SensorControl Reference Manual

Generated by Doxygen 1.4.6

Wed Oct 24 12:57:23 2007

Contents

Chapter 1

Sensor Control

Firmware to sample all sensors located on the roboter hand

(c) 2007 by Matthias Arndt <marndt@asmsoftware.de>

Chapter 2

SensorControl File Index

2.1 SensorControl File List

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

Chapter 3

SensorControl File Documentation

3.1 adc.c File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
#include "adc.h"
```

Functions

- void ADC_init ()
  initialize A/D hardware
- WORD ADC (BYTE channel)
  execute A/D conversion for specific input channel

3.1.1 Detailed Description

Definition in file adc.c.

3.1.2 Function Documentation

3.1.2.1 WORD ADC (BYTE *channel*)

execute A/D conversion for specific input channel

Reads the voltage from the specified input source and does a standard conversion.

A reading of 0x0000 implies +0V while 0x03ff implies +5V

Parameters:
channel 0-7 the A/D channel we want to measure (corresponds to a pin number)

Returns:
conversion result as a 16Bit value

Definition at line 29 of file adc.c.

Referenced by Bumper_read(), Fingersensors_read(), Photosensor_read(), and Sharp_read().

3.1.2.2 void ADC_init (void)

initialize A/D hardware

The A/D converter of the T89C51CC02 is initialized to use P1.0 and P1.1 as analogue inputs. A standard conversion (see T89C51CC02 datasheet) is used.

Definition at line 13 of file adc.c.

Referenced by main().

3.2 adc.h File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
```

Functions

- void ADC_init (void)
  initialize A/D hardware
- WORD ADC (BYTE)
  execute A/D conversion for specific input channel

3.2.1 Detailed Description

Definition in file adc.h.

3.2.2 Function Documentation

3.2.2.1 WORD ADC (BYTE *channel*)

execute A/D conversion for specific input channel

Reads the voltage from the specified input source and does a standard conversion.

A reading of 0x0000 implies +0V while 0x03ff implies +5V

Parameters:
channel 0-7 the A/D channel we want to measure (corresponds to a pin number)

Returns:
conversion result as a 16Bit value

Definition at line 29 of file adc.c.

Referenced by Bumper_read(), Fingersensors_read(), Photosensor_read(), and Sharp_read().

3.2.2.2 void ADC_init (void)

initialize A/D hardware

The A/D converter of the T89C51CC02 is initialized to use P1.0 and P1.1 as analogue inputs. A standard conversion (see T89C51CC02 datasheet) is used.

Definition at line 13 of file adc.c.

Referenced by main().

3.3 bumper.c File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
#include "bumper.h"
#include "adc.h"
```

Functions

- void Bumper_init ()
  initialize bumper control subsystem
- void Bumper_select (BYTE sensorid)
  select one of the outer bumper sensors for measurement
- WORD Bumper_read (BYTE bumpernr)
  reads one of the perimeter bumper sensors

3.3.1 Detailed Description

Definition in file bumper.c.

3.3.2 Function Documentation

3.3.2.1 WORD Bumper_read (BYTE *bumpernr*)

reads one of the perimeter bumper sensors

A given bumper sensor is selected and then being read with the A/D converter.

Parameters:
bumpernr the bumper to be read (0-7)

Returns:
the relative voltage reading of the specified bumper (+5V means no contact)

Definition at line 57 of file bumper.c.

References ADC(), Bumper_activate, Bumper_deactivate, and Bumper_select().

Referenced by main().

3.3.2.2 void Bumper_select (BYTE *sensorid*)

select one of the outer bumper sensors for measurement

A 74HCT237 decoder IC is used to select one of the bumpers. This function applies the sensor address to its A bus.

The A Bus is connected to P2.0 P2.1 and P1.5 for A0 - A2.

Parameters:
sensorid - number of the bumper to be read

Definition at line 33 of file bumper.c.

Referenced by Bumper_read().

3.4 bumper.h File Reference

```
#include "datatypes.h"
#include "t89c51cc02.h"
```

Defines

- #define Bumper_activate() P1_4=1
- #define Bumper_deactivate() P1_4=0
- #define BUMPERSENSORS_NR 8

Functions

- void Bumper_init (void)

  initialize bumper control subsystem

- void Bumper_select (BYTE)

  select one of the outer bumper sensors for measurement

- WORD Bumper_read (BYTE)

  reads one of the perimeter bumper sensors

3.4.1 Detailed Description

Definition in file bumper.h.

3.4.2 Define Documentation

3.4.2.1 #define Bumper_activate() P1_4=1

activates reading of Bumper sensors

Definition at line 6 of file bumper.h.

Referenced by Bumper_read().

3.4.2.2 #define Bumper_deactivate() P1_4=0

deactivates reading of Bumper sensors

Definition at line 8 of file bumper.h.

Referenced by Bumper_init(), and Bumper_read().

3.4.2.3 #define BUMPERSENSORS_NR 8

number of available bumper sensors

Definition at line 11 of file bumper.h.

Referenced by main().

3.4.3 Function Documentation

3.4.3.1 WORD Bumper_read (BYTE *bumpernr*)

reads one of the perimeter bumper sensors

A given bumper sensor is selected and then being read with the A/D converter.

Parameters:
bumpernr the bumper to be read (0-7)

Returns:
the relative voltage reading of the specified bumper (+5V means no contact)

Definition at line 57 of file bumper.c.

References ADC(), Bumper_activate, Bumper_deactivate, and Bumper_select().

Referenced by main().

3.4.3.2 void Bumper_select (BYTE *sensorid*)

select one of the outer bumper sensors for measurement

A 74HCT237 decoder IC is used to select one of the bumpers. This function applies the sensor address to its A bus.

The A Bus is connected to P2.0 P2.1 and P1.5 for A0 - A2.

Parameters:
sensorid - number of the bumper to be read

Definition at line 33 of file bumper.c.

Referenced by Bumper_read().

3.5 can.c File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
#include "can.h"
#include "command.h"
```

Functions

- void CAN_init ()
  initialize CAN Bus to receive and send packages
- void CAN_interrupt (void)
  ISR for handling incoming CAN Bus messages.
- void CAN_SendACK (BYTE cmd)
  acknowledge last recieved command via CAN Bus
- void CAN_SendNAK (BYTE cmd)
  reply when last command was accepted but could not be completed
- void CAN_SendMsg (BYTE length)
  transmits a message over the CAN Bus

Variables

- volatile BYTE idata can_data [8]
  data buffer to construct CAN messages to be send

3.5.1 Detailed Description

Definition in file can.c.

3.5.2 Function Documentation

3.5.2.1 void CAN_init (void)

initialize CAN Bus to receive and send packages

CAN packets are recieved on CAN ID 0x400 (hardcoded into can.h)

CAN packets are transmitted on CAN ID 0x401

The bus is put into 250kbps mode.

Definition at line 29 of file can.c.

References CAN_setchannel, CH_DISABLE, and MSK_CANGCON_GRES.

Referenced by main().

3.5.2.2 void CAN_interrupt (void)

ISR for handling incoming CAN Bus messages.

A small implicit state machine is used to decode valid CAN messages.

Definition at line 119 of file can.c.

References can_data, CAN_SendACK(), CAN_SendMsg(), CAN_SendNAK(), CAN_setchannel, Command_CheckTime, Command_ClearMonitor, Command_ClearRead, Command_DisableMonitor(), Command_DisableReport, COMMAND_EEPROM_CLEAR, COMMAND_EEPROM_SAVEMONITOR, Command_EnableMonitor(), Command_EnableReport(), COMMAND_MONITOR, COMMAND_MONITORSTATUS, COMMAND_READ, COMMAND_RECALLMONITOR, COMMAND_REPORT, COMMAND_RESET, Command_SetRead(), COMMAND_STOPALLMONITORS, COMMAND_STOPMONITOR, COMMAND_STOPREPORT, COMMAND_TIMECHECK_DISABLE, COMMAND_TIMECHECK_ENABLE, Command_TimecheckDisable, Command_TimecheckEnable, COMMAND_TIMECHECKSTATUS, monitor, read_eeprom_config, and write_eeprom_config.

3.5.2.3 void CAN_SendACK (BYTE *cmd*)

acknowledge last recieved command via CAN Bus

The ACK message consists of the command byte to be acknowledged and an ASCII ACK

Parameters:

cmd numerical representation of last CAN command to be acknowledged

Definition at line 296 of file can.c.

References CAN_ACK, CAN_enablechannel, CAN_setchannel, CH_DISABLE, and CH_TxENA.

