HARMcksL: ARM HAL toolbox (yet STM32 oriented)

1.3

Generated by Doxygen 1.8.13

Contents

1 Weak Functions List

Global SERIAL_DBG_Message_Handler (const char *msg, const uint8_t len)

This function is implemented as weak to be implemented in projects (weak one only prints & flushes the buffer)

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Data Structure Documentation

4.1 DateTime Struct Reference

Basic Date & Time struct.

`#include <time_utils.h>`

Data Fields

- uint16_t Year
  Year.
- uint8_t Month
  Month.
- uint8_t Day
  Day.

- uint8_t Weekday
  Weekday.
- uint8_t Hours
  Hours.
- uint8_t Minutes
  Minutes.
- uint8_t Seconds
  Seconds.

4.1.1 Detailed Description

Basic Date & Time struct.

4.1.2 Field Documentation

4.1.2.1 Day

`uint8_t DateTime::Day`

Day.

4.1.2.2 Hours

`uint8_t DateTime::Hours`

Hours.

4.1.2.3 Minutes

`uint8_t DateTime::Minutes`

Minutes.

4.1.2.4 Month

`uint8_t DateTime::Month`

Month.

4.1.2.5 Seconds

`uint8_t DateTime::Seconds`

Seconds.

4.1.2.6 Weekday

`uint8_t DateTime::Weekday`

Weekday.

4.1.2.7 Year

`uint16_t DateTime::Year`

Year.

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

- time_utils.h

4.2 GPIO_in Struct Reference

GPIO input structure.

`#include <GPIO_ex.h>`

Data Fields

- bool in
  Input value.
- eEdge edge
  Input edge.
- bool mem
  Memo value.
- uint32_t hIn
  Filter time.
- struct {
  GPIO_TypeDef * GPIOx
  HAL GPIO instance.
  uint16_t GPIO_Pin
  HAL GPIO pin.
  uint16_t filt
  Filter time (ms)
  bool logic
  Input logic polarity.
  bool repeat
  Callback ON repeat.
  void(* onSet)(void)
  Push callback ON function pointer.
  void(* onReset)(void)
  Push callback OFF function pointer.
  } cfg

4.2.1 Detailed Description

GPIO input structure.

4.2.2 Field Documentation

4.2.2.1 cfg

```
struct { ... } GPIO_in::cfg
```

4.2.2.2 edge

```
eEdge GPIO_in::edge
```

Input edge.

4.2.2.3 filt

```
uint16_t GPIO_in::filt
```

Filter time (ms)

4.2.2.4 GPIO_Pin

```
uint16_t GPIO_in::GPIO_Pin
```

HAL GPIO pin.

4.2.2.5 GPIOx

```
GPIO_TypeDef* GPIO_in::GPIOx
```

HAL GPIO instance.

4.2.2.6 hIn

```
uint32_t GPIO_in::hIn
```

Filter time.

4.2.2.7 in

```
bool GPIO_in::in
```

Input value.

4.2.2.8 logic

```
bool GPIO_in::logic
```

Input logic polarity.

4.2.2.9 mem

```
bool GPIO_in::mem
```

Memo value.

4.2.2.10 onReset

```
void(* GPIO_in::onReset) (void)
```

Push callback OFF function pointer.

4.2.2.11 onSet

```
void(* GPIO_in::onSet) (void)
```

Push callback ON function pointer.

4.2.2.12 repeat

```
bool GPIO_in::repeat
```

Callback ON repeat.

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

- GPIO_ex.h

4.3 logicPWM Struct Reference

Software PWM on GPIO struct.

`#include <PWM.h>`

Data Fields

- uint16_t cntr
  Counter.
- uint16_t duty
  Current Duty cycle.
- struct {
  TIM_HandleTypeDef * pTim
  Timer instance (for reference)
  GPIO_TypeDef * GPIOx
  Port of emulated PWM pin.
  uint16_t GPIO_Pin
  Pin mask on port.
  uint16_t tim_freq
  Timer frequency (for reference)
  uint16_t duty
  Duty Cycle (effective when new period starts)
  uint16_t per
  Overflow threshold (emulated PWM period)
  bool polarity
  Output polarity.
  } cfg

4.3.1 Detailed Description

Software PWM on GPIO struct.

4.3.2 Field Documentation

4.3.2.1 cfg

`struct { ... } logicPWM::cfg`

4.3.2.2 cntr

`uint16_t logicPWM::cntr`

Counter.

4.3.2.3 duty

```
uint16_t logicPWM::duty
```

Current Duty cycle.

