Simulated Timer, Watchdog and Schdeuler (template)

1.0

Generated by Doxygen 1.9.1

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 sTCconf_t Struct Reference

struct for timer counter internal configuration Individual TC configuration

Data Fields

- FpISR_t pfpISR_TC
  address of watchdog ISR Timer function, init to NULL.
- uint16_t TC_reg
  TC compare register.
- uint16_t TC_cnt
  count TC register
- uint16_t TC_capt [2]
  capture registers for later use
- uint8_t TC_sts
  status: 0: not started, 1: TC active

3.1.1 Detailed Description

struct for timer counter internal configuration Individual TC configuration

Date

2022-01-11

Definition at line 22 of file TCsim.c.

3.1.2 Field Documentation

3.1.2.1 pfpISR_TC

```
FpISR_t sTCconf_t::pfpISR_TC
```

address of watchdog ISR Timer function, init to NULL.

See also

FpISR_t

Definition at line 23 of file TCsim.c.

Referenced by TC1config(), TC2config(), TC3config(), and tc_Thread().

3.1.2.2 TC_capt

```
uint16_t sTCconf_t::TC_capt[2]
```

capture registers for later use

Definition at line 28 of file TCsim.c.

3.1.2.3 TC_cnt

```
uint16_t sTCconf_t::TC_cnt
```

count TC register

Definition at line 26 of file TCsim.c.

Referenced by tc_Thread().

3.1.2.4 TC_reg

```
uint16_t sTCconf_t::TC_reg
```

TC compare register.

Definition at line 25 of file TCsim.c.

Referenced by TC1config(), TC2config(), TC3config(), and tc_Thread().

3.1.2.5 TC_sts

```
uint8_t sTCconf_t::TC_sts
```

status: 0: not started, 1: TC active

Definition at line 30 of file TCsim.c.

Referenced by TC1config(), TC1stop(), TC2config(), TC2stop(), TC3config(), TC3stop(), and tc_Thread().

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

- TCsim.c

3.2 sTCint_t Struct Reference

struct for timer counter thread configuration TC internal configuration

Data Fields

- pthread_t threadID_TC
  thread id timer counter
- uint8_t TC_init
  TC initialized. 0: not started, 1: thread running.

3.2.1 Detailed Description

struct for timer counter thread configuration TC internal configuration

Date

2022-01-11

Definition at line 40 of file TCsim.c.

3.2.2 Field Documentation

3.2.2.1 TC_init

```
uint8_t sTCint_t::TC_init
```

TC initialized. 0: not started, 1: thread running.

Definition at line 43 of file TCsim.c.

Referenced by tc_Thread(), TCengine_start(), and TCengine_stop().

3.2.2.2 threadID_TC

```
pthread_t sTCint_t::threadID_TC
```

thread id timer counter

Definition at line 41 of file TCsim.c.

Referenced by TCengine_start().

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

- TCsim.c

3.3 sTskLst_t Struct Reference

Typedef for simple task list Simple list for tasks to be scheduled and dispatched.

```
#include <schedule.h>
```

Data Fields

- FpTsk_t fpTask
  pointer to task function
- enum eSts sts
  task status.
- uint32_t period
  scheduling period, e.g. in ms
- uint32_t elapsed
  elapsed time to period and next execution

3.3.1 Detailed Description

Typedef for simple task list Simple list for tasks to be scheduled and dispatched.

Definition at line 33 of file schedule.h.

3.3.2 Field Documentation

3.3.2.1 elapsed

```
uint32_t sTskLst_t::elapsed
```

elapsed time to period and next execution

Definition at line 37 of file schedule.h.

3.3.2.2 fpTask

`FpTsk_t sTskLst_t::fpTask`

pointer to task function

Definition at line 34 of file schedule.h.

3.3.2.3 period

`uint32_t sTskLst_t::period`

scheduling period, e.g. in ms

Definition at line 36 of file schedule.h.

3.3.2.4 sts

`enum eSts sTskLst_t::sts`

task status.

See also

eSts

Definition at line 34 of file schedule.h.

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

- schedule.h

3.4 sWDint_t Struct Reference

struct for watchdog internal configuration Keyboard thread configuration

Data Fields

- pthread_t threadID_WD
  thread id watchdog
- FpISR_t pfpISR_WD
  address of watchdog ISR Timer function, init to NULL.
- uint16_t WD_reg
  watchdog compare register
- uint16_t WD_cnt
  count WD register
- unsigned int: 7
  reserved, not used
- unsigned int WD_init: 1
  watchdog initialized. 0: not started, 1: thread running

3.4.1 Detailed Description

struct for watchdog internal configuration Keyboard thread configuration

Date

2022-01-09

Definition at line 19 of file watchdog.c.

3.4.2 Field Documentation

3.4.2.1 int

`unsigned sWDint_t::int`

reserved, not used

Definition at line 26 of file watchdog.c.

3.4.2.2 pfpISR_WD

`FpISR_t sWDint_t::pfpISR_WD`

address of watchdog ISR Timer function, init to NULL.

See also

FpISR_t

Definition at line 21 of file watchdog.c.

Referenced by WATCHDOG_Initialize(), and wd_Thread().

3.4.2.3 threadID_WD

`pthread_t sWDint_t::threadID_WD`

thread id watchdog

Definition at line 20 of file watchdog.c.

Referenced by WATCHDOG_Initialize().

3.4.2.4 WD_cnt

```
uint16_t sWDint_t::WD_cnt
```

count WD register

Definition at line 24 of file watchdog.c.

Referenced by WATCHDOG_Initialize(), WATCHDOG_Reset(), and wd_Thread().

3.4.2.5 WD_init

```
unsigned int sWDint_t::WD_init
```

watchdog initialized. 0: not started, 1: thread running

Definition at line 28 of file watchdog.c.

Referenced by WATCHDOG_Initialize(), and WATCHDOG_Reset().

3.4.2.6 WD_reg

```
uint16_t sWDint_t::WD_reg
```

watchdog compare register

Definition at line 23 of file watchdog.c.

Referenced by WATCHDOG_Initialize(), and wd_Thread().

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

- watchdog.c

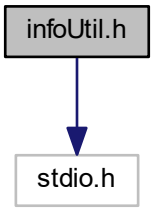
Chapter 4

File Documentation

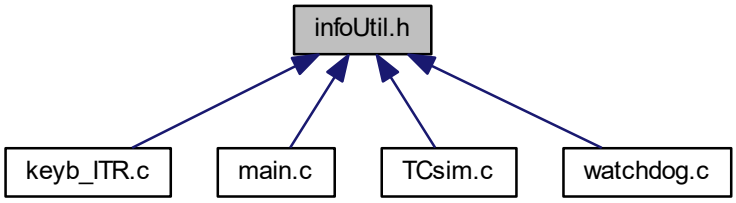
4.1 infoUtil.h File Reference

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

Include dependency graph for infoUtil.h:



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



Macros

- #define SWINFO(proj, auth) printf("This is project >%s<, author %s, compiled on %s %s\n", proj, auth, __DATE__, __TIME__)
- #define MSG(cat, msg, proc) printf("%c: %s (%s, l:%d)\n", cat, msg, proc, __LINE__)
- #define MSG_F(cat, msg, proc) printf("%c: %s (%s:%s, l:%d)\n", cat, msg, __FILE__, proc, __LINE__)
- #define MSGF(cat, msg, proc) printf("%c: %s (%s:%s, l:%d)\n", cat, msg, _FILE_, proc, __LINE__)

4.1.1 Macro Definition Documentation

4.1.1.1 MSG

```
#define MSG(
              cat,
              msg,
              proc )  printf( "%c:  %s (%s, l:%d)\n", cat, msg, proc, __LINE__ )
```

Macro: output message with line info. Implements printf statement

Parameters

cat	message category as char like I, W, E, F as 'E' e.g.
msg	human readabel message as string
proc	name of procedure as string, e.g. "Main"

Author

R. S. Mayer

Definition at line 24 of file infoUtil.h.

4.1.1.2 MSG_F

```
#define MSG_F(
              cat,
              msg,
              proc )  printf( "%c:  %s (%s:%s, l:%d)\n", cat, msg, __FILE__, proc, __LINE__ )
```

Macro: output message with line info. Implements printf statement

Parameters

cat	message category as char like I, W, E, F as 'E' e.g.
msg	human readabel message as string
proc	name of procedure as string, e.g. "Main"

Author

R. S. Mayer

Definition at line 35 of file infoUtil.h.

4.1.1.3 MSGF

```
#define MSGF(
              cat,
              msg,
              proc )  printf( "%c:  %s (%s:%s, l:%d)\n", cat, msg, _FILE_, proc, __LINE__ )
```

Macro: output message with line info. Implements printf statement Need definition of

See also

FILE

Parameters

cat	message category as char like I, W, E, F as 'E' e.g.
msg	human readabel message as string
proc	name of procedure as string, e.g. "Main"

Author

R. S. Mayer

Definition at line 48 of file infoUtil.h.