Referenced by CAN_interrupt().

3.5.2.4 void CAN_SendMsg (BYTE *length*)

transmits a message over the CAN Bus

The message content has to be specified in can_data and the data length code has to be given.

Transmission of the message is suspended as long as there is a message transmission in progress.

During transfer of the buffer contents to the CAN controller the CAN interrupt is disabled.

Parameters:

length length of data package

Definition at line 344 of file can.c.

References can_data, CAN_enablechannel, CAN_setchannel, CH_DISABLE, and CH_TxENA.

Referenced by CAN_interrupt(), and main().

3.5.2.5 void CAN_SendNAK (BYTE *cmd*)

reply when last command was accepted but could not be completed

The function works similar to CAN_SendACK but a NAK is send instead of the ACK to signal an imcomplete command or wrong data format.

Parameters:

cmd numerical representation of last CAN command for which the NAK answer is valid

Definition at line 319 of file can.c.

References CAN_enablechannel, CAN_NAK, CAN_setchannel, CH_DISABLE, and CH_TxENA.

Referenced by CAN_interrupt().

3.5.3 Variable Documentation

3.5.3.1 volatile BYTE idata can_data[8]

data buffer to construct CAN messages to be send

The buffer resides in indirect addressable IRAM space of the T89C51CC02.

The buffer is shared for both recieving and sending therefor disabling the CAN interrupt is advisable before sending.

Definition at line 19 of file can.c.

Referenced by CAN_interrupt(), CAN_SendMsg(), and main().

3.6 can.h File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
```

Defines

- #define BRP_500k 0x00
- #define SJW_500k 0x00
- #define PRS_500k 0x00
- #define PHS2_500k 0x07
- #define PHS1_500k 0x05
- #define BRP_250k 0x01
- #define PRS_250k 0x00
- #define PHS1_250k 0x05
- #define PHS2_250k 0x07
- #define SJW_250k 0x00
- #define MSK_CANGCON_ENA 0x02
- #define MSK_CANGCON_GRES 0x01
- #define DLC_MAX 8
- #define CH_DISABLE 0x00
- #define CH_RxENA 0x80
- #define CH_TxENA 0x40
- #define MSK_CANGIE_ENRX 0x20
- #define MSK_CANGIE_ENTX 0x10
- #define CAN_ACK 0x06
- #define CAN_NAK 0x15
- #define CAN_RECVID 0x400
- #define CAN_SENDID 0x401
- #define CAN_setchannel(ch) CANPAGE = (ch << 4)
- #define CAN_enablechannel(ch) CANEN |= (1 << ch)

Functions

- void CAN_init (void)

  initialize CAN Bus to receive and send packages

- void CAN_interrupt (void) interrupt 7 using 1

  ISR for handling incoming CAN Bus messages.

- void CAN_SendACK (BYTE)

  acknowledge last recieved command via CAN Bus

- void CAN_SendMsg (BYTE)

  transmits a message over the CAN Bus

- void CAN_SendNAK (BYTE)

  reply when last command was accepted but could not be completed

Variables

- volatile BYTE idata can_data [8]

  data buffer to construct CAN messages to be send

3.6.1 Detailed Description

Definition in file can.h.

3.6.2 Define Documentation

3.6.2.1 #define CAN_ACK 0x06

ASCII code for 'Acknowledge'

Definition at line 32 of file can.h.

Referenced by CAN_SendACK().

3.6.2.2 #define CAN_enablechannel(ch) CANEN |= (1 << ch)

macro to activate one of the CAN channels

Definition at line 44 of file can.h.

Referenced by CAN_SendACK(), CAN_SendMsg(), CAN_SendNAK(), and main().

3.6.2.3 #define CAN_NAK 0x15

ASCII code for 'Negative acknowledge'

Definition at line 34 of file can.h.

Referenced by CAN_SendNAK().

3.6.2.4 #define CAN_RECVID 0x400

we listen on this CAN ID

Definition at line 37 of file can.h.

3.6.2.5 #define CAN_SENDID 0x401

we recieve data on this ID

Definition at line 39 of file can.h.

3.6.2.6 #define CAN_setchannel(ch) CANPAGE = (ch << 4)

macro to select internal CAN channel

Definition at line 42 of file can.h.

Referenced by CAN_init(), CAN_interrupt(), CAN_SendACK(), CAN_SendMsg(), CAN_SendNAK(), and main().

3.6.2.7 #define DLC_MAX 8

maximum length of messages

Definition at line 24 of file can.h.

3.6.3 Function Documentation

3.6.3.1 void CAN_init (void)

initialize CAN Bus to receive and send packages

CAN packets are recieved on CAN ID 0x400 (hardcoded into can.h)

CAN packets are transmitted on CAN ID 0x401

The bus is put into 250kbps mode.

Definition at line 29 of file can.c.

References CAN_setchannel, CH_DISABLE, and MSK_CANGCON_GRES.

Referenced by main().

3.6.3.2 void CAN_interrupt (void)

ISR for handling incoming CAN Bus messages.

A small implicit state machine is used to decode valid CAN messages.

Definition at line 119 of file can.c.

References can_data, CAN_SendACK(), CAN_SendMsg(), CAN_SendNAK(), CAN_setchannel, Command_CheckTime, Command_ClearMonitor, Command_ClearRead, Command_DisableMonitor(), Command_DisableReport, COMMAND_EEPROM_CLEAR, COMMAND_EEPROM_SAVEMONITOR, Command_EnableMonitor(), Command_EnableReport(), COMMAND_MONITOR, COMMAND_MONITORSTATUS, COMMAND_READ, COMMAND_RECALLMONITOR, COMMAND_REPORT, COMMAND_RESET, Command_SetRead(), COMMAND_STOPALLMONITORS, COMMAND_STOPMONITOR, COMMAND_STOPREPORT, COMMAND_TIMECHECK_DISABLE, COMMAND_TIMECHECK_ENABLE, Command_TimecheckDisable, Command_TimecheckEnable, COMMAND_TIMECHECKSTATUS, monitor, read_eeprom_config, and write_eeprom_config.

3.6.3.3 void CAN_SendACK (BYTE *cmd*)

acknowledge last recieved command via CAN Bus

The ACK message consists of the command byte to be acknowledged and an ASCII ACK

Parameters:

cmd numerical representation of last CAN command to be acknowledged

Definition at line 296 of file can.c.

References CAN_ACK, CAN_enablechannel, CAN_setchannel, CH_DISABLE, and CH_TxENA.

Referenced by CAN_interrupt().

3.6.3.4 void CAN_SendMsg (BYTE *length*)

transmits a message over the CAN Bus

The message content has to be specified in can_data and the data length code has to be given.

Transmission of the message is suspended as long as there is a message transmission in progress.

During transfer of the buffer contents to the CAN controller the CAN interrupt is disabled.

Parameters:
length length of data package

Definition at line 344 of file can.c.

References can_data, CAN_enablechannel, CAN_setchannel, CH_DISABLE, and CH_TxENA.

Referenced by CAN_interrupt(), and main().

3.6.3.5 void CAN_SendNAK (BYTE *cmd*)

reply when last command was accepted but could not be completed

The function works similar to CAN_SendACK but a NAK is send instead of the ACK to signal an imcomplete command or wrong data format.

Parameters:
cmd numerical representation of last CAN command for which the NAK answer is valid

Definition at line 319 of file can.c.

References CAN_enablechannel, CAN_NAK, CAN_setchannel, CH_DISABLE, and CH_TxENA.

Referenced by CAN_interrupt().

3.6.4 Variable Documentation

3.6.4.1 volatile BYTE idata can_data[8]

data buffer to construct CAN messages to be send

The buffer resides in indirect addressable IRAM space of the T89C51CC02.

The buffer is shared for both recieving and sending therefor disabling the CAN interrupt is advisable before sending.

Definition at line 19 of file can.c.

Referenced by CAN_interrupt(), CAN_SendMsg(), and main().

3.7 command.c File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
#include "command.h"
#include "eeprom.h"
```

Functions

- void Command_SetRead (BYTE marker)
  marks a sensor for reading after next measurement
- void Command_EnableReport (BYTE timeframes)
  enables reporting of all sensors
- BYTE Command_ReportDue ()
  signal if a report of the sensor values is due
- void Command_EnableMonitor (BYTE marker, WORD boundary, bit direction)
  marks one sensor for monitoring so it will report a value above or beyond a specified boundary value
- void Command_DisableMonitor (BYTE marker)
  disable monitoring for a given sensor
- BYTE CheckMonitor (BYTE marker, WORD value)
  checks whether the monitoring condition for a given sensor is true
- WORD Command_GetBoundary (BYTE marker)
  reports the current boundary value for a given sensor
- void Command_SetBoundary (BYTE marker, WORD bound)
  set the boundary value for a given sensor
- void Command_ReadDefaultConfiguration ()
  load monitoring data from EEPROM and activate it again
- void Command_WriteDefaultConfiguration ()
  writes the current monitoring conditions into the EEPROM

Variables

- LONG readenable
  stores a bit vector to mark which sensor values have to be sent after the current sampling cycle.
- LONG monitor
- LONG monitor_direction
- volatile bit reporting

- volatile bit timecheck
- volatile bit read_eeprom_config
- volatile bit write_eeprom_config
- BYTE report
- BYTE report_reload
- WORD idata monitor_boundary [NR_SENSORS]

  boundary values for monitoring

- const unsigned long code masks [ ]

3.7.1 Detailed Description

Definition in file command.c.