Duty Cycle (effective when new period starts)

4.3.2.4 GPIO_Pin

```
uint16_t logicPWM::GPIO_Pin
```

Pin mask on port.

4.3.2.5 GPIOx

```
GPIO_TypeDef* logicPWM::GPIOx
```

Port of emulated PWM pin.

4.3.2.6 per

```
uint16_t logicPWM::per
```

Overflow threshold (emulated PWM period)

4.3.2.7 polarity

```
bool logicPWM::polarity
```

Output polarity.

4.3.2.8 pTim

```
TIM_HandleTypeDef* logicPWM::pTim
```

Timer instance (for reference)

4.3.2.9 tim_freq

```
uint16_t logicPWM::tim_freq
```

Timer frequency (for reference)

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

- PWM.h

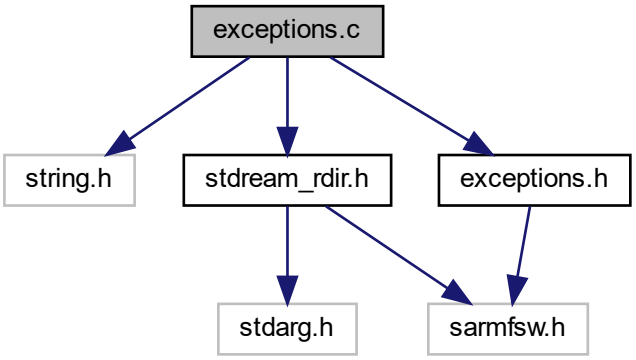
5 File Documentation

5.1 exceptions.c File Reference

Debug tool helpers.

```
#include <string.h>
#include "stdream_rdir.h"
#include "exceptions.h"
```

Include dependency graph for exceptions.c:



Functions

- void HardFault_Handler_callback (const uint32_t stack[ ])
  prints informations about current Hard Fault exception
- void Error_Handler_callback (const uint32_t stack[ ])
  prints informations about current Hard Fault exception

5.1.1 Detailed Description

Debug tool helpers.

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

5.1.2 Function Documentation

5.1.2.1 Error_Handler_callback()

```
void Error_Handler_callback (
             const uint32_t stack[] )
```

prints informations about current Hard Fault exception

Parameters

in	*stack*	- pointer to stack address

Note

HardFault_Handler_callback should not be called directly use exception_Handler() which prepares pointer to current stack instead

Warning

Depending how arm is fucked up, informations may not be printed, at least, you could inspect exception and stack through debug breakpoint

Returns

Never (anyways, arm fubared!)

5.1.2.2 HardFault_Handler_callback()

```
void HardFault_Handler_callback (
            const uint32_t stack[] )
```

prints informations about current Hard Fault exception

Parameters

in	*stack*	- pointer to stack address

Note

HardFault_Handler_callback should not be called directly use exception_Handler() which prepares pointer to current stack instead

Warning

Depending how arm is fucked up, informations may not be printed, at least, you could inspect exception and stack through debug breakpoint

Returns

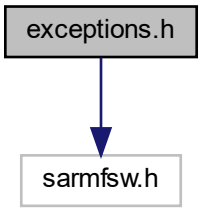
Never (anyways, arm fubared!)

5.2 exceptions.h File Reference

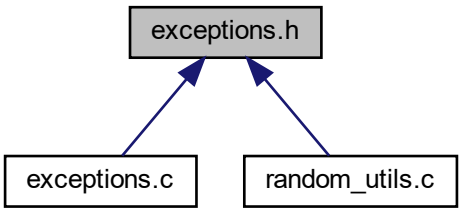
Debug tool and helpers.

```
#include "sarmfsw.h"
```

Include dependency graph for exceptions.h:



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



Macros

- #define exception_Handler(e)
  Exception handler asm caller.

Functions

- void HardFault_Handler_callback (const uint32_t stack[ ])
  prints informations about current Hard Fault exception
- void Error_Handler_callback (const uint32_t stack[ ])
  prints informations about current Hard Fault exception

5.2.1 Detailed Description

Debug tool and helpers.

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

5.2.2 Macro Definition Documentation

5.2.2.1 exception_Handler

```
#define exception_Handler(
              e )
```

Value:

```
__ASM ( "tst lr, #4 \r\n"                    \
            "ite EQ \r\n"                        \
            "mrseq r0, MSP \r\n"                 \
            "mrsne r0, PSP \r\n"                 \
            "b " #e "_Handler_callback \r\n");
```

Exception handler asm caller.

