePrint (void)

  Print the obstacle data from the IR sensors.
- void nbrPrintData (Nbr *nbrPtr, uint32 round)

  Print header and neighbor data.
- uint8 irObstaclesGetBits (void)

  Get IR obstacle bits.
- uint8 * irObstaclesGetBitMatrix (void)

  Get IR obstacle bit matrix.
- int16 irObstaclesGetBearing (void)

  Get IR obstacle bearing.
- void neighborsGetMutex (void)

  Get neighbors mutex.
- void neighborsPutMutex (void)

Put neighbors mutex.

- void neighborsTask (void *parameters)

  The neighbor update system task.
- uint32 neighborsGetRound (void)

  Get neighbor round from neighbor data.
- boolean neighborsNewRoundCheck (uint32 *roundOldPtr)

  Check to see if there is a new neighbor round. Updates the variable at the pointer.
- boolean nbrIsBeacon (Nbr *nbrPtr)

  Returns true if this neighbor is a beacon.
- boolean nbrIsRobot (Nbr *nbrPtr)

  Returns true if this neighbor is a beacon.
- uint8 nbrGetID (Nbr *nbrPtr)

  Get neighbor ID.
- int32 nbrGetBearing (Nbr *nbrPtr)

  Get neighbor bearing.
- int32 nbrGetOrientation (Nbr *nbrPtr)

  Get neighbor orientation.
- int32 nbrGetRange (Nbr *nbrPtr)

  Get neighbor range.
- boolean nbrGetOrientationValid (Nbr *nbrPtr)

  Get neighbor orientation valid.
- uint8 nbrGetRangeBits (Nbr *nbrPtr)

  Get neighbor range bits.
- uint8 nbrGetReceiverBits (Nbr *nbrPtr)

  Get neighbor receiver bits.
- uint8 nbrGetTransmitterBits (Nbr *nbrPtr)

  Get neighbor transmitter bits.
- uint32 nbrGetUpdateTime (Nbr *nbrPtr)

  Get neighbor update time.
- uint32 nbrGetUpdateRound (Nbr *nbrPtr)

  Get neighbor update round.

7.13.1 Detailed Description

used to maintain information about network neighbors, sets up data storage and callbacks

Since

Mar 2, 2011

Author

: jamesm

7.14 IRComms/neighbors.h File Reference

Data Structures

- struct NbrData

  linked list of data on messages
- struct NbrMsgRadioNbrData

IR message a neighbor has.

- struct NbrMsgRadio

  linked list a neighbor's messages

- struct Nbr

  information stored on a network neighbor

- struct NbrDatabase

  Array containing data on all the network neighbors.

- struct NbrList

  Array containing data on all the network neighbors??

- struct IRRangeData

  Data from IR signal.

Typedefs

- typedef struct NbrData NbrData

  linked list of data on messages

- typedef struct NbrMsgRadioNbrData NbrMsgRadioNbrData

  IR message a neighbor has.

- typedef struct NbrMsgRadio NbrMsgRadio

  linked list a neighbor's messages

- typedef struct Nbr Nbr

  information stored on a network neighbor

- typedef struct NbrDatabase NbrDatabase

  Array containing data on all the network neighbors.

- typedef struct NbrList NbrList

  Array containing data on all the network neighbors??

- typedef struct IRRangeData IRRangeData

  Data from IR signal.

Functions

- void neighborsInit (uint32 neighbor_period)

  Initialize neighbors and start neighbors task.

- void neighborsDisable (void)

  Disable neighbor xmit/recv.

- void neighborsXmitEnable (boolean neighbor_xmit_enable_arg)

  Enable neighbor to transmit messages.

- void neighborsSetPeriod (uint32 neighbor_period_arg)

  Set neighbor period, neighbor timeout, and obstacle timeout proportional to argument.

- void neighborsSetTimeoutRounds (uint8 timeoutRounds, uint8 minActive, uint8 maxInactive)

  Set neighbor period, neighbor timeout, and obstacle timeout proportional to argument.

- uint32 neighborsGetPeriod (void)

  Get neighbor period.

- void neighborsGetMutex (void)

  Get neighbors mutex.

- void neighborsPutMutex (void)

  Put neighbors mutex.

- uint32 neighborsGetRound (void)

  Get neighbor round from neighbor data.

- boolean neighborsNewRoundCheck (uint32 *roundOldPtr)

Check to see if there is a new neighbor round. Updates the variable at the pointer.

- void neighborsIgnore (uint8 neighborID)

  Tries add neighborID to list of neighbors to ignore.

- uint8 irObstaclesGetBits (void)

  Get IR obstacle bits.

- uint8 * irObstaclesGetBitMatrix (void)

  Get IR obstacle bit matrix.

- int16 irObstaclesGetBearing (void)

  Get IR obstacle bearing.

- void obstaclePrint (void)

  Print the obstacle data from the IR sensors.

- uint8 nbrGetID (Nbr *nbrPtr)

  Get neighbor ID.

- int32 nbrGetBearing (Nbr *nbrPtr)

  Get neighbor bearing.

- int32 nbrGetOrientation (Nbr *nbrPtr)

  Get neighbor orientation.

- int32 nbrGetRange (Nbr *nbrPtr)

  Get neighbor range.

- boolean nbrGetOrientationValid (Nbr *nbrPtr)

  Get neighbor orientation valid.

- uint8 nbrGetRangeBits (Nbr *nbrPtr)

  Get neighbor range bits.

- uint8 nbrGetReceiverBits (Nbr *nbrPtr)

  Get neighbor receiver bits.

- uint8 nbrGetTransmitterBits (Nbr *nbrPtr)

  Get neighbor transmitter bits.

- uint32 nbrGetUpdateTime (Nbr *nbrPtr)

  Get neighbor update time.

- uint32 nbrGetUpdateRound (Nbr *nbrPtr)

  Get neighbor update round.