3.7.2 Function Documentation

3.7.2.1 BYTE CheckMonitor (BYTE *marker*, WORD *value*)

checks whether the monitoring condition for a given sensor is true

The sample value is compared to the boundary for the given sensor.

The direction bit of the monitoring condition is used to determine the needed relation.

Parameters:

marker number of monitored sensor

value current sensor reading

Returns:

a positive value if the monitoring condition is true

Definition at line 153 of file command.c.

References masks, monitor, monitor_boundary, monitor_direction, and NR_SENSORS.

Referenced by main().

3.7.2.2 void Command_DisableMonitor (BYTE *marker*)

disable monitoring for a given sensor

Parameters:

marker number of sensor for which monitoring has to be disabled

Definition at line 137 of file command.c.

References masks, monitor, and NR_SENSORS.

Referenced by CAN_interrupt().

3.7.2.3 void Command_EnableMonitor (BYTE *marker*, WORD *boundary*, bit *direction*)

marks one sensor for monitoring so it will report a value above or beyond a specified boundary value

Monitoring of a given sensor is enabled.

A set monitoring direction means that sensor values above the specified boundary are reported. If the direction bit is not set sensor values below the boundary are reported.

Parameters:

marker number of sensor to be monitored

boundary boundary value of reading

direction 1 if report on value larger than the boundary, 0 on lower value

Definition at line 119 of file command.c.

References masks, monitor, monitor_boundary, monitor_direction, and NR_SENSORS.

Referenced by CAN_interrupt().

3.7.2.4 void Command_EnableReport (BYTE *timeframes*)

enables reporting of all sensors

The firmware will skip the given amount of timeframes between complete reports.

A timeframe value of 0 indicates that every sampling run has to be reported.

Parameters:

timeframes number of timeframes to be skipped before report is sent

Definition at line 70 of file command.c.

References report, report_reload, and reporting.

Referenced by CAN_interrupt().

3.7.2.5 WORD Command_GetBoundary (BYTE *marker*)

reports the current boundary value for a given sensor

The boundary value is persistent, even if the monitoring condition has been disabled with a Command_-EnableMonitor command for the given sensor.

Parameters:

marker number of sensor whose boundary has to be returned

Returns:

the boundary value of the specified sensor

Definition at line 184 of file command.c.

References monitor_boundary.

3.7.2.6 void Command_ReadDefaultConfiguration (void)

load monitoring data from EEPROM and activate it again

This function reads the EEPROM contents and checks its checksum. If the checksum is correct the values are used to set the monitoring conditions for up to all sensors.

Definition at line 206 of file command.c.

References CHECKSUM_FILLER, EEPROM_BOUNDARY, EEPROM_CHECKSUM, EEPROM_DIRECTION, EEPROM_MONITOR, EEPROM_read(), monitor, monitor_boundary, and monitor_direction.

Referenced by main().

3.7.2.7 BYTE Command_ReportDue (void)

signal if a report of the sensor values is due

The routine counts the elapsed timeframes for reports and signals when the report is due at the current time slot.

Returns:
a value <>0 indicates a report is due

Definition at line 84 of file command.c.

References report, report_reload, and reporting.

Referenced by main().

3.7.2.8 void Command_SetBoundary (BYTE *marker*, WORD *bound*)

set the boundary value for a given sensor

Parameters:
marker number of sensor for which the boundary has to be set
bound new boundary value

Definition at line 195 of file command.c.

References monitor_boundary.

3.7.2.9 void Command_SetRead (BYTE *marker*)

marks a sensor for reading after next measurement

The marker corresponds to a bit number in the readenable status word

Parameters:
marker

Definition at line 56 of file command.c.

References masks, NR_SENSORS, and readenable.

Referenced by CAN_interrupt().

3.7.2.10 void Command_WriteDefaultConfiguration (void)

writes the current monitoring conditions into the EEPROM

The needed checksum is calculated before writing the configuration.

Interrupts are disabled during EEPROM access and are reenabled afterwards.

Writing to the EEPROM costs time. Writing the configuration may led to violation of the timing constraints of the sensor sampling.

Definition at line 251 of file command.c.

References CHECKSUM_FILLER, monitor, monitor_boundary, and monitor_direction.

3.7.3 Variable Documentation

3.7.3.1 const unsigned long code masks[ ]

Initial value:

```
{ 0x00001,0x00002,0x00004,0x00008,0x00010,0x00020,
                                0x00040,0x00080,0x00100,0x00200,0x00400,0x00800,
                                0x01000,0x02000,0x04000,0x08000,0x10000,0x20000}
```

bit masks to select individual bits of a LONG

Definition at line 46 of file command.c.

Referenced by CheckMonitor(), Command_DisableMonitor(), Command_EnableMonitor(), and Command_SetRead().

3.7.3.2 LONG monitor

stores a bit vector to mark which sensors are monitored

Definition at line 20 of file command.c.

Referenced by CAN_interrupt(), CheckMonitor(), Command_DisableMonitor(), Command_EnableMonitor(), Command_ReadDefaultConfiguration(), and Command_WriteDefaultConfiguration().

3.7.3.3 WORD idata monitor_boundary[NR_SENSORS]

boundary values for monitoring

The values are located in the indirect addressable IRAM of the T89C51CC02.

Definition at line 43 of file command.c.

Referenced by CheckMonitor(), Command_EnableMonitor(), Command_GetBoundary(), Command_ReadDefaultConfiguration(), Command_SetBoundary(), and Command_WriteDefaultConfiguration().

3.7.3.4 LONG monitor_direction

bit vector which indicates the direction of boundary monitoring

Definition at line 23 of file command.c.

Referenced by CheckMonitor(), Command_EnableMonitor(), Command_ReadDefaultConfiguration(), and Command_WriteDefaultConfiguration().

3.7.3.5 volatile bit read_eeprom_config

semaphore to schedule a reread of the EEPROM contents

Definition at line 31 of file command.c.

Referenced by CAN_interrupt(), and main().

3.7.3.6 LONG readenable

stores a bit vector to mark which sensor values have to be sent after the current sampling cycle.

The bit vector is automatically cleared when all scheduled readings have taken place.

Definition at line 17 of file command.c.

Referenced by Command_SetRead(), and main().

3.7.3.7 BYTE report

number of timeframes for reporting

Definition at line 36 of file command.c.

Referenced by Command_EnableReport(), and Command_ReportDue().

3.7.3.8 volatile bit reporting

a flag that indicates if all sensor values should be reported

Definition at line 26 of file command.c.

Referenced by Command_EnableReport(), Command_ReportDue(), and main().

3.7.3.9 volatile bit timecheck

a flag that indicates whether the main loop shall report violations of the timing restrictions

Definition at line 28 of file command.c.

3.7.3.10 volatile bit write_eeprom_config

semaphore to schedule writing of new EEPROM contents

Definition at line 33 of file command.c.

Referenced by CAN_interrupt().