4.1.1.4 SWINFO

```
#define SWINFO(
              proj,
              auth )  printf( "This is project >%s<, author %s, compiled on %s %s\n", proj,
auth, __DATE__, __TIME__ )
```

Macro: Output SW and compile info. Implements printf statement

Parameters

proj	project name as string
auth	author as string

Author

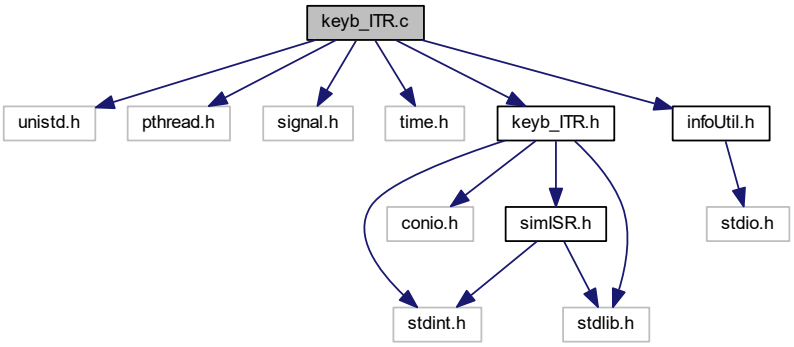
R. S. Mayer

Definition at line 13 of file infoUtil.h.

4.2 keyb_ITR.c File Reference

```
#include <unistd.h>
#include <pthread.h>
#include <signal.h>
#include <time.h>
#include "keyb_ITR.h"
#include "infoUtil.h"
```

Include dependency graph for keyb_ITR.c:



Macros

- #define _FILE_ "keyb_ITR.c"

  file name for macros in infoUtil.h
- #define _SUS_ (1000)

  sleep micro(u) seconds in threads. Comment if not to be used

Functions

- void regISR_Keyboard (FpISR_t fpISR)

  Register ISR function keyboard.
- void * keyboard_Thread (void *vargp)

  Keyboard watch thread with POSIX Watch on kbhit().
- void Start_Keyboard ()

  Start Keyboard thread Starts the POSIX thread.
- void Stop_Keyboard ()

  Stop Keyboard thread Stops the POSIX thread.

Variables

- static FpISR_t fpISR_keyb = 0

  address of ISR for keyboard handling, init to NULL.
- static pthread_t threadIDkeyboard

  thread id keyboard_Thread
- static uint8_t KEYB_state = 0

  thread state. 0: not started or to be terminated, 1: started and running

4.2.1 Macro Definition Documentation

4.2.1.1 _FILE_

```
#define _FILE_ "keyb_ITR.c"
```

file name for macros in infoUtil.h

Definition at line 11 of file keyb_ITR.c.

4.2.1.2 _SUS_

```
#define _SUS_ (1000)
```

sleep micro(u) seconds in threads. Comment if not to be used

Definition at line 13 of file keyb_ITR.c.

4.2.2 Function Documentation

4.2.2.1 keyboard_Thread()

```
void* keyboard_Thread (
            void * vargp )
```

Keyboard watch thread with POSIX Watch on kbhit().

See also

fpISR_keyb

Author

R. S. Mayer

Date

2022-01-09

Definition at line 36 of file keyb_ITR.c.

```
37 {
38     while (1) {                     // forever ...
39 #ifdef _SUS_
40         usleep(_SUS_);              // some wait
41 #endif // _SUS_
42         if (!KEYB_state)            // to be terminated
43             break;                  // end loop an terminate
44         if ( kbhit() ) {
45             if ( fpISR_keyb ) {     // ISR function pointer set
46                 (*fpISR_keyb)();    // callback ISR
47             }
48 //            else {                // temporary implementation
49 //                _Keyb_c_ = getch(); // read char
50 //            }
51         } // end kbhit
52     } // end while
53     MSGF('I',"Thread terminating","keyboard_Thread");
54     pthread_cancel(pthread_self()); // cancel this thread
55     return NULL;
56 }
```

References _SUS_, fpISR_keyb, KEYB_state, and MSGF.

Referenced by Start_Keyboard().

Here is the caller graph for this function:

4.2.2.2 regISR_Keyboard()

```
void regISR_Keyboard (
            FpISR_t fpISR )
```

Register ISR function keyboard.

Parameters

fpISR	pointer to ISR function

Definition at line 25 of file keyb_ITR.c.

```
25                                          {
26     fpISR_keyb = fpISR;
27 }
```

References fpISR_keyb.

Referenced by main().

Here is the caller graph for this function:

4.2.2.3 Start_Keyboard()

```
void Start_Keyboard ( )
```

Start Keyboard thread Starts the POSIX thread.

Keyboard watch thread with POSIX.

Definition at line 62 of file keyb_ITR.c.

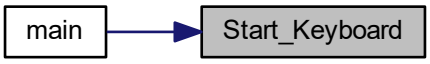
```
62                          {
63     if ( fpISR_keyb == 0 ) {
64         MSGF('E',"No ISR registered! Not started","Start_Keyboard");
65         return;
66     }
67     int err;
68     err = pthread_create(&threadIDkeyboard, NULL, keyboard_Thread, NULL);
69     if (err != 0) {
70         MSGF('E',"Thread 'keyboard_Thread' NOT started","Start_Keyboard");
71     } else {
72         MSGF('I',"Thread 'keyboard_Thread' started","Start_Keyboard");
73     }
74     KEYB_state = 1;      // set state flag running
75 }
```

References fpISR_keyb, KEYB_state, keyboard_Thread(), MSGF, and threadIDkeyboard.

Referenced by main().

Here is the call graph for this function:

Here is the caller graph for this function:



4.2.2.4 Stop_Keyboard()

```
void Stop_Keyboard ( )
```

Stop Keyboard thread Stops the POSIX thread.

Stop Keyboard thread.

Definition at line 81 of file keyb_ITR.c.

```
81                      {
82     KEYB_state = 0;      // to be terminated
83     MSGF('I',"Thread 'keyboard_Thread' stopped","Stop_Keyboard");
84 }
```

References KEYB_state, and MSGF.

4.2.3 Variable Documentation

4.2.3.1 fpISR_keyb

```
FpISR_t fpISR_keyb = 0  [static]
```

address of ISR for keyboard handling, init to NULL.

See also

FpISR_t

Definition at line 16 of file keyb_ITR.c.

Referenced by keyboard_Thread(), regISR_Keyboard(), and Start_Keyboard().

4.2.3.2 KEYB_state

```
uint8_t KEYB_state = 0  [static]
```

thread state. 0: not started or to be terminated, 1: started and running

Definition at line 18 of file keyb_ITR.c.

Referenced by keyboard_Thread(), Start_Keyboard(), and Stop_Keyboard().

4.2.3.3 threadIDkeyboard

```
pthread_t threadIDkeyboard  [static]
```

thread id keyboard_Thread

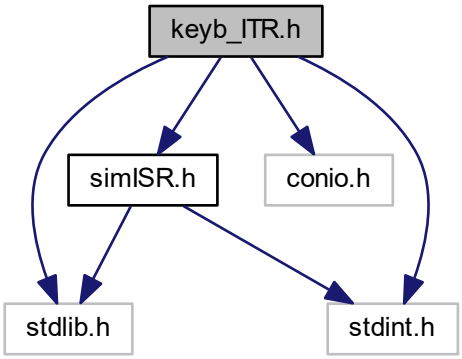
Definition at line 17 of file keyb_ITR.c.

Referenced by Start_Keyboard().

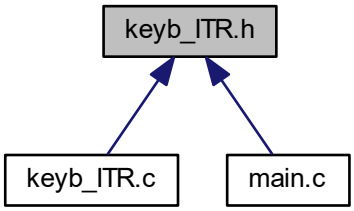
4.3 keyb_ITR.h File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <conio.h>
#include "simISR.h"
```

Include dependency graph for keyb_ITR.h:



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



Functions

- void regISR_Keyboard (FpISR_t fpISR)

  Register ISR function keyboard.
- void Start_Keyboard ()

  Keyboard watch thread with POSIX.
- void Stop_Keyboard ()

  Stop Keyboard thread.

Variables

- char _Keyb_c_

  keyboard char read only if no ISR registered

4.3.1 Function Documentation

4.3.1.1 regISR_Keyboard()

```
void regISR_Keyboard (
            FpISR_t fpISR )
```

Register ISR function keyboard.

Parameters

fpISR	pointer to ISR function

Definition at line 25 of file keyb_ITR.c.