- void nbrPrint (Nbr *nbr)

  Print information on neighbor (and information of neighbor's neighbors).

- void nbrPrintData (Nbr *nbr, uint32 round)

  Print header and neighbor data.

- boolean nbrIsRobot (Nbr *nbrPtr)

  Returns true if this neighbor is a beacon.

- boolean nbrIsBeacon (Nbr *nbrPtr)

  Returns true if this neighbor is a beacon.

7.14.1 Detailed Description

Created on: Mar 2, 2011 Author: jamesm

7.15 Motors/encoder.c File Reference

functions for wheel encoders

```
#include <stdlib.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_qei.h"
#include "driverlib/gpio.h"
#include "driverlib/pwm.h"
#include "driverlib/qei.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "roneos.h"
```

Macros

- #define VEL_CAP_F 10

Functions

- void encoder_init (void)

  Initializes the encoder.
- int32 encoder_get_ticks (uint32 enc)

  Gets the current ticks of the specified encoder.
- int32 encoder_delta_ticks (uint32 new, uint32 old)

  Gets the difference between the two input ticks.
- int32 encoder_get_direction (uint32 enc)

  Gets the current rotating direction of the encoder.
- int32 encoder_get_velocity (uint32 enc)

  Gets the current velocity of the specified encoder.
- void encoder_pose_update (void)

  Updates the encoder's pose.
- void encoder_pose_clear (void)

  Clears the x, y, theta and odometer values of the encoder.
- void encoder_get_pose (Pose *posePtr)

  Gets the pose of the encoder and stores it in variables (x, y, theta) of where posePtr points to.
- uint32 encoderGetOdometer (void)

  returns the odometer reading in mm
- void encoder_set_pose (Pose *posePtr)

  Sets the pose of the encoder as variables of where posePtr points to.

7.15.1 Detailed Description

functions for wheel encoders

Since

Mar 2, 2011

Author

: jamesm

7.15.2 Macro Definition Documentation

7.15.2.1 #define VEL_CAP_F 10

Defines

7.15.3 Function Documentation

7.15.3.1 int32 encoder_delta_ticks (uint32 *new*, uint32 *old*)

Gets the difference between the two input ticks.

Parameters

new	is the new encoder position
old	is the old encoder position

Returns

the difference between old and new position with rollover protection

7.15.3.2 int32 encoder_get_direction (uint32 *enc*)

Gets the current rotating direction of the encoder.

Parameters

enc	specifies which encoder (right or left) to look up

Returns

the current rotating direction

7.15.3.3 void encoder_get_pose (Pose * *posePtr*)

Gets the pose of the encoder and stores it in variables (x, y, theta) of where posePtr points to.

Parameters

posePtr	points to a Pose structure

Returns

void

7.15.3.4 int32 encoder_get_ticks (uint32 *enc*)

Gets the current ticks of the specified encoder.

Ticks can then be converted to a measurement of distance.

Parameters

enc	specifies which encoder's ticks you want

Returns

the current position of the specified encoder; returns 0 if it's unavailable TODO: change all functions to camelCase

7.15.3.5 int32 encoder_get_velocity (uint32 *enc*)

Gets the current velocity of the specified encoder.

Parameters

enc	specifies which encoder (right or left) to look up

Returns

the current velocity

7.15.3.6 void encoder_init (void)

Initializes the encoder.

Enables the peripherals. Sets the state of the odometer to 0,0,0.

Returns

void

7.15.3.7 void encoder_pose_clear (void)

Clears the x, y, theta and odometer values of the encoder.

Returns

void

7.15.3.8 void encoder_pose_update (void)

Updates the encoder's pose.

There is an assumption that this function will be called on a regular basis in to ensure accuracy. Keep updating in micrometers. The encoders have a resolution of 0.0625mm. That is, each tick represents a change of 0.0625 mm. Our pose is stored in micrometers and microradians. Each tick corresponds to 62.5 micrometers. Rounding to 63.

Returns

void

7.15.3.9 void encoder_set_pose (Pose * *posePtr*)

Sets the pose of the encoder as variables of where posePtr points to.

Parameters

posePtr	points to a Pose structure

Returns

void

7.15.3.10 uint32 encoderGetOdometer (void)

returns the odometer reading in mm

Returns

void

7.16 Motors/motor.c File Reference

functions dealing with the two motors on the rone

```
#include <stdlib.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_qei.h"
#include "driverlib/gpio.h"
#include "driverlib/pwm.h"
#include "driverlib/qei.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "roneos.h"
```

Data Structures

- struct motorVelocityData

  contains information on the robot's state used for smooth motor control

Typedefs

- typedef struct motorVelocityData motorVelocityData

  contains information on the robot's state used for smooth motor control

Functions

- void motorInit ()

  Initializes motor.
- void motorSetPWM (uint32 motor, int32 dutyCycle)

  Sets PWM duty cycle for the specified motor if remote control mode is off.
- void motorBrake (uint32 motor)

  Brakes one motor with the specified duty cycle.
- void motorTimeoutStop (void)

  Turns off the motors at a low level/.
- void motorCommandTimerUpdate (void)

  Updates the motor command timer.
- void motorSetVelocity_NonCmd (uint32 motor, int32 velocity)

Sets the current velocity of the specified motor.