3.8 command.h File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
```

Defines

- #define NR_SENSORS 18
- #define COMMAND_READ 0x00
- #define COMMAND_MONITOR 0x10
- #define COMMAND_STOPMONITOR 0x11
- #define COMMAND_MONITORSTATUS 0x12
- #define COMMAND_RECALLMONITOR 0x13
- #define COMMAND_STOPALLMONITORS 0x14
- #define COMMAND_REPORT 0x20
- #define COMMAND_STOPREPORT 0x21
- #define COMMAND_TIMECHECK_ENABLE 0xA0
- #define COMMAND_TIMECHECK_DISABLE 0xA1
- #define COMMAND_TIMECHECKSTATUS 0xA2
- #define COMMAND_EEPROM_SAVEMONITOR 0xE0
- #define COMMAND_EEPROM_CLEAR 0xE1
- #define COMMAND_RESET 0xff
- #define Command_ClearRead() readenable=0
- #define Command_DisableReport() reporting=0
- #define Command_ClearMonitor() monitor=0
- #define Command_CheckMonitor() (monitor!=0)
- #define Command_TimecheckEnable() timecheck=1
- #define Command_TimecheckDisable() timecheck=0
- #define Command_CheckTime() (timecheck==1)
- #define CHECKSUM_FILLER 0x3c2a

Functions

- void Command_SetRead (BYTE)

  marks a sensor for reading after next measurement

- BYTE Command_ReportDue (void)

  signal if a report of the sensor values is due

- void Command_EnableMonitor (BYTE, WORD, bit)

  marks one sensor for monitoring so it will report a value above or beyond a specified boundary value

- void Command_DisableMonitor (BYTE)

  disable monitoring for a given sensor

- BYTE CheckMonitor (BYTE, WORD)

  checks whether the monitoring condition for a given sensor is true

- void Command_EnableReport (BYTE)

enables reporting of all sensors

- WORD Command_GetBoundary (BYTE)

  reports the current boundary value for a given sensor

- void Command_SetBoundary (BYTE, WORD)

  set the boundary value for a given sensor

- void Command_ReadDefaultConfiguration (void)

  load monitoring data from EEPROM and activate it again

- void Command_WriteDefaultConfiguration (void)

  writes the current monitoring conditions into the EEPROM

Variables

- LONG readenable

  stores a bit vector to mark which sensor values have to be sent after the current sampling cycle.

- LONG monitor
- LONG monitor_direction
- volatile bit reporting
- volatile bit timecheck
- volatile bit read_eeprom_config
- volatile bit write_eeprom_config

3.8.1 Detailed Description

Definition in file command.h.

3.8.2 Define Documentation

3.8.2.1 #define CHECKSUM_FILLER 0x3c2a

start value for checksum calculation

Definition at line 59 of file command.h.

Referenced by Command_ReadDefaultConfiguration(), and Command_WriteDefaultConfiguration().

3.8.2.2 #define Command_CheckMonitor() (monitor!=0)

checkes whether sensors are monitored

Definition at line 50 of file command.h.

3.8.2.3 #define Command_CheckTime() (timecheck==1)

checks whether monitoring of the timing constraints is active

Definition at line 56 of file command.h.

Referenced by CAN_interrupt().

3.8.2.4 #define Command_ClearMonitor() monitor=0

macro to disable all monitoring conditions

Definition at line 48 of file command.h.

Referenced by CAN_interrupt().

3.8.2.5 #define Command_ClearRead() readenable=0

macro to clear reading markers

Definition at line 44 of file command.h.

Referenced by CAN_interrupt().

3.8.2.6 #define Command_DisableReport() reporting=0

macro to disable whole sample reporting

Definition at line 46 of file command.h.

Referenced by CAN_interrupt().

3.8.2.7 #define COMMAND_EEPROM_CLEAR 0xE1

symbolic command name to clear the monitoring configuration inside of the EEPROM

Definition at line 39 of file command.h.

Referenced by CAN_interrupt().

3.8.2.8 #define COMMAND_EEPROM_SAVEMONITOR 0xE0

symbolic command name to save the active monitoring conditions to the EEPROM

Definition at line 37 of file command.h.

Referenced by CAN_interrupt().

3.8.2.9 #define COMMAND_MONITOR 0x10

symbolic command name to set a monitoring condition for a given sensor

Definition at line 17 of file command.h.

Referenced by CAN_interrupt().

3.8.2.10 #define COMMAND_MONITORSTATUS 0x12

symbolic command name to read the status of active monitoring conditions

Definition at line 21 of file command.h.

Referenced by CAN_interrupt().

3.8.2.11 #define COMMAND_READ 0x00

symbolic command name to schedule reading of a given sensor

Definition at line 15 of file command.h.

Referenced by CAN_interrupt().

3.8.2.12 #define COMMAND_RECALLMONITOR 0x13

symbolic command name to schedule a reinitialization of the monitoring conditions from EEPROM contents

Definition at line 23 of file command.h.

Referenced by CAN_interrupt().

3.8.2.13 #define COMMAND_REPORT 0x20

symbolic command name to schedule complete sample reports

Definition at line 27 of file command.h.

Referenced by CAN_interrupt().

3.8.2.14 #define COMMAND_RESET 0xff

symbolic command name to reset all monitoring and reporting conditions

Definition at line 41 of file command.h.

Referenced by CAN_interrupt().

3.8.2.15 #define COMMAND_STOPALLMONITORS 0x14

symbolic command name to remove all active monitoring conditions

Definition at line 25 of file command.h.

Referenced by CAN_interrupt().

3.8.2.16 #define COMMAND_STOPMONITOR 0x11

symbolic command name to disable a monitoring condition for a given sensor

Definition at line 19 of file command.h.

Referenced by CAN_interrupt().

3.8.2.17 #define COMMAND_STOPREPORT 0x21

symbolic command name to stop complete reporting

Definition at line 29 of file command.h.

Referenced by CAN_interrupt().

3.8.2.18 #define COMMAND_TIMECHECK_DISABLE 0xA1

symbolic command name to disable time constraint monitoring

Definition at line 33 of file command.h.

Referenced by CAN_interrupt().

3.8.2.19 #define COMMAND_TIMECHECK_ENABLE 0xA0

symbolic command name to enable monitoring of the time constraints

Definition at line 31 of file command.h.

Referenced by CAN_interrupt().

3.8.2.20 #define Command_TimecheckDisable() timecheck=0

macro to disable monitoring of the timing constraints

Definition at line 54 of file command.h.

Referenced by CAN_interrupt().

3.8.2.21 #define Command_TimecheckEnable() timecheck=1

macro to enable monitoring of the timing constraints

Definition at line 52 of file command.h.

Referenced by CAN_interrupt(), and main().

3.8.2.22 #define COMMAND_TIMECHECKSTATUS 0xA2

symbolic command name to read the status of time constraint monitoring

Definition at line 35 of file command.h.

Referenced by CAN_interrupt().

3.8.2.23 #define NR_SENSORS 18

number of sensors available in the system

Definition at line 12 of file command.h.

Referenced by CheckMonitor(), Command_DisableMonitor(), Command_EnableMonitor(), and Command_SetRead().

3.8.3 Function Documentation

3.8.3.1 BYTE CheckMonitor (BYTE *marker*, WORD *value*)

checks whether the monitoring condition for a given sensor is true

The sample value is compared to the boundary for the given sensor.

The direction bit of the monitoring condition is used to determine the needed relation.

Parameters:

marker number of monitored sensor

value current sensor reading

Returns:

a positive value if the monitoring condition is true

Definition at line 153 of file command.c.

References masks, monitor, monitor_boundary, monitor_direction, and NR_SENSORS.

Referenced by main().

3.8.3.2 void Command_DisableMonitor (BYTE *marker*)

disable monitoring for a given sensor

Parameters:

marker number of sensor for which monitoring has to be disabled

Definition at line 137 of file command.c.

References masks, monitor, and NR_SENSORS.

Referenced by CAN_interrupt().

3.8.3.3 void Command_EnableMonitor (BYTE *marker*, WORD *boundary*, bit *direction*)

marks one sensor for monitoring so it will report a value above or beyond a specified boundary value

Monitoring of a given sensor is enabled.

A set monitoring direction means that sensor values above the specified boundary are reported. If the direction bit is not set sensor values below the boundary are reported.

Parameters:

marker number of sensor to be monitored

boundary boundary value of reading

direction 1 if report on value larger than the boundary, 0 on lower value

Definition at line 119 of file command.c.

References masks, monitor, monitor_boundary, monitor_direction, and NR_SENSORS.

Referenced by CAN_interrupt().

3.8.3.4 void Command_EnableReport (BYTE *timeframes*)

enables reporting of all sensors

The firmware will skip the given amount of timeframes between complete reports.

A timeframe value of 0 indicates that every sampling run has to be reported.

Parameters:
timeframes number of timeframes to be skipped before report is sent

Definition at line 70 of file command.c.

References report, report_reload, and reporting.

Referenced by CAN_interrupt().

3.8.3.5 WORD Command_GetBoundary (BYTE *marker*)

reports the current boundary value for a given sensor

The boundary value is persistent, even if the monitoring condition has been disabled with a Command_EnableMonitor command for the given sensor.

Parameters:
marker number of sensor whose boundary has to be returned

Returns:
the boundary value of the specified sensor

Definition at line 184 of file command.c.

References monitor_boundary.

3.8.3.6 void Command_ReadDefaultConfiguration (void)

load monitoring data from EEPROM and activate it again

This function reads the EEPROM contents and checks its checksum. If the checksum is correct the values are used to set the monitoring conditions for up to all sensors.

Definition at line 206 of file command.c.

References CHECKSUM_FILLER, EEPROM_BOUNDARY, EEPROM_CHECKSUM, EEPROM_DIRECTION, EEPROM_MONITOR, EEPROM_read(), monitor, monitor_boundary, and monitor_direction.

Referenced by main().

3.8.3.7 BYTE Command_ReportDue (void)

signal if a report of the sensor values is due

The routine counts the elapsed timeframes for reports and signals when the report is due at the current time slot.

Returns:
a value <>0 indicates a report is due

Definition at line 84 of file command.c.

References report, report_reload, and reporting.

Referenced by main().

3.8.3.8 void Command_SetBoundary (BYTE *marker*, WORD *bound*)

set the boundary value for a given sensor

Parameters:

marker number of sensor for which the boundary has to be set

bound new boundary value

Definition at line 195 of file command.c.

References monitor_boundary.

3.8.3.9 void Command_SetRead (BYTE *marker*)

marks a sensor for reading after next measurement

The marker corresponds to a bit number in the readenable status word

Parameters:

marker

Definition at line 56 of file command.c.

References masks, NR_SENSORS, and readenable.

Referenced by CAN_interrupt().

3.8.3.10 void Command_WriteDefaultConfiguration (void)

writes the current monitoring conditions into the EEPROM

The needed checksum is calculated before writing the configuration.

Interrupts are disabled during EEPROM access and are reenabled afterwards.

Writing to the EEPROM costs time. Writing the configuration may led to violation of the timing constraints of the sensor sampling.

Definition at line 251 of file command.c.

References CHECKSUM_FILLER, monitor, monitor_boundary, and monitor_direction.

3.8.4 Variable Documentation

3.8.4.1 LONG monitor

stores a bit vector to mark which sensors are monitored

Definition at line 20 of file command.c.

Referenced by CAN_interrupt(), CheckMonitor(), Command_DisableMonitor(), Command_EnableMonitor(), Command_ReadDefaultConfiguration(), and Command_WriteDefaultConfiguration().

3.8.4.2 LONG monitor_direction

bit vector which indicates the direction of boundary monitoring

Definition at line 23 of file command.c.

Referenced by CheckMonitor(), Command_EnableMonitor(), Command_ReadDefaultConfiguration(), and Command_WriteDefaultConfiguration().

3.8.4.3 volatile bit read_eeprom_config

semaphore to schedule a reread of the EEPROM contents

Definition at line 31 of file command.c.

Referenced by CAN_interrupt(), and main().

3.8.4.4 LONG readenable

stores a bit vector to mark which sensor values have to be sent after the current sampling cycle.

The bit vector is automatically cleared when all scheduled readings have taken place.

Definition at line 17 of file command.c.

Referenced by Command_SetRead(), and main().

3.8.4.5 volatile bit reporting

a flag that indicates if all sensor values should be reported

Definition at line 26 of file command.c.

Referenced by Command_EnableReport(), Command_ReportDue(), and main().

3.8.4.6 volatile bit timecheck

a flag that indicates whether the main loop shall report violations of the timing restrictions

Definition at line 28 of file command.c.

3.8.4.7 volatile bit write_eeprom_config

semaphore to schedule writing of new EEPROM contents

Definition at line 33 of file command.c.

Referenced by CAN_interrupt().

3.9 datatypes.h File Reference

Defines

- #define DATATYPES 1

Typedefs

- typedef unsigned char BYTE
- typedef unsigned int WORD
- typedef unsigned long LONG

3.9.1 Detailed Description

Definition in file datatypes.h.

3.9.2 Define Documentation

3.9.2.1 #define DATATYPES 1

ensures that datatypes are not defined twice

Definition at line 25 of file datatypes.h.

3.9.3 Typedef Documentation

3.9.3.1 BYTE

defines the unified datatype BYTE which contains 8bits (default datatype on 8051 derivates)

Definition at line 8 of file datatypes.h.

3.9.3.2 LONG

defines datatype long unisgned 32bit integer

Definition at line 22 of file datatypes.h.

3.9.3.3 WORD

defines datatype WORD 16bit unsigned integer

Definition at line 15 of file datatypes.h.

3.10 eeprom.c File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
#include "eeprom.h"
```