Note

The exception_Handler should be called with corresponding exception name **e** as parameter

5.2.3 Function Documentation

5.2.3.1 Error_Handler_callback()

```
void Error_Handler_callback (
             const uint32_t stack[] )
```

prints informations about current Hard Fault exception

Parameters

in	*stack*	- pointer to stack address

Note

HardFault_Handler_callback should not be called directly use exception_Handler() which prepares pointer to current stack instead

Warning

Depending how arm is fucked up, informations may not be printed, at least, you could inspect exception and stack through debug breakpoint

Returns

Never (anyways, arm fubared!)

5.2.3.2 HardFault_Handler_callback()

```
void HardFault_Handler_callback (
            const uint32_t stack[] )
```

prints informations about current Hard Fault exception

Parameters

in	*stack*	- pointer to stack address

Note

HardFault_Handler_callback should not be called directly use exception_Handler() which prepares pointer to current stack instead

Warning

Depending how arm is fucked up, informations may not be printed, at least, you could inspect exception and stack through debug breakpoint

Returns

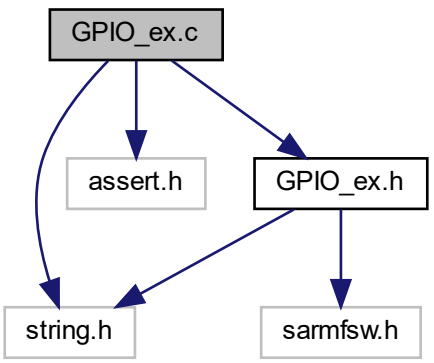
Never (anyways, arm fubared!)

5.3 GPIO_ex.c File Reference

Simple extension for GPIOs.

```
#include <string.h>
#include <assert.h>
#include "GPIO_ex.h"
```

Include dependency graph for GPIO_ex.c:



Macros

- #define MAX_PINS_PORT 16

Functions

- void GPIO_in_init (GPIO_in ∗in, GPIO_TypeDef ∗GPIOx, const uint16_t GPIO_Pin, const bool logic, const uint16_t filter, void(∗onSet)(void), void(∗onReset)(void), const bool repeat)

  Initialize GPIO_in instance.
- void GPIO_in_handler (GPIO_in ∗in)

  Handles GPIO_in read and treatment.
- FctERR str_GPIO_name (char ∗name, const GPIO_TypeDef ∗GPIOx, const uint16_t GPIO_Pin)

  Get name from Port, Pin.

5.3.1 Detailed Description

Simple extension for GPIOs.

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

5.3.2 Macro Definition Documentation

5.3.2.1 MAX_PINS_PORT

```
#define MAX_PINS_PORT 16
```

5.3.3 Function Documentation

5.3.3.1 GPIO_in_handler()

```
void GPIO_in_handler (
            GPIO_in * in )
```

Handles GPIO_in read and treatment.

Parameters

in,out	*in*	- input instance to handle

Returns

Nothing

5.3.3.2 GPIO_in_init()

```
void GPIO_in_init (
            GPIO_in * in,
            GPIO_TypeDef * GPIOx,
            const uint16_t GPIO_Pin,
            const bool logic,
            const uint16_t filter,
            void(*)(void) onSet,
            void(*)(void) onReset,
            const bool repeat )
```

Initialize GPIO_in instance.

Parameters

in,out	*in*	- input instance to initialize
in	*GPIOx*	- port to read from
in	*GPIO_Pin*	- pin to read from
in	*logic*	- set to 0 if pull-up (switching to GND), 1 if pull-down (switching to Vdd)
in	*filter*	- input filtering time
in	*onSet*	- Pointer to callback ON function
in	*onReset*	- Pointer to callback OFF function
in	*repeat*	- To repeat callback ON as long as input is set

Returns

Nothing

5.3.3.3 str_GPIO_name()

```
FctERR str_GPIO_name (
            char * name,
            const GPIO_TypeDef * GPIOx,
            const uint16_t GPIO_Pin )
```

Get name from Port, Pin.