- void motorSetVelocity (uint32 motor, int32 velocity)

  Sets the velocity of the specified motor if remote control mode is off.
- int32 motorGetVelocity (uint32 motor)

  Gets the current velocity of the specified motor.
- void motorSetTVRV_NonCmd (int32 tv, int32 rv)

  Sets the translational and radial velocity of the motor.
- void motorSetTVRV (int32 tv, int32 rv)

  Sets the translation and radial velocity of the motors if remote control mode is off.
- void motorGetTVRV (int32 *tvPtr, int32 *rvPtr)

  Gets the translational and radial velocity of the motor.
- boolean waypointMoveDone (void)

  Returns whether current waypoint has been reached.
- void waypointMove (Pose *posePtr, int32 speed)

  MotorCommandMode set so that waypoint behavior with absolute coordinates is done.
- void waypointMoveRelative (Pose *posePtr, int32 speed)

  MotorCommandMode set so that waypoint behavior with relative coordinates is done.
- void waypointMoveTheta (Pose *posePtr, int32 speed)

  MotorCommandMode set so that waypoint behavior with absolute coordinates is done.
- void waypointMoveThetaRelative (Pose *posePtr, int32 speed)

  MotorCommandMode set so that waypoint behavior with relative coordinates is done.
- void motorVelocityUpdate (void)

  Updates the velocity data for both motors.

7.16.1 Detailed Description

functions dealing with the two motors on the rone

Since

Mar 2, 2011

Author

: jamesm

7.16.2 Function Documentation

7.16.2.1 void motorBrake (uint32 *motor*)

Brakes one motor with the specified duty cycle.

Parameters

motor	(left or right)
dutyCycle	duty cycle of PWM

7.16.2.2 void motorCommandTimerUpdate (void)

Updates the motor command timer.

If no motor command has been received, timeout and enables the charger. This function should be called at 10 hz.

Returns

void

7.16.2.3 void motorGetTVRV (int32 * *tvPtr,* int32 * *rvPtr*)

Gets the translational and radial velocity of the motor.

Parameters

tvPtr	pointer to the desired translational velocity
rvPtr	pointer to the desired rotational velocity

Returns

void

7.16.2.4 int32 motorGetVelocity (uint32 *motor*)

Gets the current velocity of the specified motor.

Parameters

motor	left or right motor

Returns

the current velocity; 0 if the input parameter is not recognized

7.16.2.5 void motorInit (void)

Initializes motor.

Enables PWM and initializes the motorVelocityData struct for both left and right motor. Also sets the command timer to 0

Returns

void

7.16.2.6 void motorSetPWM (uint32 *motor,* int32 *dutyCycle*)

Sets PWM duty cycle for the specified motor if remote control mode is off.

Sets PWM duty cycle for the specified motor for both reverse and forward signals.

Parameters

motor	left or right motor
dutyCycle	duty cycle of PWM

Returns

void

7.16.2.7 void motorSetTVRV (int32 *tv,* int32 *rv*)

Sets the translation and radial velocity of the motors if remote control mode is off.

Parameters

tv	the translational velocity
rv	the rotational velocity

Returns

void

7.16.2.8 void motorSetTVRV_NonCmd (int32 *tv,* int32 *rv*)

Sets the translational and radial velocity of the motor.

Returns

void

7.16.2.9 void motorSetVelocity (uint32 *motor,* int32 *velocity*)

Sets the velocity of the specified motor if remote control mode is off.

Parameters

motor	left or right motor
velocity	motor velocity to be set in mm/s

Returns

void

7.16.2.10 void motorSetVelocity_NonCmd (uint32 *motor,* int32 *velocity*)

Sets the current velocity of the specified motor.

Parameters

motor	left or right motor
velocity	motor velocity to be set in mm/s

Returns

void

7.16.2.11 void motorTimeoutStop (void)

Turns off the motors at a low level/.

Returns

void

7.16.2.12 void motorVelocityUpdate (void)

Updates the velocity data for both motors.

Returns

void

7.16.2.13 void waypointMove (Pose * *posePtr,* int32 *speed*)

MotorCommandMode set so that waypoint behavior with absolute coordinates is done.

Returns

void

7.16.2.14 boolean waypointMoveDone (void)

Returns whether current waypoint has been reached.

Returns

waypointDone boolean that keeps track of whether the current waypoint has been reached or not

7.16.2.15 void waypointMoveRelative (Pose * *posePtr,* int32 *speed*)

MotorCommandMode set so that waypoint behavior with relative coordinates is done.

Returns

void

7.16.2.16 void waypointMoveTheta (Pose * *posePtr,* int32 *speed*)

MotorCommandMode set so that waypoint behavior with absolute coordinates is done.

Returns

void

7.16.2.17 void waypointMoveThetaRelative (Pose * *posePtr,* int32 *speed*)

MotorCommandMode set so that waypoint behavior with relative coordinates is done.

Returns

void

7.17 Sensors/accelerometer.c File Reference

interface functions for 3D accelerometer in the robot

```
#include "inc/lm3s8962.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/ssi.h"
#include "roneos.h"
```

7.17.1 Detailed Description

interface functions for 3D accelerometer in the robot

Since

Jul 22, 2010

Author

jamesm

7.18 Sensors/accelerometer.h File Reference

Functions

- int16 accelerometerGetValue (uint32 axis)

  Gets the accelerometer value.

7.18.1 Detailed Description

Created on: Mar 19, 2011 Author: jamesm

7.18.2 Function Documentation

7.18.2.1 int16 accelerometerGetValue (uint32 *axis*)

Gets the accelerometer value.

Parameters

axis	uint32 of the axis to get the value of.

Returns

int16 value of accelerometer.

7.19 Sensors/bump_sensor.c File Reference

reads bump sensor information, helper functions for this data

```
#include "roneos.h"
#include "bump_sensor.h"
```

Functions

- uint8 bumpSensorsGetBits ()

  Get bump sensor bits.
- int16 bumpSensorsGetBearing ()

  Get bump sensor bearing.

7.19.1 Detailed Description

reads bump sensor information, helper functions for this data

Since

Jul 22, 2010

Author

jamesm

7.19.2 Function Documentation

7.19.2.1 int16 bumpSensorsGetBearing ( )

Get bump sensor bearing.

Returns

the bump sensor bearing

7.19.2.2 uint8 bumpSensorsGetBits ( )

Get bump sensor bits.