Functions

- BYTE EEPROM_read (BYTE address)
  read a byte from the EEPROM
- void EEPROM_write (BYTE d, BYTE address)
  write a byte into the EEPROM latches
- void EEPROM_flush ()
  writes latched EEPROM contents to the EEPROM

Variables

- unsigned char xdata * eepromptr

3.10.1 Detailed Description

Definition in file eeprom.c.

3.10.2 Function Documentation

3.10.2.1 void EEPROM_flush (void)

writes latched EEPROM contents to the EEPROM

Prepare the new contents with EEPROM_write before writing them to the EEPROM.

Interrupts are disabled (and restored afterwards) during the actual EEPROM write.

Definition at line 77 of file eeprom.c.

References EEPROM_wait.

3.10.2.2 BYTE EEPROM_read (BYTE *address*)

read a byte from the EEPROM

A byte is read from the EEPROM at the given EEPROM address.

This routine is limited for reading the first 256 Bytes of the EEPROM memory.

Parameters:

address address offset into EEPROM to read from

Returns:
value of EEPROM at the given address

Definition at line 23 of file eeprom.c.

References EEPROM_wait, and eepromptr.

Referenced by Command_ReadDefaultConfiguration().

3.10.2.3 void EEPROM_write (BYTE *d*, BYTE *address*)

write a byte into the EEPROM latches

This function only latches data but the data is not written to the EEPROM.

Execute EEPROM_flush to actually write the new contents into the EEPROM.

Parameters:
d data byte to write

address address of EEPROM cell

Definition at line 54 of file eeprom.c.

References EEPROM_wait, and eepromptr.

3.10.3 Variable Documentation

3.10.3.1 unsigned char xdata* eepromptr

address pointer into EEPROM

Definition at line 12 of file eeprom.c.

Referenced by EEPROM_read(), and EEPROM_write().

3.11 eeprom.h File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
```

Defines

- #define EEPROM_CANID_RECV 0
- #define EEPROM CANID_SEND 2
- #define EEPROM_MONITOR 4
- #define EEPROM_DIRECTION 8
- #define EEPROM_BOUNDARY 12
- #define EEPROM_CHECKSUM 48
- #define EEPROM_wait() while((EECON&MSK_EECON_EEBUSY)!=0)

Functions

- BYTE EEPROM_read (BYTE)

  read a byte from the EEPROM

- void EEPROM_write (BYTE, BYTE)

  write a byte into the EEPROM latches

- void EEPROM_flush (void)

  writes latched EEPROM contents to the EEPROM

3.11.1 Detailed Description

Definition in file eeprom.h.

3.11.2 Define Documentation

3.11.2.1 #define EEPROM CANID_SEND 2

offset address for CAN sending ID inside of EEPROM

Definition at line 17 of file eeprom.h.

3.11.2.2 #define EEPROM_BOUNDARY 12

offset address into EEPROM for storing boundary values for all sensors

Definition at line 23 of file eeprom.h.

Referenced by Command_ReadDefaultConfiguration().

3.11.2.3 #define EEPROM_CANID_RECV 0

offset address for CAN reception ID inside of EEPROM

Definition at line 15 of file eeprom.h.

3.11.2.4 #define EEPROM_CHECKSUM 48

offset address into EEPROM to store the checksum

Definition at line 25 of file eeprom.h.

Referenced by Command_ReadDefaultConfiguration().

3.11.2.5 #define EEPROM_DIRECTION 8

offset address into EEPROM to store monitoring direction bits

Definition at line 21 of file eeprom.h.

Referenced by Command_ReadDefaultConfiguration().

3.11.2.6 #define EEPROM_MONITOR 4

offset address into EEPROM to store monitor flags

Definition at line 19 of file eeprom.h.

Referenced by Command_ReadDefaultConfiguration().

3.11.2.7 #define EEPROM_wait() while((EECON&MSK_EECON_EEBUSY)!=0)

waits while the EEPROM is busy

Definition at line 27 of file eeprom.h.

Referenced by EEPROM_flush(), EEPROM_read(), and EEPROM_write().

3.11.3 Function Documentation

3.11.3.1 void EEPROM_flush (void)

writes latched EEPROM contents to the EEPROM

Prepare the new contents with EEPROM_write before writing them to the EEPROM.

Interrupts are disabled (and restored afterwards) during the actual EEPROM write.

Definition at line 77 of file eeprom.c.

References EEPROM_wait.

3.11.3.2 BYTE EEPROM_read (BYTE *address*)

read a byte from the EEPROM

A byte is read from the EEPROM at the given EEPROM address.

This routine is limited for reading the first 256 Bytes of the EEPROM memory.

Parameters:

address address offset into EEPROM to read from

Returns:

value of EEPROM at the given address

Definition at line 23 of file eeprom.c.

References EEPROM_wait, and eepromptr.

Referenced by Command_ReadDefaultConfiguration().

3.11.3.3 void EEPROM_write (BYTE *d*, BYTE *address*)

write a byte into the EEPROM latches

This function only latches data but the data is not written to the EEPROM.

Execute EEPROM_flush to actually write the new contents into the EEPROM.

Parameters:

d data byte to write

address address of EEPROM cell

Definition at line 54 of file eeprom.c.

References EEPROM_wait, and eepromptr.

3.12 fingersensors.c File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
#include "fingersensors.h"
#include "adc.h"
```