```
25                                          {
26     fpISR_keyb = fpISR;
```

```
27 }
```

References fpISR_keyb.

Referenced by main().

Here is the caller graph for this function:

4.3.1.2 Start_Keyboard()

```
void Start_Keyboard ( )
```

Keyboard watch thread with POSIX.

Start Keyboard thread

Keyboard watch thread with POSIX.

Definition at line 62 of file keyb_ITR.c.

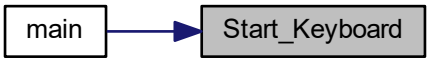
```
62                          {
63     if ( fpISR_keyb == 0 ) {
64         MSGF('E',"No ISR registered! Not started","Start_Keyboard");
65         return;
66     }
67     int err;
68     err = pthread_create(&threadIDkeyboard, NULL, keyboard_Thread, NULL);
69     if (err != 0) {
70         MSGF('E',"Thread 'keyboard_Thread' NOT started","Start_Keyboard");
71     } else {
72         MSGF('I',"Thread 'keyboard_Thread' started","Start_Keyboard");
73     }
74     KEYB_state = 1;     // set state flag running
75 }
```

References fpISR_keyb, KEYB_state, keyboard_Thread(), MSGF, and threadIDkeyboard.

Referenced by main().

Here is the call graph for this function:

Here is the caller graph for this function:



4.3.1.3 Stop_Keyboard()

```
void Stop_Keyboard ( )
```

Stop Keyboard thread.

Stop Keyboard thread.

Definition at line 81 of file keyb_ITR.c.

```
81                      {
82     KEYB_state = 0;      // to be terminated
83     MSGF('I',"Thread 'keyboard_Thread' stopped","Stop_Keyboard");
84 }
```

References KEYB_state, and MSGF.

4.3.2 Variable Documentation

4.3.2.1 _Keyb_c_

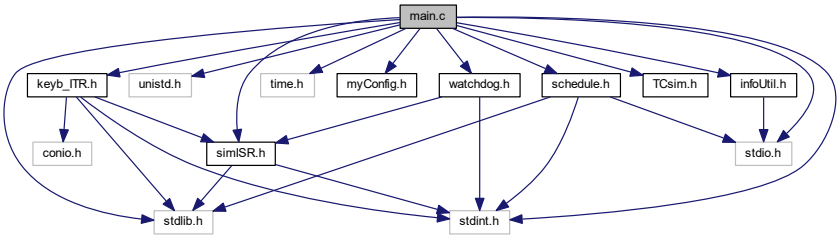
```
char _Keyb_c_  [extern]
```

keyboard char read only if no ISR registered

4.4 main.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>
#include <time.h>
#include "myConfig.h"
#include "simISR.h"
#include "keyb_ITR.h"
#include "TCsim.h"
#include "infoUtil.h"
#include "schedule.h"
#include "watchdog.h"
```

Include dependency graph for main.c:



Macros

- #define _FILE_ "main.c"

  file name for macros in infoUtil.h

Functions

- ISR_t myKeyboard_ISR ()

  Service Routine for simulated interrupt ISR for Keyboard.
- task_t task1 (uint32_t input)
- void schedule ()
- void dispatch ()
- ISR_t myTC1_ISR ()

  Service Routine for simulated interrupt ISR for TC 2.
- int main ()

  Template MPS, Termin 4 mit volatile, Scheduler und Watchdog.

Variables

- static volatile char cKB = 0

  character read in
- static uint8_t bIsrTC1 = 0

  flag for ISR TC1 timer

4.4.1 Macro Definition Documentation

4.4.1.1 _FILE_

```
#define _FILE_ "main.c"
```

file name for macros in infoUtil.h

Definition at line 16 of file main.c.

4.4.2 Function Documentation

4.4.2.1 dispatch()

```
void dispatch ( )
```

Dispatch (execute) ready task in scheduler

See also

sTskLst_t

Attention

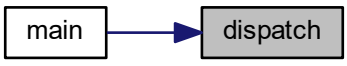
Muss implementiert werden!

Definition at line 65 of file main.c.

```
65                  {
66      // »> TODO «<
67  }
```

Referenced by main().

Here is the caller graph for this function:



4.4.2.2 main()

```
int main ( )
```

Template MPS, Termin 4 mit volatile, Scheduler und Watchdog.

Author

R. S. Mayer

Date

2021-06-02

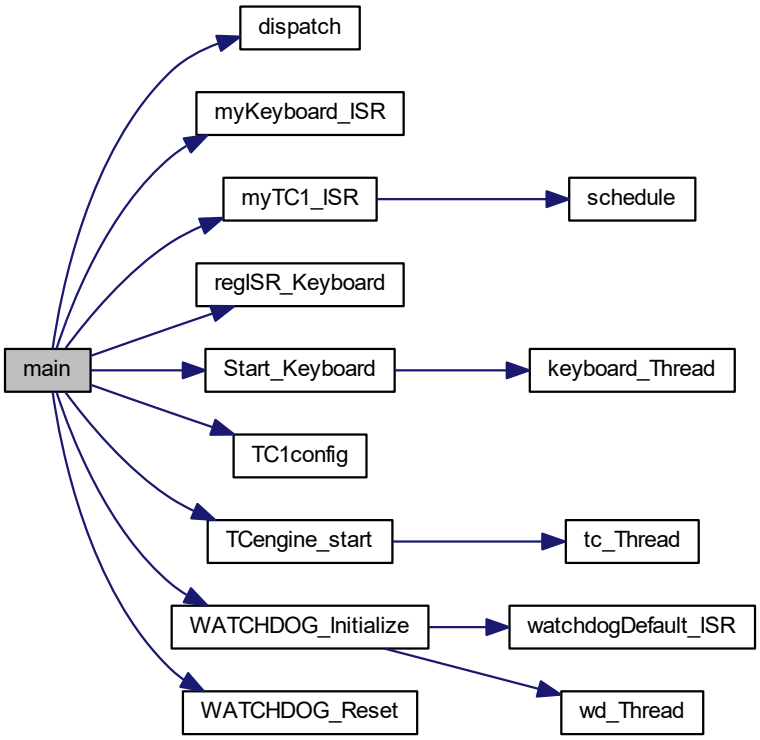
Definition at line 87 of file main.c.

```
88 {
89     SWINFO( "MPS21-Prakt_5", _AUTHOR_ );     // info about project
90     MSGF( 'I', "Starting ...", "main" );    // a line info
91 
92     /*
93      * +++ initialisation +++
94      */
95     // keyboard
96     regISR_Keyboard( myKeyboard_ISR );  // register Keyboard interrupt
97     Start_Keyboard();                   // start
98 
99     // Timer for Scheduler
100     TC1config( myTC1_ISR, 100 );        // config timer 1, ISR, 100 ms
101     TCengine_start();                   // start simulated timer
102 
103     MSGF( 'I', "»» please press 'h' or 'H' for Help ««", "main" );  // a line info
104 
105     /*
106      * +++ Start Watchdog +++
107      * TODO: testen, dann erst einmal auskommentieren!
108      */
109     printf( "\n" );     // clear line
110     MSGF( 'I', "Start Watchdog »»Use key 'r' or program dies within 8 sec!««", "main" );  // a line info
111     printf(    "                    ------------------------------------------------\n\n" );     // clear
        line
112     WATCHDOG_Initialize( WDT_TIMEOUT_8000 );   // activate watchdog
113 
114     /*
115      *  +++ endless loop +++
116      */
117     uint8_t run = 1;
118     do {                                // +++ endless loop
119         /* +++ handle tc interrupts +++ */
120         // (if any)
121 
122         /* +++ handle keyboard +++      */
123         if (cKB) {
124             // +++ reset watchdog +++
125             WATCHDOG_Reset();           // on any keyboard
126 
127             switch (cKB) {
128                 case 'r':               // reset watchdog
129                     WATCHDOG_Reset();
130                     MSGF( 'I', "Watchdog reseted", "main" );    // a line info
131                     break;
132                 case 'q':               // quit
133                     run = 0;
134                     break;
135                 case 'h':               // help
136                 case 'H':               // help
137                     printf("\nHelp:\n\tH,h: help\n\tq: quit\n\tr: WD reset\n");
138                     break;
139                 default:
140                     printf("[%c]", cKB);
141                     break;
142             }
143 
144             cKB = 0;                    // don't forget!
145         } // if cKB
146 
147         /* +++ dispatch scheduled tasks +++ */
148         dispatch();                     // »» TODO «« implement scheduler!
```

```
149
150     } while (run);                      // --- end loop ---
151
152 //    sleep(1);           // wait 1 sec
153     usleep(500000);                     // wait 500 ms
154
155     MSGF( 'I', "... terminating", "main" );    // a line info
156     return 0;
157 }
```

References _AUTHOR_, cKB, dispatch(), MSGF, myKeyboard_ISR(), myTC1_ISR(), regISR_Keyboard(), Start_↩ Keyboard(), SWINFO, TC1config(), TCengine_start(), WATCHDOG_Initialize(), WATCHDOG_Reset(), and WDT↩ _TIMEOUT_8000.