Returns

the bump sensor bits

7.20 Sensors/gyro.c File Reference

interface to initialize and read 3D gyro

```
#include "inc/lm3s8962.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_qei.h"
#include "driverlib/gpio.h"
#include "driverlib/adc.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "roneos.h"
```

7.20.1 Detailed Description

interface to initialize and read 3D gyro

Since

Jul 22, 2010

Author

jamesm

7.21 Sensors/light_sensor.c File Reference

interface to initialize and read light sensor ring on robot

```
#include "inc/lm3s8962.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_qei.h"
#include "driverlib/gpio.h"
#include "driverlib/adc.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "roneos.h"
#include "lightSensorCalibration.h"
```

Functions

- void light_sensor_init (void)

  Initializes light sensor.
- int32 light_sensor_get_value (uint32 light_sensor)

  Gets the ADC value of the specified light sensor.

7.21.1 Detailed Description

interface to initialize and read light sensor ring on robot

Since

Jul 22, 2010

Author

jamesm

7.21.2 Function Documentation

7.21.2.1 int32 light_sensor_get_value (uint32 *light_sensor*)

Gets the ADC value of the specified light sensor.

If the input light sensor is not recognized, returns 0

Parameters

light_sensor	specifies which light sensor you want to check. Only {0, 1, 2} are allowed on v11 robots and {0, 1, 2, 3} allowed on v12 robots

Returns

the ADC value; 0 if the input parameter is not recognized. Note that for calibrated robots, this value can be negative, i.e. less that the calibration conditions.

7.21.2.2 void light_sensor_init (void)

Initializes light sensor.

Sets sampling speed and configures and enables the light sensor.

Returns

void

7.22 SerialIO/basicPrinting.c File Reference

print methods for converting data into strings with different formats

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

Functions

- char * bitString8 (char *string, uint8 val)

  Function that allows us to print binary.
- void posePrint (Pose *posePtr)

  Print a pose structure.
- uint8 ctoi_hex4 (char c)

  Convert single character to an integer.
- uint8 atoi_hex8 (char *string)

  Convert 8-bit hex string to an integer.
- uint16 atoi_hex16 (char *string)

Convert 16-bit hex string to an integer.

- uint32 atoi_hex32 (char *string)

  Convert 32-bit hex string to an integer.

7.22.1 Detailed Description

print methods for converting data into strings with different formats

Since

Mar 24, 2012

Author

: jamesm

7.22.2 Function Documentation

7.22.2.1 uint16 atoi_hex16 (char * *string*)

Convert 16-bit hex string to an integer.

Parameters

string	16-bit hex string to be converted

Returns

hex integer version of input

7.22.2.2 uint32 atoi_hex32 (char * *string*)

Convert 32-bit hex string to an integer.

Parameters

string	32-bit hex string to be converted

Returns

hex integer version of input

7.22.2.3 uint8 atoi_hex8 (char * *string*)

Convert 8-bit hex string to an integer.

Parameters

string	8-bit hex string to be converted

Returns

hex integer version of input

7.22.2.4 char* bitString8 (char * *string,* uint8 *val*)

Function that allows us to print binary.

Creates a binary version of the input character in the space given by string.

Parameters

**string*	is the char pointer that will point to the binary bitstring of val
val	is the value to be converted to binary

Returns

a pointer to the binary string converted from val

7.22.2.5 uint8 ctoi_hex4 (char *c*)

Convert single character to an integer.

Converts a single character that represents a unicode number (hex) to a unicode number(integer)

Parameters

c	is the character to be converted

Returns

val is the integer value

7.22.2.6 void posePrint (Pose * *posePtr*)

Print a pose structure.

Print a pose structure. Prints in braces to be fancy.

Parameters

posePtr	is a pointer to a pose.

7.23 SerialIO/serial.c File Reference

serial UART communication functions

```
#include <stdio.h>
#include "inc/lm3s8962.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/ssi.h"
#include "driverlib/interrupt.h"
#include "roneos.h"
```

Functions

- void sputchar (char c)

  Sends out a character. (DEPRICATED - has hard to solve race condition problems, and is not used by cprintf)
- void sputcharFlush (void)

  Flushes the buffer and starts the serial xmit.
- int sgetchar (void)

  Gets a character from serial receive buffer.
- void uartIRQHandler (void)

  Interrupt that handles bytes coming through serial port.
- void serial_init ()

  Initializes serial I/O.

7.23.1 Detailed Description

serial UART communication functions

Since

Jul 21, 2010

Author

sjb2

7.23.2 Function Documentation

7.23.2.1 void serial_init (void)

Initializes serial I/O.

Enable the peripherals used by this example. Enable processor interrupts. Set GPIO A0 and A1 as UART pins. Configure the UART for 115,200, 8-N-1 operation. Enable the UART interrupt.

Returns

void

7.23.2.2 int sgetchar (void)

Gets a character from serial receive buffer.

Returns

first character from serial receive buffer.

7.23.2.3 void sputchar (char *c*)

Sends out a character. (DEPRICATED - has hard to solve race condition problems, and is not used by cprintf)

Sends the character c to the transmit FIFO for the port specified by UARTO_BASE (base address).

Parameters

c	is the character to be transmitted

Returns

void Buffers a character on the serial output buffer.

Sends the character c to the transmit FIFO for the port specified by UARTO_BASE (base address).

Parameters

c	is the character to be transmitted

Returns

void

7.23.2.4 void sputcharFlush (void)

Flushes the buffer and starts the serial xmit.

loads the UART fifo from the RAM buffer. This starts a xmit.

Returns

void

7.23.2.5 void uartIRQHandler (void)

Interrupt that handles bytes coming through serial port.

Returns

void

7.24 SerialIO/serialCommand.c File Reference

processes serial commands and links them with desired function

```
#include <string.h>
#include "roneos.h"
```

Functions

- uint32 serialCommandGetTimestamp (SerialCmd *serialCmdPtr)

  Get the time this command was last called.
- void serialCommandAdd (SerialCmd *serialCmdPtr, char *name, void(*funcPtr)(char *message))

Add serial command to linked list.