Functions

- void Fingersensors_init ()
  initializes fingersensors subsystem
- void Fingersensors_read ()
  read all finger mounted force sensing resistors

Variables

- WORD idata fingersensors [FINGERSENSORS_NR]
  storage buffer for current fingersensor values

3.12.1 Detailed Description

Definition in file fingersensors.c.

3.12.2 Function Documentation

3.12.2.1 void Fingersensors_init (void)

initializes fingersensors subsystem

P1.1 and P1.2 are configured as valid analogue inputs for the subsystem.

Definition at line 33 of file fingersensors.c.

References fingersensors, FINGERSENSORS_NR, and Fingersensors_PowerOff.

Referenced by main().

3.12.2.2 void Fingersensors_read (void)

read all finger mounted force sensing resistors

The samples are saved into the buffer. (see fingersensors)

Definition at line 49 of file fingersensors.c.

References ADC(), fingersensors, Fingersensors_PowerOff, Fingersensors_SelectBank1, and Fingersensors_SelectBank2.

Referenced by main().

3.12.3 Variable Documentation

3.12.3.1 WORD idata fingersensors[FINGERSENSORS_NR]

storage buffer for current fingersensor values

The buffer is located in indirect addressable IRAM of the T89C51CC02

Definition at line 26 of file fingersensors.c.

Referenced by Fingersensors_init(), Fingersensors_read(), and main().

3.13 fingersensors.h File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
```

Defines

- #define FINGERSENSOR_COLUMN_SELECT0 P1_6
- #define FINGERSENSOR_COLUMN_SELECT1 P1_7
- #define Fingersensors_PowerOff() FINGERSENSOR_COLUMN_SELECT0=0; FINGERSENSOR_COLUMN_SELECT1=0
- #define Fingersensors_SelectBank1() FINGERSENSOR_COLUMN_SELECT0=1; FINGERSENSOR_COLUMN_SELECT1=0
- #define Fingersensors_SelectBank2() FINGERSENSOR_COLUMN_SELECT0=0; FINGERSENSOR_COLUMN_SELECT1=1
- #define FINGERSENSORS_NR 4

Functions

- void Fingersensors_init (void)
  initializes fingersensors subsystem
- void Fingersensors_read (void)
  read all finger mounted force sensing resistors

Variables

- WORD idata fingersensors [FINGERSENSORS_NR]
  storage buffer for current fingersensor values

3.13.1 Detailed Description

Definition in file fingersensors.h.

3.13.2 Define Documentation

3.13.2.1 #define FINGERSENSOR_COLUMN_SELECT0 P1_6

port pin to access sensor matrix column 0

Definition at line 5 of file fingersensors.h.

3.13.2.2 #define FINGERSENSOR_COLUMN_SELECT1 P1_7

port pin to access sensor matrix column 1

Definition at line 7 of file fingersensors.h.

3.13.2.3 #define FINGERSENSORS_NR 4

number of mounted finger sensors

Definition at line 16 of file fingersensors.h.

Referenced by Fingersensors_init(), and main().

3.13.2.4 #define Fingersensors_PowerOff() FINGERSENSOR_COLUMN_SELECT0=0; FINGERSENSOR_COLUMN_SELECT1=0

disables finger mounted sensors

Definition at line 10 of file fingersensors.h.

Referenced by Fingersensors_init(), and Fingersensors_read().

3.13.2.5 #define Fingersensors_SelectBank1() FINGERSENSOR_COLUMN_SELECT0=1; FINGERSENSOR_COLUMN_SELECT1=0

selects Bank 1 of the finger mounted sensors

Definition at line 12 of file fingersensors.h.

Referenced by Fingersensors_read().

3.13.2.6 #define Fingersensors_SelectBank2() FINGERSENSOR_COLUMN_SELECT0=0; FINGERSENSOR_COLUMN_SELECT1=1

selects Bank 2 of the finger mounted sensors

Definition at line 14 of file fingersensors.h.

Referenced by Fingersensors_read().

3.13.3 Function Documentation

3.13.3.1 void Fingersensors_init (void)

initializes fingersensors subsystem

P1.1 and P1.2 are configured as valid analogue inputs for the subsystem.

Definition at line 33 of file fingersensors.c.

References fingersensors, FINGERSENSORS_NR, and Fingersensors_PowerOff.

Referenced by main().

3.13.3.2 void Fingersensors_read (void)

read all finger mounted force sensing resistors

The samples are saved into the buffer. (see fingersensors)

Definition at line 49 of file fingersensors.c.

References ADC(), fingersensors, Fingersensors_PowerOff, Fingersensors_SelectBank1, and Fingersensors_SelectBank2.

Referenced by main().

3.13.4 Variable Documentation

3.13.4.1 WORD idata fingersensors[FINGERSENSORS_NR]

storage buffer for current fingersensor values

The buffer is located in indirect addressable IRAM of the T89C51CC02

Definition at line 26 of file fingersensors.c.

Referenced by Fingersensors_init(), Fingersensors_read(), and main().

3.14 main.c File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
#include "adc.h"
#include "can.h"
#include "timer.h"
#include "sharp.h"
#include "fingersensors.h"
#include "photosensor.h"
#include "bumper.h"
#include "command.h"
#include "eeprom.h"
```

Defines

- #define DEBUG 1
- #define LENGTH_SENSORBUFFER 128
- #define SAMPLEPOINTS 18
- #define BUFFER_LENGTH 6 ∗ SAMPLEPOINTS
- #define SERIALNUMBER 0x01

Functions

- void main ()

Variables

- const char version [ ] = "Sensor Control 07092007 #1"
- WORD xdata sensorbuffer [BUFFER_LENGTH]

  circular buffer for sensor readings

3.14.1 Detailed Description

Definition in file main.c.

3.14.2 Define Documentation

3.14.2.1 #define BUFFER_LENGTH 6 ∗ SAMPLEPOINTS

size of sensor backbuffer in WORDS

Definition at line 34 of file main.c.

Referenced by main().

3.14.2.2 #define DEBUG 1

Debug flag - enables additional debug output via CAN

Definition at line 24 of file main.c.

3.14.2.3 #define LENGTH_SENSORBUFFER 128

Maximum size of the external data memory in 16Bit Words

Definition at line 27 of file main.c.

3.14.2.4 #define SAMPLEPOINTS 18

combined number of samples taken per measurement point

Definition at line 31 of file main.c.

Referenced by main().

3.14.2.5 #define SERIALNUMBER 0x01

main serial number of the firmware

Definition at line 37 of file main.c.

Referenced by main().

3.14.3 Function Documentation

3.14.3.1 void main ()

Initializes all sensor subsystems and implements the main sensor reading cycle.

Definition at line 53 of file main.c.