Here is the call graph for this function:



4.4.2.3 myKeyboard_ISR()

```
ISR_t myKeyboard_ISR ( )
```

Service Routine for simulated interrupt ISR for Keyboard.

< read from keyboard

Definition at line 25 of file main.c.

```
25                      {
26     cKB = getch();              //!< read from keyboard
```

```
27 }
```

References cKB.

Referenced by main().

Here is the caller graph for this function:

4.4.2.4 myTC1_ISR()

```
ISR_t myTC1_ISR ( )
```

Service Routine for simulated interrupt ISR for TC 2.

Definition at line 74 of file main.c.

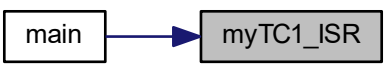
```
74                  {
75     schedule();                          // »> TODO «< implementieren!
76     static int ts;
77     ts++;
78     if ( (ts % 10) == 0 )
79         printf( "." );
80 }
```

References schedule().

Referenced by main().

Here is the call graph for this function:

Here is the caller graph for this function:



4.4.2.5 schedule()

```
void schedule ( )
```

Schedule after each timer event. Set waiting task(s) to ready, when period is reached

See also

sTskLst_t

Attention

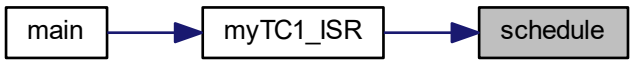
Muss implementiert werden!

Definition at line 56 of file main.c.

```
56                  {
57      // »> TODO «<
58  }
```

Referenced by myTC1_ISR().

Here is the caller graph for this function:



4.4.2.6 task1()

```
task_t task1 (
              uint32_t input )
```

Example task 1

Parameters

input	default 0

Returns

out of eRet

See also

eRet

Definition at line 37 of file main.c.

```
37                                  {
38     printf( "[Task 1], param: %d\n ", input );
39     return SUCCESS;
40 }
```

References SUCCESS.

4.4.3 Variable Documentation

4.4.3.1 bIsrTC1

```
uint8_t bIsrTC1 = 0  [static]
```

flag for ISR TC1 timer

Definition at line 69 of file main.c.

4.4.3.2 cKB

```
volatile char cKB = 0  [static]
```

character read in

See also

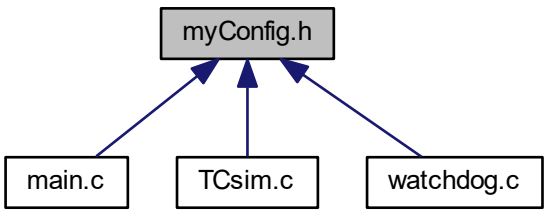
myKeyboard_ISR

Definition at line 20 of file main.c.

Referenced by main(), and myKeyboard_ISR().

4.5 myConfig.h File Reference

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



Macros

- #define MY_CORR_CLOCKS_PER_MS (1)
- #define _AUTHOR_ "Nick Otin and Marie O'hana"

  define here YOUR name(s)

4.5.1 Macro Definition Documentation

4.5.1.1 _AUTHOR_

```
#define _AUTHOR_ "Nick Otin and Marie O'hana"
```

define here YOUR name(s)

Definition at line 17 of file myConfig.h.

4.5.1.2 MY_CORR_CLOCKS_PER_MS

```
#define MY_CORR_CLOCKS_PER_MS (1)
```

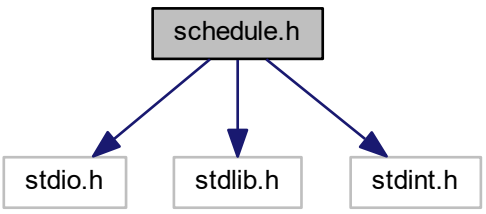
Correction clocks per milliseconds from <time.h>. Must be int and >=1. Adaptation may be needed for systems other than on Windows with clocks faster than 1000/s.

Definition at line 14 of file myConfig.h.

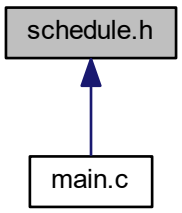
4.6 schedule.h File Reference

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

Include dependency graph for schedule.h:



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



Data Structures

- struct sTskLst_t

  Typedef for simple task list Simple list for tasks to be scheduled and dispatched.

Typedefs

- typedef uint32_t task_t

  Function typedef task_t Type of task for scheduler. For pointer to function.
- typedef task_t(∗ FpTsk_t) (uint32_t)

  function pointer to task_t (ATTENTION: 64bit!).

Enumerations

- enum eSts { WAIT , RDY , RUN , TERM }
- enum eRet { SUCCESS , WARN , ERROR }

4.6.1 Typedef Documentation

4.6.1.1 FpTsk_t

```
typedef task_t(* FpTsk_t) (uint32_t)
```

function pointer to task_t (ATTENTION: 64bit!).

See also

task_t

Definition at line 15 of file schedule.h.

4.6.1.2 task_t

```
typedef uint32_t task_t
```

Function typedef task_t Type of task for scheduler. For pointer to function.

See also

FpISR_t

Definition at line 13 of file schedule.h.

4.6.2 Enumeration Type Documentation

4.6.2.1 eRet

```
enum eRet
```

Enumerator

SUCCESS	
WARN	
ERROR	

Definition at line 27 of file schedule.h.

```
27 { SUCCESS, WARN, ERROR } ;    //!< possible task return stati.
```

4.6.2.2 eSts

```
enum eSts
```

Possible task stati. waiting, ready-to-run, running, terminated

Enumerator

WAIT	waiting for period time elapsed -> RDY
RDY	ready for execution on next dispatch. -> RUN
RUN	running. After -> WAIT
TERM	terminated. Currently not used

Definition at line 20 of file schedule.h.

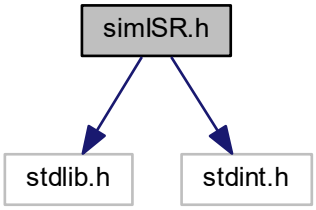
```
20          {
21     WAIT,       //!< waiting for period time elapsed -> RDY
22     RDY,        //!< ready for execution on next dispatch. -> RUN
```

```
23     RUN,         //!< running. After -> WAIT
24     TERM         //!< terminated. Currently not used
25 };
```

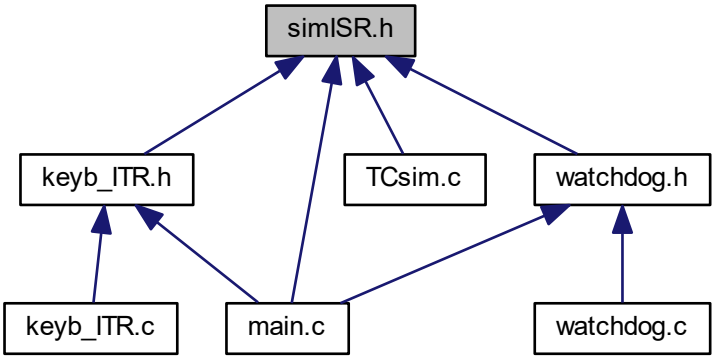
4.7 simISR.h File Reference

```
#include <stdlib.h>
#include <stdint.h>
```

Include dependency graph for simISR.h:



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



Typedefs

- typedef void ISR_t

  function typedef ISR Type of Interrupt Service Routine (ISR) as simulation for callback on event. For pointer to function
- typedef ISR_t(∗ FpISR_t) ()

  function pointer to ISR_t (ATTENTION: 64bit!).

4.7.1 Typedef Documentation

4.7.1.1 FpISR_t

```
typedef ISR_t(* FpISR_t) ()
```

function pointer to ISR_t (ATTENTION: 64bit!).

See also

ISR_t

Definition at line 14 of file simISR.h.

4.7.1.2 ISR_t

```
typedef void ISR_t
```

function typedef ISR Type of Interrupt Service Routine (ISR) as simulation for callback on event. For pointer to function

See also

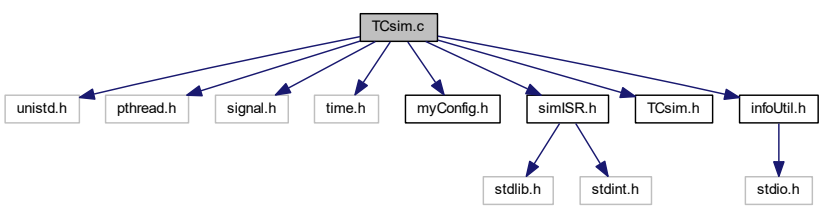
FpISR_t

Definition at line 12 of file simISR.h.