- void serialCommandInit ()

  Initializes serial command processing.

7.24.1 Detailed Description

processes serial commands and links them with desired function

Since

Mar 17, 2012

Author

Sunny Kim

Warning

only partially commented

7.24.2 Function Documentation

7.24.2.1 void serialCommandAdd (SerialCmd * *serialCmdPtr,* char * *name,* void(*)(char *message) *funcPtr*)

Add serial command to linked list.

Parameters

serialCmdPtr	pointer to serial command to be added
name	name of serial command
funcPtr	function pointer to function that will be executed when command is sent to serial port

Returns

void

7.24.2.2 uint32 serialCommandGetTimestamp (SerialCmd * *serialCmdPtr*)

Get the time this command was last called.

Parameters

serialCmdPtr	pointer to serial command to be added

Returns

tick time of last call

7.24.2.3 void serialCommandInit (void)

Initializes serial command processing.

Returns

void

7.25 SerialIO/systemCommands.c File Reference

parses char strings that are system commands

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "roneos.h"
```

Data Structures

- struct guiCmdData
  Commands from the GUI.

Functions

- boolean irCommsSendMessage_internal (irCommsMessage *irMessagePtr)
  TODO:
- boolean irCommsGetMessage_internal (irCommsMessage *irMessagePtr)
  TODO:
- void systemCommandsInit ()
  Initialize system commands.

7.25.1 Detailed Description

parses char strings that are system commands

Since

Apr 2, 2012

Author

jamesm

7.25.2 Function Documentation

7.25.2.1 boolean irCommsGetMessage_internal (irCommsMessage * *irMessagePtr*)

TODO:

Parameters

irMessagePtr	message to transmit

Returns

TRUE if _____

7.25.2.2 boolean irCommsSendMessage_internal (irCommsMessage * *irMessagePtr*)

TODO:

Parameters

irMessagePtr	message to transmit

Returns

TRUE if _____

7.25.2.3 void systemCommandsInit (void)

Initialize system commands.

Returns

void

7.26 System/charger.c File Reference

initializes/enables/disables battery charger

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "roneos.h"
```

7.26.1 Detailed Description

initializes/enables/disables battery charger

Since

Apr 2, 2012

Author

jamesm

7.27 System/intMath.c File Reference

fast integer math (no floating point processor on rone)

```
#include <math.h>
#include <stdlib.h>
#include "roneos.h"
#include "intMathTrigLookup.c"
```

Functions

- uint32 decToZero (uint32 val)

  Continually decrements the input value by one until it is closest to zero.
- int32 average (int32 val1, int32 val2)

  Average two values.
- uint32 circularInc (uint32 index, uint32 maxIndex)

  Circularly increments the index by 1.
- uint32 circularDec (uint32 index, uint32 maxIndex)

  Circularly decrements the index by 1.
- uint32 sqrtInt (uint32 val)

  Compute the integer square root of a number.
- void pack32 (uint8 *arrayPtr, uint32 dataWord)

  Pack a 32-bit dataWord into 8-bit, pointed to by char pointer arrayPtr.
- void pack24 (uint8 *arrayPtr, uint32 dataWord)

  Pack a 24-bit dataWord into 8-bit, pointed to by char pointer arrayPtr.
- void pack16 (uint8 *arrayPtr, uint32 dataWord)

  Pack a 16-bit dataWord into 8-bit, pointed to by pointer arrayPtr.
- uint16 unpack16 (uint8 *arrayPtr)

  Unpacks an 8-bit data into 16-bit.
- uint32 unpack24 (uint8 *arrayPtr)

  Unpacks an 8-bit data into 24-bit.
- uint32 unpack32 (uint8 *arrayPtr)

  Unpacks an 8-bit data into 32 bit. Implemented in a pedantic way to avoid assumptions of endianness.
- int16 normalizeAngleMilliRad (int16 angle)

  Normalizes the angle.
- int16 normalizeAngleMilliRad2 (int16 angle)

  Normalizes the angle.
- int32 normalizeAngleMicroRad (int32 angle)

  Normalizes the angle.
- int16 sinMilliRad (int16 angle)

  Interprets the angle as milli-radian of sine.
- int16 cosMilliRad (int16 angle)
- int16 smallestAngleDifference (int16 thetaGoal, int16 theta)

  Calculates the smallest angle difference between the two input angles.
- int32 poseAngleDiff (Pose *poseGoalPtr, Pose *posePtr)

  Calculates the smallestAngleDifference between two poses.
- void poseAdd (Pose *poseResPtr, Pose *pose1Ptr, Pose *pose2Ptr)

  Adds two Poses and places result in a Pose.
- int32 poseDistance (Pose *pose1Ptr, Pose *pose2Ptr)

  Calculates distance between two poses.
- int32 boundAbs (int32 val, int32 bound)

  Bounds the value with one specified bound as both lower and upper bound.
- int32 min (int32 x, int32 y)

  Finds the min of the two arguments.