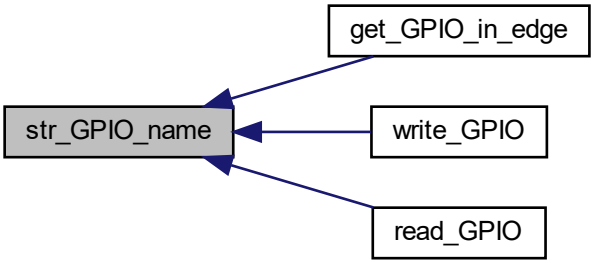
Parameters

in,out	*name*	- pointer to string for name
in	*GPIOx*	- port to write to
in	*GPIO_Pin*	- pin to write to

Returns

Error code

Here is the caller graph for this function:

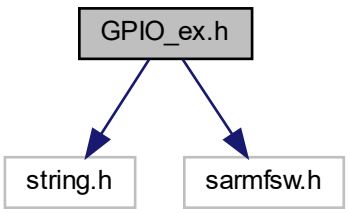


5.4 GPIO_ex.h File Reference

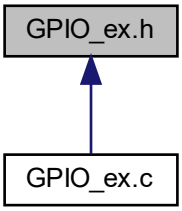
Simple extension for GPIOs.

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

Include dependency graph for GPIO_ex.h:



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



Data Structures

- struct GPIO_in
  GPIO input structure.

Typedefs

- typedef struct GPIO_in GPIO_in

Functions

- void GPIO_in_init (GPIO_in ∗in, GPIO_TypeDef ∗GPIOx, const uint16_t GPIO_Pin, const bool logic, const uint16_t filter, void(∗onSet)(void), void(∗onReset)(void), const bool repeat)
  Initialize GPIO_in instance.
- void GPIO_in_handler (GPIO_in ∗in)
  Handles GPIO_in read and treatment.
- bool get_GPIO_in (const GPIO_in ∗in)
  Get GPIO_in input value.
- bool get_GPIO_in_edge (const GPIO_in ∗in)
  Get GPIO_in input edge.
- FctERR str_GPIO_name (char ∗name, const GPIO_TypeDef ∗GPIOx, const uint16_t GPIO_Pin)
  Get name from Port, Pin.
- void write_GPIO (GPIO_TypeDef ∗GPIOx, const uint16_t GPIO_Pin, const eGPIOState Act)
  Write GPIO.
- GPIO_PinState read_GPIO (GPIO_TypeDef ∗GPIOx, const uint16_t GPIO_Pin)
  Read GPIO.

5.4.1 Detailed Description

Simple extension for GPIOs.

Author

SMFSW

Copyright

5.4.2 Typedef Documentation

5.4.2.1 GPIO_in

```
typedef struct GPIO_in GPIO_in
```

5.4.3 Function Documentation

5.4.3.1 get_GPIO_in()

```
bool get_GPIO_in (
            const GPIO_in * in )  [inline]
```

Get GPIO_in input value.

Parameters

in	*in*	- input instance

Returns

Input value

5.4.3.2 get_GPIO_in_edge()

```
bool get_GPIO_in_edge (
            const GPIO_in * in )  [inline]
```

Get GPIO_in input edge.

Parameters

in	*in*	- input instance

Returns

Input edge

Here is the call graph for this function:

5.4.3.3 GPIO_in_handler()

```
void GPIO_in_handler (
            GPIO_in * in )
```

Handles GPIO_in read and treatment.

Parameters

in,out	*in*	- input instance to handle

Returns

Nothing

5.4.3.4 GPIO_in_init()

```
void GPIO_in_init (
            GPIO_in * in,
            GPIO_TypeDef * GPIOx,
            const uint16_t GPIO_Pin,
            const bool logic,
            const uint16_t filter,
            void(*)(void) onSet,
            void(*)(void) onReset,
            const bool repeat )
```

Initialize GPIO_in instance.

Parameters

in,out	*in*	- input instance to initialize
in	*GPIOx*	- port to read from

Parameters

in	*GPIO_Pin*	- pin to read from
in	*logic*	- set to 0 if pull-up (switching to GND), 1 if pull-down (switching to Vdd)
in	*filter*	- input filtering time
in	*onSet*	- Pointer to callback ON function
in	*onReset*	- Pointer to callback OFF function
in	*repeat*	- To repeat callback ON as long as input is set

Returns

Nothing

5.4.3.5 read_GPIO()

```
GPIO_PinState read_GPIO (
            GPIO_TypeDef * GPIOx,
            const uint16_t GPIO_Pin )  [inline]
```

Read GPIO.

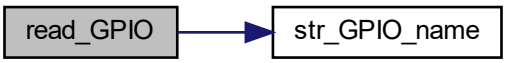
Parameters

in	*GPIOx*	- port to read from
in	*GPIO_Pin*	- pin to read from

Returns

Pin state

Here is the call graph for this function:



5.4.3.6 str_GPIO_name()

```
FctERR str_GPIO_name (
            char * name,
            const GPIO_TypeDef * GPIOx,
            const uint16_t GPIO_Pin )
```

Get name from Port, Pin.