4.8 TCsim.c File Reference

```
#include <unistd.h>
#include <pthread.h>
#include <signal.h>
#include <time.h>
#include "myConfig.h"
#include "simISR.h"
#include "TCsim.h"
#include "infoUtil.h"
```

Include dependency graph for TCsim.c:



Data Structures

- struct sTCconf_t

  struct for timer counter internal configuration Individual TC configuration
- struct sTCint_t

  struct for timer counter thread configuration TC internal configuration

Macros

- #define _FILE_ "TCsim.c"

  file name for macros in infoUtil.h
- #define VERBOSE

  for verbose output

Functions

- void * tc_Thread (void *vargp)

  Watchdog thread with POSIX Watch on timeout.
- void TCengine_start ()

  start TC engine Start timer counter engine thread
- void TCengine_stop ()

  stop TC engine Stop timer counter engine thread. Stops all timer counters.
- void TC1config (FpISR_t fpISR, uint16_t tc)

  config TC1 Configuration of TC1
- void TC2config (FpISR_t fpISR, uint16_t tc)

  config TC2 Configuration of TC2
- void TC3config (FpISR_t fpISR, uint16_t tc)

  config TC3 Configuration of TC3
- void TC1stop ()

  Stop TC1 Stop TC1, disables handling in TC engine. Keeps previous configuration.
- void TC2stop ()

  Stop TC2 Stop TC2, disables handling in TC engine. Keeps previous configuration.
- void TC3stop ()

  Stop TC3 Stop TC3, disables handling in TC engine. Keeps previous configuration.

Variables

- static sTCint_t sTCint = {0}

  TC thread ctrl struct instance.
- static sTCconf_t sTC1 = {0}

  TC1 ctrl struct instance.
- static sTCconf_t sTC2 = {0}

  TC2 ctrl struct instance.
- static sTCconf_t sTC3 = {0}

  TC3 ctrl struct instance.

4.8.1 Macro Definition Documentation

4.8.1.1 _FILE_

```
#define _FILE_ "TCsim.c"
```

file name for macros in infoUtil.h

Definition at line 12 of file TCsim.c.

4.8.1.2 VERBOSE

```
#define VERBOSE
```

for verbose output

Definition at line 15 of file TCsim.c.

4.8.2 Function Documentation

4.8.2.1 TC1config()

```
void TC1config (
            FpISR_t fpISR,
            uint16_t tc )
```

config TC1 Configuration of TC1

config TC1

Parameters

fpISR	pointer to ISR
tc	timer register 0 < tc <= TC_MAX_COUNT

Definition at line 162 of file TCsim.c.

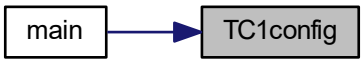
```
162                                                 {
163     if ( fpISR != 0 ) {          // check pointer
164         sTC1.pfpISR_TC = fpISR;
165     }
166     else {
167         MSGF('E',"Null pointer to ISR","TC1config");
168         return;                  // leave function
169     }
170
171     if ( (tc == 0) || (tc > TC_MAX_COUNT) ) {          // check pointer
172         MSGF('E',"tc register not within limits","TC1config");
173         return;                  // leave function
174     }
175     else {
176        sTC1.TC_reg = tc;
177     }
178
```

```
179     sTC1.TC_sts = 1;            // TC successfully initialized
180 #ifdef VERBOSE
181         MSGF('I',"TC1 successfully initilaized","TC1config");
182 #endif // VERBOSE
183 }
```

References MSGF, sTCconf_t::pfpISR_TC, sTC1, TC_MAX_COUNT, sTCconf_t::TC_reg, and sTCconf_t::TC_sts.

Referenced by main().

Here is the caller graph for this function:



4.8.2.2 TC1stop()

```
void TC1stop ( )
```

Stop TC1 Stop TC1, disables handling in TC engine. Keeps previous configuration.

Stop TC1.

Definition at line 245 of file TCsim.c.

```
245                 {
246     sTC1.TC_sts = 0;
247 }
```

References sTC1, and sTCconf_t::TC_sts.

4.8.2.3 TC2config()

```
void TC2config (
            FpISR_t fpISR,
            uint16_t tc )
```

config TC2 Configuration of TC2

config TC2

Parameters

fpISR	pointer to ISR
tc	timer register 0 < tc <= TC_MAX_COUNT

Definition at line 190 of file TCsim.c.

```
190                                             {
191     if ( fpISR != 0 ) {             // check pointer
192         sTC2.pfpISR_TC = fpISR;
193     }
194     else {
195         MSGF('E',"Null pointer to ISR","TC2config");
196         return;                     // leave function
197     }
198
199     if ( (tc == 0) || (tc > TC_MAX_COUNT) ) {         // check pointer
200         MSGF('E',"tc register not within limits","TC2config");
201         return;                     // leave function
202     }
203     else {
204        sTC2.TC_reg = tc;
205     }
206
207     sTC2.TC_sts = 1;                // TC successfully initialized
208 #ifdef VERBOSE
209         MSGF('I',"TC2 successfully initilaized","TC2config");
210 #endif // VERBOSE
211 }
```

References MSGF, sTCconf_t::pfpISR_TC, sTC2, TC_MAX_COUNT, sTCconf_t::TC_reg, and sTCconf_t::TC_sts.

4.8.2.4 TC2stop()

```
void TC2stop ( )
```

Stop TC2 Stop TC2, disables handling in TC engine. Keeps previous configuration.

Stop TC2.

Definition at line 253 of file TCsim.c.

```
253                 {
254     sTC2.TC_sts = 0;
255 }
```

References sTC2, and sTCconf_t::TC_sts.

4.8.2.5 TC3config()

```
void TC3config (
              FpISR_t fpISR,
              uint16_t tc )
```

config TC3 Configuration of TC3

config TC3

Parameters

fpISR	pointer to ISR
tc	timer register 0 < tc <= TC_MAX_COUNT

Definition at line 218 of file TCsim.c.

```
218                                             {
219     if ( fpISR != 0 ) {          // check pointer
220         sTC3.pfpISR_TC = fpISR;
221     }
222     else {
223         MSGF('E',"Null pointer to ISR","TC3config");
224         return;                  // leave function
225     }
226
227     if ( (tc == 0) || (tc > TC_MAX_COUNT) ) {          // check pointer
228         MSGF('E',"tc register not within limits","TC3config");
229         return;                  // leave function
230     }
231     else {
232        sTC3.TC_reg = tc;
233     }
234
235     sTC3.TC_sts = 1;             // TC successfully initialized
236 #ifdef VERBOSE
237         MSGF('I',"TC3 successfully initilaized","TC3config");
238 #endif // VERBOSE
239 }
```

References MSGF, sTCconf_t::pfpISR_TC, sTC3, TC_MAX_COUNT, sTCconf_t::TC_reg, and sTCconf_t::TC_sts.

4.8.2.6 TC3stop()

```
void TC3stop ( )
```

Stop TC3 Stop TC3, disables handling in TC engine. Keeps previous configuration.

Stop TC3.

Definition at line 261 of file TCsim.c.

```
261                 {
262     sTC3.TC_sts = 0;
263 }
```

References sTC3, and sTCconf_t::TC_sts.

4.8.2.7 tc_Thread()

```
void* tc_Thread (
             void * vargp )
```

Watchdog thread with POSIX Watch on timeout.

See also

fpISR_keyb

Waiting with *SUS* e.g. 10000 us = 1 ms improves performance of system output, e.q. printf(...) significantly!

See also

SUS

Author

R. S. Mayer

Date

2022-01-11

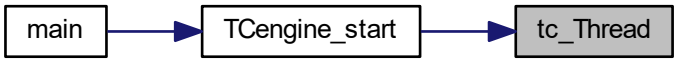
Definition at line 64 of file TCsim.c.

```
65 {
66     MSGF('I',"Thread 'tc_Thread' starts ...","tc_Thread");
67 
68     clock_t start = clock() / MY_CORR_CLOCKS_PER_MS; // system clock (adapt to used system)
69     clock_t ticks;
70     while (1) {                          // forever ...
71         // check engine shall still run
72         if ( !sTCint.TC_init ) {
73             // terminate thread
74             break;
75         }
76         // wait for next tick(s)
77         while ( (ticks = (clock() / MY_CORR_CLOCKS_PER_MS - start)) == 0 ) {
78 #ifdef _SUS_
79             usleep(_SUS_);                  // some wait
80 #endif // _SUS_                                          // do nothing but wait
81         }  // time elapsed
82 
83         /* handle timer(s) */
84         // TC 1
85         if (sTC1.TC_sts) {  // timer 1 active
86             sTC1.TC_cnt += ticks;
87             if ( sTC1.TC_cnt >= sTC1.TC_reg ) { // counter reaches register
88                 (*sTC1.pfpISR_TC)();            // call ISR
89                 sTC1.TC_cnt = 0;                // reset counter
90             }
91         }
92         // TC 2
93         if (sTC2.TC_sts) {  // timer 1 active
94             sTC2.TC_cnt += ticks;
95             if ( sTC2.TC_cnt >= sTC2.TC_reg ) { // counter reaches register
96                 (*sTC2.pfpISR_TC)();            // call ISR
97                 sTC2.TC_cnt = 0;                // reset counter
98             }
99         }
100         // TC 3
101         if (sTC3.TC_sts) {  // timer 1 active
102             sTC3.TC_cnt += ticks;
103             if ( sTC3.TC_cnt >= sTC3.TC_reg ) { // counter reaches register
104                 (*sTC3.pfpISR_TC)();            // call ISR
105                 sTC3.TC_cnt = 0;                // reset counter
106             }
107         }
108 
109         start = clock() / MY_CORR_CLOCKS_PER_MS;    // system clock (adapt to used system)
110     } // end while
111 
112     /* shutdown thread */
113     MSGF('I',"... ending thread ...","tc_Thread");
114     MSGF('I',"... thread terminated","tc_Thread");
115     pthread_cancel(pthread_self()); // cancel this thread
116     return NULL;
117 }
```

References _SUS_, MSGF, MY_CORR_CLOCKS_PER_MS, sTCconf_t::pfpISR_TC, sTC1, sTC2, sTC3, sTCint, sTCconf_t::TC_cnt, sTCint_t::TC_init, sTCconf_t::TC_reg, and sTCconf_t::TC_sts.