- int32 max (int32 x, int32 y)

  Finds the min of the two arguments.
- int32 bound (int32 val, int32 lowerBound, int32 upperBound)

  Bounds the value with specified lower and upper bound.
- uint8 bitsCount (uint32 val)

  Counts how many bits the value has.
- int32 filterIIR (int32 currentVal, int32 newVal, int32 alpha)

  IIR (low pass) filter for integer quantities.
- int16 atan2MilliRad (int32 y, int32 x)

  Gets atan2 approximation in miiliradians.
- int32 averageAnglesMicroRad (int32 theta1, int32 theta2)

  Calculates the average of the two angles in microrad.
- int16 averageAngles (int16 angle1, int16 angle2)

  Calculates the average of the two angles in millirad.
- int16 averageAnglesLeftToRight (int16 angleLeft, int16 angleRight)

  Calculates the average of the two angles in millirad.
- int16 averageArrayAngle (int16 angleArray[], int32 size)

  Calculates the average of the angles in the array.
- int16 byteToMillirad (int8 angle)

  unit conversion (16bit to 8bit)
- int8 milliradToByte (int16 angle)

  unit conversion (16bit to 8bit)
- int16 byteToMilliradUnsigned (uint8 angle)

  unit conversion (16bit to 8bit)
- uint8 milliradToByteUnsigned (int16 angle)

  unit conversion (16bit to 8bit)
- int16 angleFromBitVector (uint8 bitVector)

  Calculates the resultant angle from the bit vector. This assumes that bit0 = 0 rad.
- int16 angleFromBitVectorOffset (uint8 bitVector)

  Calculates the resultant angle from the bit vector. This assumes that bit0 = 383 rad.

7.27.1 Detailed Description

fast integer math (no floating point processor on rone)

Since

Apr 2, 2012

Author

jamesm

7.27.2 Function Documentation

7.27.2.1 int16 angleFromBitVector (uint8 *bitVector*)

Calculates the resultant angle from the bit vector. This assumes that bit0 = 0 rad.

Parameters

bitVector	is the vector of bits

Returns

the average angle

7.27.2.2 int16 angleFromBitVectorOffset (uint8 *bitVector*)

Calculates the resultant angle from the bit vector. This assumes that bit0 = 383 rad.

Parameters

bitVector	is the vector of bits

Returns

the average angle

7.27.2.3 int16 atan2MilliRad (int32 *y*, int32 *x*)

Gets atan2 approximation in miiliradians.

Originally developed by John Aspinal at iRobot. It is quite good.

Parameters

y	y-coordinate of the point to be calculated
x	x-coordinate of the point to be calculated

Returns

atan2 approximation of the input point, specified by (x,y) coordinate

7.27.2.4 int32 average (int32 *val1*, int32 *val2*)

Average two values.

Parameters

val1	is the first value
val2	is the second value

Returns

the average of val1 and val2

7.27.2.5 int16 averageAngles (int16 *angle1*, int16 *angle2*)

Calculates the average of the two angles in millirad.

Parameters

angle1	is the first angle to be averaged
angle2	is the second angle to be averaged

Returns

the average angle

7.27.2.6 int16 averageAnglesLeftToRight (int16 *angleLeft,* int16 *angleRight*)

Calculates the average of the two angles in millirad.

Parameters

angle1	is the first angle to be averaged
angle2	is the second angle to be averaged

Returns

the average angle

7.27.2.7 int32 averageAnglesMicroRad (int32 *theta1,* int32 *theta2*)

Calculates the average of the two angles in microrad.

Parameters

theta1	is the first angle to be averaged
theta2	is the second angle to be averaged

Returns

the average angle

7.27.2.8 int16 averageArrayAngle (int16 *angleArray[],* int32 *size*)

Calculates the average of the angles in the array.

Calculates the average of the first "size (a number)" of angles in angleArray.

Parameters

angleArray[]	is the array of angles to be averaged
size	specifies how many elements in the array (starting from the first) should be averaged

Returns

the average of the angles in the array (returns 0 if given a nonpositive size)

7.27.2.9 uint8 bitsCount (uint32 *val*)

Counts how many bits the value has.

Ignores leading zeros.

Parameters

val	is the value to be counted

Returns

the number of bits of the input value

7.27.2.10 int32 bound (int32 *val,* int32 *lowerBound,* int32 *upperBound*)

Bounds the value with specified lower and upper bound.

Bounds the value so that it stays within the range of lowerBound <= value <= upperBound. If it exceeds the bound, set it to the lower/upper bound.

Parameters

val	is the value to be bounded
lowerBound	is the lower bound
upperBound	is the upper bound

Returns

the bounded value

7.27.2.11 int32 boundAbs (int32 *val,* int32 *bound*)

Bounds the value with one specified bound as both lower and upper bound.

Bounds the input value so that it stays within the range of -bound <= value <= bound. If it exceeds the bound, set it to the bound.

Parameters

val	is the value to be bounded
bound	is the lower and upper bound

Returns

the bounded value

7.27.2.12 int16 byteToMillirad (int8 *angle*)

unit conversion (16bit to 8bit)

Parameters

int8	the angle as a Byte

Returns

int16 angle in milli-radians

7.27.2.13 int16 byteToMilliradUnsigned (uint8 *angle*)

unit conversion (16bit to 8bit)

Parameters

int8	the angle as a Byte

Returns

int16 angle in milli-radians

7.27.2.14 uint32 circularDec (uint32 *index,* uint32 *maxIndex*)

Circularly decrements the index by 1.

Decrements the index by 1. If the index reaches 0, resets it to maximum index. Circular meaning it goes back to maximum index.

Parameters

index	the index to be decremented
maxIndex	the maximum index

Returns

the decremented circular index

7.27.2.15 uint32 circularInc (uint32 *index,* uint32 *maxIndex*)

Circularly increments the index by 1.

Increments the index by 1. Resets index to 0 if it exceeds the maximum index Circular meaning it goes back to 0.

Parameters

index	is the index to be incremented
maxIndex	is the maximum index

Returns

the incremented circular index

7.27.2.16 int16 cosMilliRad (int16 *angle*)

Interprets the angle as milli-randian of cosine.

Parameters

angle	the angle to be interpreted

Returns

angle as milli-radian of cosine

7.27.2.17 uint32 decToZero (uint32 *val*)

Continually decrements the input value by one until it is closest to zero.

Parameters

val	the value to be decremented

Returns

the decremented value (within the range of 0 <= val < 1)

7.27.2.18 int32 filterIIR (int32 *sample,* int32 *average,* int32 *alpha*)

IIR (low pass) filter for integer quantities.

runs an IIR filter for integer quantities.