References ADC_init(), BUFFER_LENGTH, Bumper_init(), Bumper_read(), BUMPERSENSORS_NR, can_data, CAN_enablechannel, CAN_init(), CAN_SendMsg(), CAN_setchannel, CH_DISABLE, CH_-TxENA, CheckMonitor(), Command_ReadDefaultConfiguration(), Command_ReportDue(), Command_-TimecheckEnable, fingersensors, Fingersensors_init(), FINGERSENSORS_NR, Fingersensors_read(), measurement_task, Photosensor_init(), Photosensor_PowerOff, Photosensor_read(), read_eeprom_config, readenable, reporting, SAMPLEPOINTS, sensorbuffer, SERIALNUMBER, Sharp_init(), Sharp_Power-Off, Sharp_PowerOn, Sharp_read(), and Timer_init().

3.14.4 Variable Documentation

3.14.4.1 WORD xdata sensorbuffer[BUFFER_LENGTH]

circular buffer for sensor readings

We store all sequential sensor readings into this buffer.

The buffer resides in XRAM space of the T89C51CC02 ("Externer Datenspeicher")

Definition at line 48 of file main.c.

Referenced by main().

3.14.4.2 const char version[ ] = "Sensor Control 07092007 #1"

the version number encodes the current date in DDMMYYYY form and a revision number

Definition at line 40 of file main.c.

3.15 photosensor.c File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
#include "photosensor.h"
#include "adc.h"
#include "sharp.h"
#include <stdlib.h>
```

Functions

- void Photosensor_init ()
  initialize photosensor subsystem
- void Photosensor_toggle (void)
  toggles LED of selected photosensor
- WORD Photosensor_read (BYTE sensornr)
  read the value of a single photosensor

Variables

- BYTE psensor
- const char psensor_select [ ] = {0,0x08,0x10,0x20,0x40,0x80,0,0}

3.15.1 Detailed Description

Definition in file photosensor.c.

3.15.2 Function Documentation

3.15.2.1 void Photosensor_init (void)

initialize photosensor subsystem

The photosensors are turned off by default.

Definition at line 31 of file photosensor.c.

References Photosensor_PowerOff, and psensor.

Referenced by main().

3.15.2.2 WORD Photosensor_read (BYTE *sensornr*)

read the value of a single photosensor

The given sensor is activated first and than sampled with a simple differential scheme.

Parameters:
sensornr number of photo sensor to read

Returns:
voltage value of given sensor

Definition at line 79 of file photosensor.c.

References ADC(), Photosensor_PowerOff, Photosensor_select, and SHARP_PWRCTL.

Referenced by main().

3.15.2.3 void Photosensor_toggle (void)

toggles LED of selected photosensor

The photosensor is selected with the Photosensor_select function.

Invalid sensor numbers will turn off all photosensors.

Definition at line 44 of file photosensor.c.

References Photosensor_PowerOff, Photosensor_select, and psensor.

3.15.3 Variable Documentation

3.15.3.1 BYTE psensor

stores number of active photosensor

Definition at line 21 of file photosensor.c.

Referenced by Photosensor_init(), and Photosensor_toggle().

3.15.3.2 const char psensor_select[ ] = {0,0x08,0x10,0x20,0x40,0x80,0,0}

bit masks to activate 1 of 5 photo sensors

Definition at line 24 of file photosensor.c.

3.16 photosensor.h File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
```

Defines

- #define PHOTOSENSOR_OFF 0x03
- #define Photosensor_PowerOff() P3=(P3 & PHOTOSENSOR_OFF)
- #define Photosensor_select(s) psensor=s; P3=((P3 & PHOTOSENSOR_OFF)|psensor_-select[psensor & 0x07])
- #define PHOTOSENSORS_NR 5

Functions

- void Photosensor_init (void)

  initialize photosensor subsystem

- void Photosensor_toggle (void)

  toggles LED of selected photosensor

- WORD Photosensor_read (BYTE)

  read the value of a single photosensor

3.16.1 Detailed Description

Definition in file photosensor.h.

3.16.2 Define Documentation

3.16.2.1 #define PHOTOSENSOR_OFF 0x03

bit mask used to turn off all photosensors

Definition at line 7 of file photosensor.h.

3.16.2.2 #define Photosensor_PowerOff() P3=(P3 & PHOTOSENSOR_OFF)

macro to turn off all photosensors

Definition at line 9 of file photosensor.h.

Referenced by main(), Photosensor_init(), Photosensor_read(), and Photosensor_toggle().

3.16.2.3 #define Photosensor_select(s) psensor=s; P3=((P3 & PHOTOSENSOR_OFF)|psensor_select[psensor & 0x07])

macro to activate a given photosensor

Definition at line 11 of file photosensor.h.

Referenced by Photosensor_read(), and Photosensor_toggle().

3.16.2.4 #define PHOTOSENSORS_NR 5

number of active photosensors

Definition at line 13 of file photosensor.h.

3.16.3 Function Documentation

3.16.3.1 void Photosensor_init (void)

initialize photosensor subsystem

The photosensors are turned off by default.

Definition at line 31 of file photosensor.c.

References Photosensor_PowerOff, and psensor.

Referenced by main().

3.16.3.2 WORD Photosensor_read (BYTE *sensornr*)

read the value of a single photosensor

The given sensor is activated first and than sampled with a simple differential scheme.

Parameters:

sensornr number of photo sensor to read

Returns:

voltage value of given sensor

Definition at line 79 of file photosensor.c.

References ADC(), Photosensor_PowerOff, Photosensor_select, and SHARP_PWRCTL.

Referenced by main().

3.16.3.3 void Photosensor_toggle (void)

toggles LED of selected photosensor

The photosensor is selected with the Photosensor_select function.

Invalid sensor numbers will turn off all photosensors.

Definition at line 44 of file photosensor.c.

References Photosensor_PowerOff, Photosensor_select, and psensor.

3.17 sharp.c File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
#include "adc.h"
#include "sharp.h"
```

Functions

- void Sharp_init ()

  initializes GP2D120 subsystem

- void Sharp_TimerInit ()

  initializes an ISR to signal that the Sharp GP2D120 is ready to be read.

- void Sharp_Timer_interrupt (void)

  Timer ISR for delaying the Sharp GP2D120.

- WORD Sharp_read ()

  read Sharp GP2D120 sensor if it is active

Variables

- volatile bit sharp_ready
- volatile bit wait_bit

3.17.1 Detailed Description

Definition in file sharp.c.

3.17.2 Function Documentation

3.17.2.1 WORD Sharp_read (void)

read Sharp GP2D120 sensor if it is active

Returns:
0xffff if Sharp is not active, Sharp reading else

Definition at line 86 of file sharp.c.

References ADC(), SHARP_PWRCTL, and sharp_ready.

Referenced by main().

3.17.2.2 void Sharp_Timer_interrupt (void)

Timer ISR for delaying the Sharp GP2D120.

This Timer ISR delays the Sharp GP2D120 and signals its readiness after 50ms via the sharp_ready semaphore. The interrupt is triggered every 25ms. Therefor the routine uses a flag (wait_bit) to signal its second run. The sensor readiness is signalled in the second run of the ISR. The ISR deactivates itself after its second execution.

Definition at line 64 of file sharp.c.

References sharp_ready, SHARP_TIMER_RELOAD_H, SHARP_TIMER_RELOAD_L, and wait_bit.

3.17.2.3 void Sharp_TimerInit (void)

initializes an ISR to signal that the Sharp GP2D120 is ready to be read.

Initializes an ISR to signal that the Sharp GP2D120 is ready to output a valid distance reading. Activates Timer 1 which counts for 25ms twice and sets a readiness signal upon 2nd ISR invocation. (

See also:
sharp_ready)

Definition at line 39 of file sharp.c.

References sharp_ready, SHARP_TIMER_RELOAD_H, SHARP_TIMER_RELOAD_L, and wait_bit.

3.17.3 Variable Documentation

3.17.3.1 volatile bit sharp_ready

A semaphore that signals whether the GP2D120 is ready to be read

Definition at line 19 of file sharp.c.

Referenced by Sharp_read(), Sharp_Timer_interrupt(), and Sharp_TimerInit().

3.17.3.2 volatile bit wait_bit

A semaphore that signals the second run of the Sharp delaying Timer interrupt if 0

Definition at line 21 of file sharp.c.

Referenced by Sharp_Timer_interrupt(), and Sharp_TimerInit().

3.18 sharp.h File Reference