Referenced by TCengine_start().

Here is the caller graph for this function:



4.8.2.8 TCengine_start()

```
void TCengine_start ( )
```

start TC engine Start timer counter engine thread

start TC engine

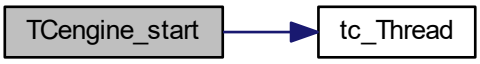
Definition at line 122 of file TCsim.c.

```
122                         {
123     if ( sTCint.TC_init ) {
124         MSGF('W',"TCengine already initialized","TCengine_start");
125         return;                                                    // stop here
126     }
127
128     // +++ start thread +++
129     int err;
130     err = pthread_create(&sTCint.threadID_TC, NULL, tc_Thread, NULL);
131     if (err != 0) {
132         MSGF('E',"Thread 'tc_Thread' NOT started","TCengine_start");
133         return;                                                    // stop here
134     } else {
135 #ifdef VERBOSE
136         MSGF('I',"Thread 'tc_Thread' started","TCengine_start");
137 #endif // VERBOSE
138     }
139     sTCint.TC_init = 1;     // set initialized
140 }
```

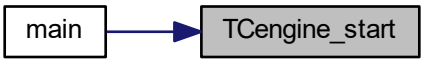
References MSGF, sTCint, sTCint_t::TC_init, tc_Thread(), and sTCint_t::threadID_TC.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.2.9 TCengine_stop()

```
void TCengine_stop ( )
```

stop TC engine Stop timer counter engine thread. Stops all timer counters.

stop TC engine

Definition at line 145 of file TCsim.c.

```
145                     {
146     if (!sTCint.TC_init) {
147         MSGF('W',"Thread 'tc_Thread' not running (terminated)","TCengine_stop");
148     }
149     else {
150 #ifdef VERBOSE
151         MSGF('I',"Terminating thread 'tc_Thread' ...","TCengine_stop");
152 #endif // VERBOSE
153     sTCint.TC_init = 0;
154     }
155 }
```

References MSGF, sTCint, and sTCint_t::TC_init.

4.8.3 Variable Documentation

4.8.3.1 sTC1

```
sTCconf_t sTC1 = {0}  [static]
```

TC1 ctrl struct instance.

Definition at line 48 of file TCsim.c.

Referenced by TC1config(), TC1stop(), and tc_Thread().

4.8.3.2 sTC2

```
sTCconf_t sTC2 = {0}  [static]
```

TC2 ctrl struct instance.

Definition at line 49 of file TCsim.c.

Referenced by TC2config(), TC2stop(), and tc_Thread().

4.8.3.3 sTC3

```
sTCconf_t sTC3 = {0}  [static]
```

TC3 ctrl struct instance.

Definition at line 50 of file TCsim.c.

Referenced by TC3config(), TC3stop(), and tc_Thread().

4.8.3.4 sTCint

```
sTCint_t sTCint = {0}  [static]
```

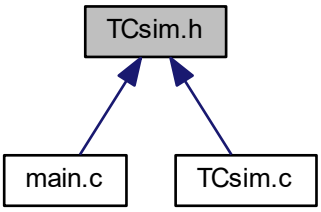
TC thread ctrl struct instance.

Definition at line 46 of file TCsim.c.

Referenced by tc_Thread(), TCengine_start(), and TCengine_stop().

4.9 TCsim.h File Reference

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



Macros

- #define TC_MAX_COUNT (65000)

  TC maximum count in capture mode (max 2^ 16 - 3)

Functions

- void TCengine_start ()
  start TC engine
- void TCengine_stop ()
  stop TC engine
- void TC1config (FpISR_t fpISR, uint16_t tc)
  config TC1
- void TC2config (FpISR_t fpISR, uint16_t tc)
  config TC2
- void TC3config (FpISR_t fpISR, uint16_t tc)
  config TC3
- void TC1stop ()
  Stop TC1.
- void TC2stop ()
  Stop TC2.
- void TC3stop ()
  Stop TC3.

4.9.1 Macro Definition Documentation

4.9.1.1 TC_MAX_COUNT

```
#define TC_MAX_COUNT (65000)
```

TC maximum count in capture mode (max 2^16 - 3)

Definition at line 4 of file TCsim.h.

4.9.2 Function Documentation

4.9.2.1 TC1config()

```
void TC1config (
            FpISR_t fpISR,
            uint16_t tc )
```

config TC1

config TC1

Parameters

fpISR	pointer to ISR
tc	timer register 0 < tc <= TC_MAX_COUNT

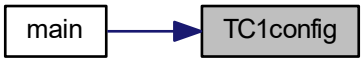
Definition at line 162 of file TCsim.c.

```
162                                             {
163     if ( fpISR != 0 ) {          // check pointer
164         sTC1.pfpISR_TC = fpISR;
165     }
166     else {
167         MSGF('E',"Null pointer to ISR","TC1config");
168         return;                  // leave function
169     }
170
171     if ( (tc == 0) || (tc > TC_MAX_COUNT) ) {          // check pointer
172         MSGF('E',"tc register not within limits","TC1config");
173         return;                  // leave function
174     }
175     else {
176        sTC1.TC_reg = tc;
177     }
178
179     sTC1.TC_sts = 1;             // TC successfully initialized
180 #ifdef VERBOSE
181         MSGF('I',"TC1 successfully initilaized","TC1config");
182 #endif // VERBOSE
183 }
```

References MSGF, sTCconf_t::pfpISR_TC, sTC1, TC_MAX_COUNT, sTCconf_t::TC_reg, and sTCconf_t::TC_sts.

Referenced by main().

Here is the caller graph for this function:



4.9.2.2 TC1stop()

```
void TC1stop ( )
```

Stop TC1.

Stop TC1.

Definition at line 245 of file TCsim.c.

```
245                 {
246     sTC1.TC_sts = 0;
247 }
```

References sTC1, and sTCconf_t::TC_sts.

4.9.2.3 TC2config()

```
void TC2config (
             FpISR_t fpISR,
             uint16_t tc )
```

config TC2

config TC2

Parameters

fpISR	pointer to ISR
tc	timer register 0 < tc <= TC_MAX_COUNT

Definition at line 190 of file TCsim.c.

```
190                                                 {
191     if ( fpISR != 0 ) {         // check pointer
192         sTC2.pfpISR_TC = fpISR;
193     }
194     else {
195         MSGF('E',"Null pointer to ISR","TC2config");
196         return;                 // leave function
197     }
198
199     if ( (tc == 0) || (tc > TC_MAX_COUNT) ) {         // check pointer
200         MSGF('E',"tc register not within limits","TC2config");
201         return;                 // leave function
202     }
203     else {
204        sTC2.TC_reg = tc;
205     }
206
207     sTC2.TC_sts = 1;            // TC successfully initialized
208 #ifdef VERBOSE
209         MSGF('I',"TC2 successfully initilaized","TC2config");
210 #endif // VERBOSE
211 }
```

References MSGF, sTCconf_t::pfpISR_TC, sTC2, TC_MAX_COUNT, sTCconf_t::TC_reg, and sTCconf_t::TC_sts.

4.9.2.4 TC2stop()

```
void TC2stop ( )
```

Stop TC2.

Stop TC2.

Definition at line 253 of file TCsim.c.

```
253                 {
254     sTC2.TC_sts = 0;
255 }
```

References sTC2, and sTCconf_t::TC_sts.