Parameters

current	value
sample	new measurement
alpha	IIR constant. This is divided by 100, so 100 = all newVal and 0 = all currentVal

Returns

filtered value

7.27.2.19 int32 max (int32 *x,* int32 *y*)

Finds the min of the two arguments.

Finds the min of two arguments.

Parameters

x,y	is the value to be compared

Returns

the min value

7.27.2.20 int8 milliradToByte (int16 *angle*)

unit conversion (16bit to 8bit)

Parameters

int16	angle

Returns

int8 the angle as a Byte

7.27.2.21 uint8 milliradToByteUnsigned (int16 *angle*)

unit conversion (16bit to 8bit)

Parameters

int16	angle

Returns

int8 the angle as a Byte

7.27.2.22 int32 min (int32 *x*, int32 *y*)

Finds the min of the two arguments.

Finds the min of two arguments.

Parameters

x,y	is the value to be compared

Returns

the min value

7.27.2.23 int32 normalizeAngleMicroRad (int32 *angle*)

Normalizes the angle.

Normalizes the angle to make it stay in the range of 0 <= angle < millirad_PI.

Parameters

angle	the angle to be normalized

Returns

the normalized angle Normalizes the angle.

Normalizes the angle to make it stay in the range of 0 <= angle < microrad_2PI.

Parameters

angle	the angle to be normalized

Returns

the normalized angle

7.27.2.24 int16 normalizeAngleMilliRad (int16 *angle*)

Normalizes the angle.

Normalizes the angle to make it stay in the range of 0 <= angle < millirad_2PI

Parameters

angle	the angle to be normalized

Returns

the normalized angle

7.27.2.25 int16 normalizeAngleMilliRad2 (int16 *angle*)

Normalizes the angle.

Normalizes the angle to make it stay in the range of -millirad_PI $<$ angle $<=$ millirad_PI.

Parameters

angle	the angle to be normalized

Returns

the normalized angle

7.27.2.26 void pack16 (uint8 * *arrayPtr,* uint32 *dataWord*)

Pack a 16-bit dataWord into 8-bit, pointed to by pointer arrayPtr.

Parameters

arrayPtr	points to the packed 8-bit dataWord
dataWord	16-bit data to be packed

Returns

void

7.27.2.27 void pack24 (uint8 * *arrayPtr,* uint32 *dataWord*)

Pack a 24-bit dataWord into 8-bit, pointed to by char pointer arrayPtr.

Parameters

arrayPtr	points to the packed 8-bit dataWord
dataWord	24-bit data to be packed

Returns

void

7.27.2.28 void pack32 (uint8 * *arrayPtr,* uint32 *dataWord*)

Pack a 32-bit dataWord into 8-bit, pointed to by char pointer arrayPtr.

Parameters

arrayPtr	points to the packed 8-bit dataWord
dataWord	32-bit data to be packed

Returns

void

7.27.2.29 void poseAdd (Pose * *poseResPtr*, Pose * *pose1Ptr*, Pose * *pose2Ptr*)

Adds two Poses and places result in a Pose.

Parameters

poseResPtr	pointer to Pose to hold result
pose1Ptr	pointer to first Pose
pose2Ptr	pointer to second Pose

Returns

void

7.27.2.30 int32 poseAngleDiff (Pose * *poseGoalPtr*, Pose * *posePtr*)

Calculates the smallestAngleDifference between two poses.

Parameters

poseGoalPtr	pointer to goal pose
posePtr	pointer to a pose

Returns

smallest angle difference between two poses

7.27.2.31 int32 poseDistance (Pose * *pose1Ptr*, Pose * *pose2Ptr*)

Calculates distance between two poses.

Parameters

pose1Ptr	pointer to first Pose
pose2Ptr	pointer to second Pose

Returns

distance

7.27.2.32 int16 sinMilliRad (int16 *angle*)

Interprets the angle as milli-radian of sine.

Parameters

angle	the angle to be interpreted

Returns

if angle is greater than pi/4, angle as milli-radian of sine. else, 0.

7.27.2.33 int16 smallestAngleDifference (int16 *thetaGoal*, int16 *theta*)

Calculates the smallest angle difference between the two input angles.

The difference will be within the range of -MILLIRAD_PI <- difference <= MILLIRAD_PI.

Parameters

thetaGoal	is first angle
theta	is second angle

Returns

the difference between thetaGoal and theta

7.27.2.34 uint32 sqrtInt (uint32 *val*)

Compute the integer square root of a number.

Based on Microchip app note TB040. Can't take the root of numbers higher than MAX_INT32.

Parameters

val	is the number to be computed

Returns

the computed integer square root

7.27.2.35 uint16 unpack16 (uint8 * *arrayPtr*)

Unpacks an 8-bit data into 16-bit.

Parameters

arrayPtr	points to data with 8-bit wordlength

Returns

unpacked input data with 16-bit wordlength

7.27.2.36 uint32 unpack24 (uint8 * *arrayPtr*)

Unpacks an 8-bit data into 24-bit.

Parameters

arrayPtr	points to data with 8-bit wordlength

Returns

unpacked input data with 24-bit wordlength

7.27.2.37 uint32 unpack32 (uint8 * *arrayPtr*)

Unpacks an 8-bit data into 32 bit. Implemented in a pedantic way to avoid assumptions of endianness.

Parameters

arrayPtr	points to data with 8-bit wordlength

Returns

unpacked input data with 32-bit wordlength

7.28 System/msp430Bootloader.c File Reference

boot loader functions on MSP430

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "inc/hw_nvic.h"
#include "inc/hw_sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/systick.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/ssi.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"
#include "driverlib/flash.h"
#include "roneos.h"
```

7.28.1 Detailed Description

boot loader functions on MSP430

Since

Jul 31, 2012

Author

mrdouglass

7.29 System/pwm.c File Reference

This is a PWM module which was originally created for the IR beacon. It is meant to control some PWM setup, but mostly for setting and changing PWM on the 8962 pins. PWM outputs are used for things like single LEDs and the power adjustment on the IR beacons.