```
#include "datatypes.h"
#include "t89c51cc02.h"
```

Defines

- #define SHARP_PWRCTL P3_2
- #define SHARP_PWRCTL_MASK 0x04
- #define SHARP_ONLY 0x03
- #define SHARP_TIMER_RELOAD_H 0x7d
- #define SHARP_TIMER_RELOAD_L 0xcb
- #define Sharp_PowerOn() P3=(P3 & SHARP_ONLY)|SHARP_PWRCTL_MASK; Sharp_TimerInit();
- #define Sharp_PowerOff() SHARP_PWRCTL=0; TR1=0; ET1=0;

Functions

- void Sharp_init (void)
  initializes GP2D120 subsystem
- void Sharp_TimerInit (void)
  initializes an ISR to signal that the Sharp GP2D120 is ready to be read.
- WORD Sharp_read (void)
  read Sharp GP2D120 sensor if it is active
- void Sharp_Timer_interrupt (void) interrupt 3 using 2
  Timer ISR for delaying the Sharp GP2D120.

3.18.1 Detailed Description

Definition in file sharp.h.

3.18.2 Define Documentation

3.18.2.1 #define SHARP_ONLY 0x03

bit mask to clear all enable bits for optic sensors

Definition at line 11 of file sharp.h.

3.18.2.2 #define Sharp_PowerOff() SHARP_PWRCTL=0; TR1=0; ET1=0;

deactivates the Sharp GP2D120 and its Timer

Definition at line 21 of file sharp.h.

Referenced by main(), and Sharp_init().

3.18.2.3 #define Sharp_PowerOn() P3=(P3 & SHARP_ONLY)|SHARP_PWRCTL_MASK; Sharp_TimerInit();

deactivates all optic sensors except for the Sharp GP2D120 and starts the Sharp Timer ISR (see Sharp_-Timer_interrupt)

Definition at line 19 of file sharp.h.

Referenced by main().

3.18.2.4 #define SHARP_PWRCTL P3_2

Port pin that activates the powersupply for the Sharp GP2D120

Definition at line 6 of file sharp.h.

Referenced by Photosensor_read(), and Sharp_read().

3.18.2.5 #define SHARP_PWRCTL_MASK 0x04

bit mask for accessing the Shapr powercontrol flag

Definition at line 8 of file sharp.h.

3.18.2.6 #define SHARP_TIMER_RELOAD_H 0x7d

reload values to get approx. 25ms timing @ 16MHz: 0x10000 - 0x8235 = 0x7dcb - HIByte

Definition at line 14 of file sharp.h.

Referenced by Sharp_Timer_interrupt(), and Sharp_TimerInit().

3.18.2.7 #define SHARP_TIMER_RELOAD_L 0xcb

reload values to get approx. 25ms timing @ 16MHz: 0x10000 - 0x8235 = 0x7dcb - LOByte

Definition at line 16 of file sharp.h.

Referenced by Sharp_Timer_interrupt(), and Sharp_TimerInit().

3.18.3 Function Documentation

3.18.3.1 WORD Sharp_read (void)

read Sharp GP2D120 sensor if it is active

Returns:
0xffff if Sharp is not active, Sharp reading else

Definition at line 86 of file sharp.c.

References ADC(), SHARP_PWRCTL, and sharp_ready.

Referenced by main().

3.18.3.2 void Sharp_Timer_interrupt (void)

Timer ISR for delaying the Sharp GP2D120.

This Timer ISR delays the Sharp GP2D120 and signals its readiness after 50ms via the sharp_ready semaphore. The interrupt is triggered every 25ms. Therefor the routine uses a flag (wait_bit) to signal its second run. The sensor readiness is signalled in the second run of the ISR. The ISR deactivates itself after its second execution.

Definition at line 64 of file sharp.c.

References sharp_ready, SHARP_TIMER_RELOAD_H, SHARP_TIMER_RELOAD_L, and wait_bit.

3.18.3.3 void Sharp_TimerInit (void)

initializes an ISR to signal that the Sharp GP2D120 is ready to be read.

Initializes an ISR to signal that the Sharp GP2D120 is ready to output a valid distance reading. Activates Timer 1 which counts for 25ms twice and sets a readiness signal upon 2nd ISR invocation. (

See also:
sharp_ready)

Definition at line 39 of file sharp.c.

References sharp_ready, SHARP_TIMER_RELOAD_H, SHARP_TIMER_RELOAD_L, and wait_bit.

3.19 timer.c File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
#include "timer.h"
```

Functions

- void Timer_init ()
  activate main timing interrupt to achieve a 1ms timing
- void Timer_interrupt (void)
  keeps time frames for measurements and signals pending measurements in 62ms ticks

Variables

- volatile bit measurement_task
  semaphore to signal a pending sensor measurement
- BYTE timekeeper

3.19.1 Detailed Description

Definition in file timer.c.

3.19.2 Function Documentation

3.19.2.1 void Timer_init (void)

activate main timing interrupt to achieve a 1ms timing

Timer 0 of the T89C51CC02 is used to maintain the timing.

Definition at line 32 of file timer.c.

References measurement_task, timekeeper, TIMER_MILLISECONDS, TIMER_RELOAD_H, and TIMER_RELOAD_L.

Referenced by main().

3.19.2.2 void Timer_interrupt (void)

keeps time frames for measurements and signals pending measurements in 62ms ticks

measurement_task is set after the amount of ms specified in TIMER_MILLISECONDS have passed.

The timer keeps running for the whole duty cycle of the sensor measurement.

Definition at line 56 of file timer.c.

References measurement_task, timekeeper, TIMER_MILLISECONDS, TIMER_RELOAD_H, and TIMER_RELOAD_L.

3.19.3 Variable Documentation

3.19.3.1 volatile bit measurement_task

semaphore to signal a pending sensor measurement

The semaphore is to be set from the main Timer ISR exclusively.

It is cleared after sampling all sensors.

Definition at line 22 of file timer.c.

Referenced by main(), Timer_init(), and Timer_interrupt().

3.19.3.2 BYTE timekeeper

keeps the amount of milliseconds left until another sampling run has to be scheduled

Definition at line 25 of file timer.c.

Referenced by Timer_init(), and Timer_interrupt().

3.20 timer.h File Reference

```
#include "t89c51cc02.h"
#include "datatypes.h"
```

Defines

- #define TIMER_RELOAD_H 0xFA
- #define TIMER_RELOAD_L 0xCB
- #define TIMER_MILLISECONDS 62
  length of sampling intervalls in milliseconds

Functions

- void Timer_init (void)
  activate main timing interrupt to achieve a 1ms timing
- void Timer_interrupt (void) interrupt 1 using 2
  keeps time frames for measurements and signals pending measurements in 62ms ticks

Variables

- volatile bit measurement_task
  semaphore to signal a pending sensor measurement

3.20.1 Detailed Description

Definition in file timer.h.

3.20.2 Define Documentation

3.20.2.1 #define TIMER_MILLISECONDS 62

length of sampling intervalls in milliseconds

This intervall includes sampling _and_ reporting of its values according to the monitoring conditions.

Definition at line 16 of file timer.h.

Referenced by Timer_init(), and Timer_interrupt().

3.20.2.2 #define TIMER_RELOAD_H 0xFA

reload values to get approx. 1ms timing @ 16MHz: 0x10000 - 0x0535 = 0xfacb - HIBYTE

Definition at line 7 of file timer.h.

Referenced by Timer_init(), and Timer_interrupt().

3.20.2.3 #define TIMER_RELOAD_L 0xCB

reload values to get approx. 1ms timing @ 16MHz: 0x10000 - 0x0535 = 0xfacb - LOBYTE

Definition at line 9 of file timer.h.

Referenced by Timer_init(), and Timer_interrupt().

3.20.3 Function Documentation

3.20.3.1 void Timer_init (void)

activate main timing interrupt to achieve a 1ms timing

Timer 0 of the T89C51CC02 is used to maintain the timing.

Definition at line 32 of file timer.c.

References measurement_task, timekeeper, TIMER_MILLISECONDS, TIMER_RELOAD_H, and TIMER_RELOAD_L.

Referenced by main().

3.20.3.2 void Timer_interrupt (void)

keeps time frames for measurements and signals pending measurements in 62ms ticks

measurement_task is set after the amount of ms specified in TIMER_MILLISECONDS have passed.

The timer keeps running for the whole duty cycle of the sensor measurement.

Definition at line 56 of file timer.c.

References measurement_task, timekeeper, TIMER_MILLISECONDS, TIMER_RELOAD_H, and TIMER_RELOAD_L.

3.20.4 Variable Documentation

3.20.4.1 volatile bit measurement_task

semaphore to signal a pending sensor measurement

The semaphore is to be set from the main Timer ISR exclusively.

It is cleared after sampling all sensors.

Definition at line 22 of file timer.c.

Referenced by main(), Timer_init(), and Timer_interrupt().

Index