4.9.2.5 TC3config()

```
void TC3config (
            FpISR_t fpISR,
            uint16_t tc )
```

config TC3

config TC3

Parameters

fpISR	pointer to ISR
tc	timer register 0 < tc <= TC_MAX_COUNT

Definition at line 218 of file TCsim.c.

```
218                                             {
219     if ( fpISR != 0 ) {             // check pointer
220         sTC3.pfpISR_TC = fpISR;
221     }
222     else {
223         MSGF('E',"Null pointer to ISR","TC3config");
224         return;                     // leave function
225     }
226
227     if ( (tc == 0) || (tc > TC_MAX_COUNT) ) {         // check pointer
228         MSGF('E',"tc register not within limits","TC3config");
229         return;                     // leave function
230     }
231     else {
232        sTC3.TC_reg = tc;
233     }
234
235     sTC3.TC_sts = 1;                // TC successfully initialized
236 #ifdef VERBOSE
237         MSGF('I',"TC3 successfully initilaized","TC3config");
238 #endif // VERBOSE
239 }
```

References MSGF, sTCconf_t::pfpISR_TC, sTC3, TC_MAX_COUNT, sTCconf_t::TC_reg, and sTCconf_t::TC_sts.

4.9.2.6 TC3stop()

```
void TC3stop ( )
```

Stop TC3.

Stop TC3.

Definition at line 261 of file TCsim.c.

```
261                 {
262     sTC3.TC_sts = 0;
263 }
```

References sTC3, and sTCconf_t::TC_sts.

4.9.2.7 TCengine_start()

```
void TCengine_start ( )
```

start TC engine

start TC engine

Definition at line 122 of file TCsim.c.

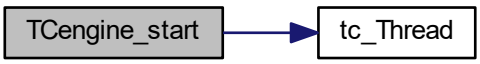
```
122                         {
123     if ( sTCint.TC_init ) {
124         MSGF('W',"TCengine already initialized","TCengine_start");
125         return;                                                    // stop here
126     }
```

```
127
128     // +++ start thread +++
129     int err;
130     err = pthread_create(&sTCint.threadID_TC, NULL, tc_Thread, NULL);
131     if (err != 0) {
132         MSGF('E',"Thread 'tc_Thread' NOT started","TCengine_start");
133         return;                                                    // stop here
134     } else {
135 #ifdef VERBOSE
136         MSGF('I',"Thread 'tc_Thread' started","TCengine_start");
137 #endif // VERBOSE
138     }
139     sTCint.TC_init = 1;     // set initialized
140 }
```

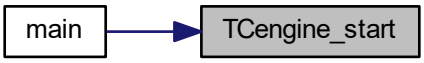
References MSGF, sTCint, sTCint_t::TC_init, tc_Thread(), and sTCint_t::threadID_TC.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.8 TCengine_stop()

```
void TCengine_stop ( )
```

stop TC engine

stop TC engine

Definition at line 145 of file TCsim.c.

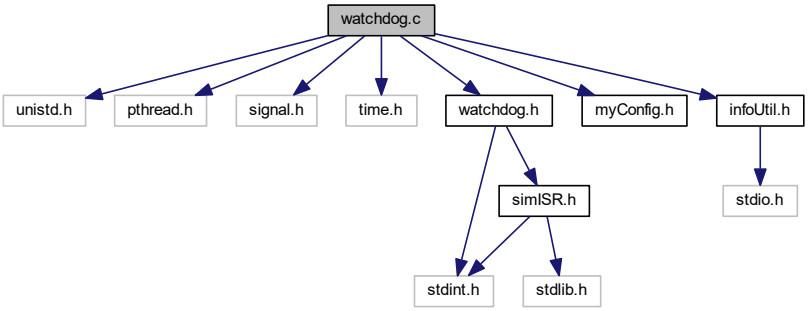
```
145                     {
146     if (!sTCint.TC_init) {
147         MSGF('W',"Thread 'tc_Thread' not running (terminated)","TCengine_stop");
148     }
149     else {
150 #ifdef VERBOSE
151         MSGF('I',"Terminating thread 'tc_Thread' ...","TCengine_stop");
152 #endif // VERBOSE
153     sTCint.TC_init = 0;
154     }
155 }
```

References MSGF, sTCint, and sTCint_t::TC_init.

4.10 watchdog.c File Reference

```
#include <unistd.h>
#include <pthread.h>
#include <signal.h>
#include <time.h>
#include "watchdog.h"
#include "myConfig.h"
#include "infoUtil.h"
```

Include dependency graph for watchdog.c:



Data Structures

- struct sWDint_t

  struct for watchdog internal configuration Keyboard thread configuration

Macros

- #define _FILE_ "watchdog.c"

  file name for macros in infoUtil.h
- #define _SUS_ (10000)

  sleep micro(u) seconds in thread. Comment if not to be used

Functions

- ISR_t watchdogDefault_ISR ()
- void * wd_Thread (void *vargp)

  Watchdog thread with POSIX Watch on timeout.
- void WATCHDOG_Initialize (uint16_t timeout)

  Initialization of watchdog timer Uses a service routine on timeout. May be the internal default or a user written service.
- void WATCHDOG_Reset ()

  Reset watchdog timer Refresh/reset the running watchdog timer.

Variables

- static sWDint_t sWDint = {0}

  watchdog ctrl struct instance

4.10.1 Macro Definition Documentation

4.10.1.1 _FILE_

```
#define _FILE_ "watchdog.c"
```

file name for macros in infoUtil.h

Definition at line 10 of file watchdog.c.

4.10.1.2 _SUS_

```
#define _SUS_ (10000)
```

sleep micro(u) seconds in thread. Comment if not to be used

Definition at line 12 of file watchdog.c.

4.10.2 Function Documentation

4.10.2.1 WATCHDOG_Initialize()

```
void WATCHDOG_Initialize (
            uint16_t timeout )
```

Initialization of watchdog timer Uses a service routine on timeout. May be the internal default or a user written service.

Initialization of watchdog timer.

Once started, watchdog cannot be stopped.

See also

WATCHDOG_Reset()

Parameters

timeout	in ms

Author

R.S. Mayer

Date

2022-01-09

Version

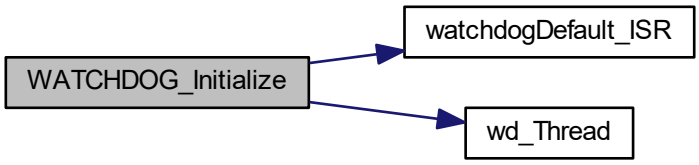
Definition at line 100 of file watchdog.c.

```
100                                             {
101     if ( sWDint.WD_init ) {
102         MSGF('W',"WATCHDOG already initialized","WATCHDOG_Initialize");
103         return;                                                     // stop here
104     }
105     // +++ initialize memory +++
106 //    sWDint.pfpISR_WD = (FpISR_t*) (HW_BASEADDR + ADDR_ISR_WD);    // set ISR vector address
107 //    if ( fpISR == 0 ) {                                           // use default ISR
108         sWDint.pfpISR_WD = watchdogDefault_ISR; // register default ISR callback
109 //    }
110 //    else {                                                        // user defined ISR (!= 0)
111 //        *(sWDint.pfpISR_WD) = fpISR;                              // register user defined ISR callback
112 //    }
113     // +++ set values +++
114     sWDint.WD_reg = timeout;    // set timeout
115     sWDint.WD_cnt = 0;          // reset count
116     // +++ start thread now +++
117     int err;
118     err = pthread_create(&sWDint.threadID_WD, NULL, wd_Thread, NULL);
119     if (err != 0) {
120         MSGF('E',"Thread 'wd_Thread' NOT started","WATCHDOG_Initialize");
121         return;                                                     // stop here
122     } else {
123         MSGF('I',"Thread 'wd_Thread' started","WATCHDOG_Initialize");
124     }
125     sWDint.WD_init = 1;     // set initialized
126     return;
127 }
```

References MSGF, sWDint_t::pfpISR_WD, sWDint, sWDint_t::threadID_WD, watchdogDefault_ISR(), sWDint_t::↩ WD_cnt, sWDint_t::WD_init, sWDint_t::WD_reg, and wd_Thread().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

4.10.2.2 WATCHDOG_Reset()

```
void WATCHDOG_Reset ( )
```

Reset watchdog timer Refresh/reset the running watchdog timer.

Reset watchdog timer.

Author

R.S. Mayer

Date

2021-05-14

Version

Definition at line 136 of file watchdog.c.

```
136                         {
137     if (!sWDint.WD_init) {
138         MSGF('W',"Watchdog not initialized, nothing to do","WATCHDOG_Reset");
139         return;
140     }
141     sWDint.WD_cnt = 0;      // reset count
142 }
```

References MSGF, sWDint, sWDint_t::WD_cnt, and sWDint_t::WD_init.

Referenced by main().

Here is the caller graph for this function:

4.10.2.3 watchdogDefault_ISR()

```
ISR_t watchdogDefault_ISR ( )
```

Default watchdog ISR. Action is exit(0).