```
#include "inc/lm3s8962.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/pwm.h"
#include "roneos.h"
```

7.29.1 Detailed Description

This is a PWM module which was originally created for the IR beacon. It is meant to control some PWM setup, but mostly for setting and changing PWM on the 8962 pins. PWM outputs are used for things like single LEDs and the power adjustment on the IR beacons.

Since

Jun 1, 2012

Author

Lindsay

7.30 System/spi_message.c File Reference

SPI commands for the MSP430.

```
#include <string.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "inc/hw_nvic.h"
#include "inc/hw_sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/systick.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/ssi.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"
#include "driverlib/flash.h"
#include "roneos.h"
```

7.30.1 Detailed Description

SPI commands for the MSP430.

Since

Apr 2, 2012

Author

jamesm

7.31 System/system.c File Reference

System-level code: initialize and shutdown the robot, monitor power, set delays, etc.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "inc/hw_nvic.h"
#include "inc/hw_sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/systick.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/ssi.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"
#include "driverlib/flash.h"
#include "roneos.h"
```

Data Structures

- struct errorMsg

  Error message includes information to track error.
- struct warningMessage

  Warning message includes information to track warning.

Typedefs

- typedef struct errorMsg errorMsg

  Error message includes information to track error.
- typedef struct warningMessage warningMessage

  Warning message includes information to track warning.

Functions

- void radioIntHandler (void)

  Handles radio interrupt.
- void uartIRQHandler (void)

  Interrupt that handles bytes coming through serial port.
- void irCommsHandler (void)

  Handles interrupt requests generated by the IR comms.
- void systemHeartbeatTask (void ∗parameters)

Background tasks performed during each heartbeat.

- void systemInit (void)

  Initializes the r-one hardware.
- char * sysGetFilenameFromPath (char *filepathString)

  Gets the file name from path.
- void systemIDInit (void)

  Gets the robot ID.
- void systemPrintMemUsage (void)

  Print the heap and stack usage.

7.31.1 Detailed Description

System-level code: initialize and shutdown the robot, monitor power, set delays, etc.

Since

Jul 26, 2010

7.31.2 Function Documentation

7.31.2.1 void irCommsHandler (void)

Handles interrupt requests generated by the IR comms.

Clears IRQ line for next interrupt, transmit, receive, return to idle mode after message processed.

Returns

void

7.31.2.2 void radioIntHandler (void)

Handles radio interrupt.

Returns

void

7.31.2.3 char* sysGetFilenameFromPath (char * *filepathString*)

Gets the file name from path.

Parameters

filepathString	the file path

Returns

a pointer that points to the file name

7.31.2.4 void systemHeartbeatTask (void * *parameters*)

Background tasks performed during each heartbeat.

Each heartbeat = every 16 milliseconds. Updates blinky, IRBeacon, leds, accelerometer, motor velocity, pose, and motor command timer.

Returns

void

7.31.2.5 void systemIDInit (void)

Gets the robot ID.

roneID is assigned robot ID. If robot is uninitialized, roneID is DEFAULT_RONEID. The ID is set using LM Flash. the format for the ID is: 00-50-C2-00-01-XX where XX is the robotID

Returns

void

7.31.2.6 void systemInit (void)

Initializes the r-one hardware.

Initalizes roneID, charger, blinky, buttons, IRBeacon, SPI, LED, and serial. the heartbeat light blinks three times after the initializations are done. Initializes encoder, light sensor, motor, gyro, accelerometer, IR_comms, radio, ad cfprintf, sd card. Prints out the date, time, and roneID after everything is initialized. Rone starts heartbeat after this initialization.

Returns

void

7.31.2.7 void systemPrintMemUsage (void)

Print the heap and stack usage.

Returns

void

7.31.2.8 void uartIRQHandler (void)

Interrupt that handles bytes coming through serial port.

Returns

void

7.32 System/system.h File Reference

Functions

- void systemInit (void)

  Initializes the r-one hardware.
- void systemHeartbeatTask (void *parameters)

Background tasks performed during each heartbeat.

- void systemPrintMemUsage (void)

  Print the heap and stack usage.

- uint32 systemUSBConnected (void)

  Check USBlevel.

- void systemIDInit (void)

  Gets the robot ID.

7.32.1 Detailed Description

Since

Mar 26, 2011

Author

jamesm

7.32.2 Function Documentation

7.32.2.1 void systemHeartbeatTask (void * *parameters*)

Background tasks performed during each heartbeat.

Each heartbeat = every 16 milliseconds. Updates blinky, IRBeacon, leds, accelerometer, motor velocity, pose, and motor command timer.

Returns

void

7.32.2.2 void systemIDInit (void)

Gets the robot ID.

roneID is assigned robot ID. If robot is uninitialized, roneID is DEFAULT_RONEID. The ID is set using LM Flash. the format for the ID is: 00-50-C2-00-01-XX where XX is the robotID

Returns

void

7.32.2.3 void systemInit (void)

Initializes the r-one hardware.

Initalizes roneID, charger, blinky, buttons, IRBeacon, SPI, LED, and serial. the heartbeat light blinks three times after the initializations are done. Initializes encoder, light sensor, motor, gyro, accelerometer, IR_comms, radio, ad cfprintf, sd card. Prints out the date, time, and roneID after everything is initialized. Rone starts heartbeat after this initialization.

Returns

void

7.32.2.4 void systemPrintMemUsage (void)

Print the heap and stack usage.

Returns

void

7.32.2.5 uint32 systemUSBConnected (void)

Check USBlevel.

Unfinished, and returns FALSE unconditionally.

Returns

FALSE

Index