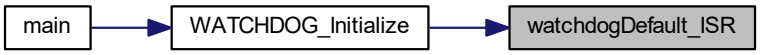
Definition at line 36 of file watchdog.c.

```
36                              {
37     printf( "\n" );
38     MSGF('I',"WATCHDOG fired!","watchdogDefault_ISR");
39     usleep( 500000 );           // wait 1/2 sec
40     exit(0);                    // terminates program
41 }
```

References MSGF.

Referenced by WATCHDOG_Initialize().

Here is the caller graph for this function:



4.10.2.4 wd_Thread()

```
void* wd_Thread (
             void * vargp )
```

Watchdog thread with POSIX Watch on timeout.

See also

fpISR_keyb

Waiting with *SUS* e.g. 10000 us = 1 ms improves performance of system output, e.q. printf(...) significantly!

See also

SUS

Author

R. S. Mayer

Date

2021-05-14

Version

2021-05-

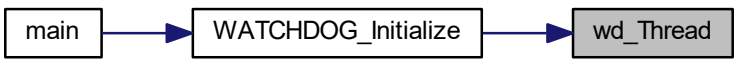
Definition at line 56 of file watchdog.c.

```
57 {
58     MSGF('I',"Thread 'WD_Thread' starts ...","wd_Thread");
59 
60     clock_t start = clock() / MY_CORR_CLOCKS_PER_MS; // system clock (adapt to used system)
61     clock_t ticks;
62     while (1) {                          // forever ...
63         // wait for next tick(s)
64         while ( (ticks = (clock() / MY_CORR_CLOCKS_PER_MS - start)) == 0 ) {
65 #ifdef _SUS_
66             usleep(_SUS_);                  // some wait
67 #endif // _SUS_
68                                             // do nothing but wait
69         }  // time elapsed
70         sWDint.WD_cnt += ticks;             // just increment meanwhile elapsed time
71 
72         if ( sWDint.WD_cnt >= sWDint.WD_reg ) {     // timeout reached
73             if ( sWDint.pfpISR_WD ) {       // ISR function pointer set
74                 (*sWDint.pfpISR_WD)();      // callback ISR
75                 break;                      // break loop, terminate thread
76             }
77         } // if timeout
78         start = clock() / MY_CORR_CLOCKS_PER_MS;    // system clock (adapt to used system)
79     } // end while
80     MSGF('I',"... ending thread ...","wd_Thread");
81 
82     MSGF('I',"... thread terminated","wd_Thread");
83     pthread_cancel(pthread_self()); // cancel this thread
84     return NULL;
85 }
```

References _SUS_, MSGF, MY_CORR_CLOCKS_PER_MS, sWDint_t::pfpISR_WD, sWDint, sWDint_t::WD_cnt, and sWDint_t::WD_reg.

Referenced by WATCHDOG_Initialize().

Here is the caller graph for this function:



4.10.3 Variable Documentation

4.10.3.1 sWDint

```
sWDint_t sWDint = {0}  [static]
```

watchdog ctrl struct instance

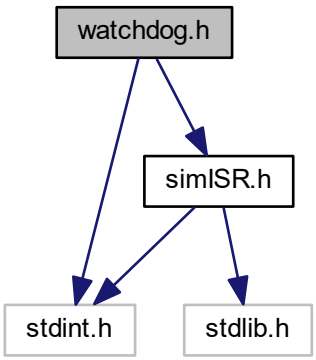
Definition at line 31 of file watchdog.c.

Referenced by WATCHDOG_Initialize(), WATCHDOG_Reset(), and wd_Thread().

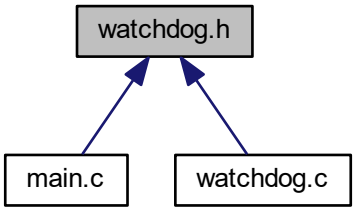
4.11 watchdog.h File Reference

```
#include <stdint.h>
#include "simISR.h"
```

Include dependency graph for watchdog.h:



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



Macros

- #define WDT_ACTION_DEFAULT (0)
  use default watchdog action on timeout
- #define WDT_TIMEOUT_500 (500)
  watchdog timeout 500 ms
- #define WDT_TIMEOUT_1000 (1000)
  watchdog timeout 1 s
- #define WDT_TIMEOUT_2000 (2000)
  watchdog timeout 2 s
- #define WDT_TIMEOUT_4000 (4000)
  watchdog timeout 4 s
- #define WDT_TIMEOUT_8000 (8000)
  watchdog timeout 8 s

Functions

- void WATCHDOG_Initialize (uint16_t timeout)

  Initialization of watchdog timer.
- void WATCHDOG_Reset ()

  Reset watchdog timer.

4.11.1 Macro Definition Documentation

4.11.1.1 WDT_ACTION_DEFAULT

```
#define WDT_ACTION_DEFAULT (0)
```

use default watchdog action on timeout

Definition at line 7 of file watchdog.h.

4.11.1.2 WDT_TIMEOUT_1000

```
#define WDT_TIMEOUT_1000 (1000)
```

watchdog timeout 1 s

Definition at line 10 of file watchdog.h.

4.11.1.3 WDT_TIMEOUT_2000

```
#define WDT_TIMEOUT_2000 (2000)
```

watchdog timeout 2 s

Definition at line 11 of file watchdog.h.

4.11.1.4 WDT_TIMEOUT_4000

```
#define WDT_TIMEOUT_4000 (4000)
```

watchdog timeout 4 s

Definition at line 12 of file watchdog.h.

4.11.1.5 WDT_TIMEOUT_500

```
#define WDT_TIMEOUT_500 (500)
```

watchdog timeout 500 ms

Definition at line 9 of file watchdog.h.

4.11.1.6 WDT_TIMEOUT_8000

```
#define WDT_TIMEOUT_8000 (8000)
```

watchdog timeout 8 s

Definition at line 13 of file watchdog.h.

4.11.2 Function Documentation

4.11.2.1 WATCHDOG_Initialize()

```
void WATCHDOG_Initialize (
             uint16_t timeout )
```

Initialization of watchdog timer.

Initialization of watchdog timer.

Once started, watchdog cannot be stopped.

See also

WATCHDOG_Reset()

Parameters

timeout	in ms

Author

R.S. Mayer

Date

2022-01-09

Version

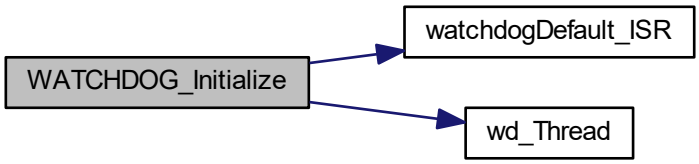
Definition at line 100 of file watchdog.c.

```
100                                             {
101     if ( sWDint.WD_init ) {
102         MSGF('W',"WATCHDOG already initialized","WATCHDOG_Initialize");
103         return;                                                     // stop here
104     }
105     // +++ initialize memory +++
106 //    sWDint.pfpISR_WD = (FpISR_t*) (HW_BASEADDR + ADDR_ISR_WD);    // set ISR vector address
107 //    if ( fpISR == 0 ) {                                           // use default ISR
108         sWDint.pfpISR_WD = watchdogDefault_ISR; // register default ISR callback
109 //    }
110 //    else {                                                        // user defined ISR (!= 0)
111 //        *(sWDint.pfpISR_WD) = fpISR;                              // register user defined ISR callback
112 //    }
113     // +++ set values +++
114     sWDint.WD_reg = timeout;    // set timeout
115     sWDint.WD_cnt = 0;          // reset count
116     // +++ start thread now +++
117     int err;
118     err = pthread_create(&sWDint.threadID_WD, NULL, wd_Thread, NULL);
119     if (err != 0) {
120         MSGF('E',"Thread 'wd_Thread' NOT started","WATCHDOG_Initialize");
121         return;                                                     // stop here
122     } else {
123         MSGF('I',"Thread 'wd_Thread' started","WATCHDOG_Initialize");
124     }
125     sWDint.WD_init = 1;     // set initialized
126     return;
127 }
```

References MSGF, sWDint_t::pfpISR_WD, sWDint, sWDint_t::threadID_WD, watchdogDefault_ISR(), sWDint_t::↩
WD_cnt, sWDint_t::WD_init, sWDint_t::WD_reg, and wd_Thread().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

4.11.2.2 WATCHDOG_Reset()

```
void WATCHDOG_Reset ( )
```

Reset watchdog timer.

Reset watchdog timer.

Author

R.S. Mayer

Date

2021-05-14

Version

Definition at line 136 of file watchdog.c.

```
136                         {
137     if (!sWDint.WD_init) {
138         MSGF('W',"Watchdog not initialized, nothing to do","WATCHDOG_Reset");
139         return;
140     }
141     sWDint.WD_cnt = 0;      // reset count
142 }
```

References MSGF, sWDint, sWDint_t::WD_cnt, and sWDint_t::WD_init.

Referenced by main().

Here is the caller graph for this function:

Index