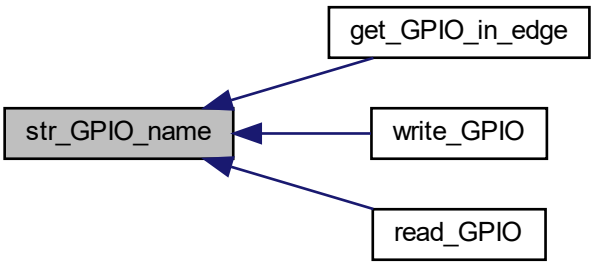
Parameters

in,out	*name*	- pointer to string for name
in	*GPIOx*	- port to write to
in	*GPIO_Pin*	- pin to write to

Returns

Error code

Here is the caller graph for this function:



5.4.3.7 write_GPIO()

```
void write_GPIO (
            GPIO_TypeDef * GPIOx,
            const uint16_t GPIO_Pin,
            const eGPIOState Act )  [inline]
```

Write GPIO.

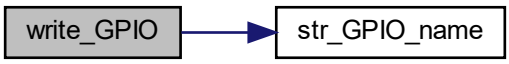
Parameters

in	*GPIOx*	- port to write to
in	*GPIO_Pin*	- pin to write to
in	*Act*	- type of write

Returns

Nothing

Here is the call graph for this function:

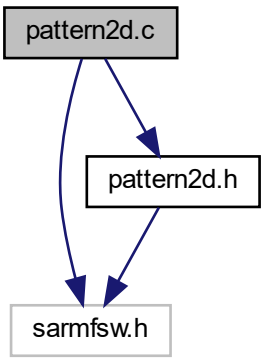


5.5 pattern2d.c File Reference

2 dimensional patterns utilities

```
#include "sarmfsw.h"
#include "pattern2d.h"
```

Include dependency graph for pattern2d.c:



Functions

- uint16_t pattern_evaluate (const uint16_t array[ ][2], const uint16_t nb, const uint16_t val)

  2 dimensional pattern evaluation algorithm

5.5.1 Detailed Description

2 dimensional patterns utilities

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

5.5.2 Function Documentation

5.5.2.1 pattern_evaluate()

```
uint16_t pattern_evaluate (
            const uint16_t array[][2],
            const uint16_t nb,
            const uint16_t val )
```

2 dimensional pattern evaluation algorithm

Parameters

in	*array*	- pointer to 2 dimensional
in	*nb*	- Number of items of the array
in	*val*	- Value to evaluate

Returns

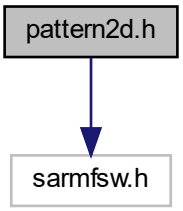
Evaluated value in regard of val

5.6 pattern2d.h File Reference

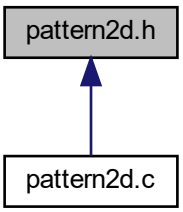
2 dimensional patterns utilities

```
#include "sarmfsw.h"
```

Include dependency graph for pattern2d.h:



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



Macros

- #define PATTERN_TAB(name, nbElem)

  *Pattern tab typedef declaration with **name** catenation and **nbElem** max tab elements.*
- #define PATTERN_EVALUATE(name, val) pattern_evaluate(name.array, name.nb, val)

  Macro to call linearization on a PATTERN_TAB typedef.

Functions

- uint16_t pattern_evaluate (const uint16_t array[ ][2], const uint16_t nb, const uint16_t val)

  2 dimensional pattern evaluation algorithm

5.6.1 Detailed Description

2 dimensional patterns utilities

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

5.6.2 Macro Definition Documentation

5.6.2.1 PATTERN_EVALUATE

```
#define PATTERN_EVALUATE(
              name,
              val ) pattern_evaluate(name.array, name.nb, val)
```

Macro to call linearization on a PATTERN_TAB typedef.

5.6.2.2 PATTERN_TAB

```
#define PATTERN_TAB(
              name,
              nbElem )
```

Value:

```
typedef struct pattern##name {          \
                                            const uint16_t  nb;                 \
                                            const uint16_t  array[nbElem][2];   \
                                        } pattern##name;
```

Pattern tab typedef declaration with **name** catenation and **nbElem** max tab elements.

5.6.3 Function Documentation

5.6.3.1 pattern_evaluate()

```
uint16_t pattern_evaluate (
             const uint16_t array[][2],
             const uint16_t nb,
             const uint16_t val )
```

2 dimensional pattern evaluation algorithm

Parameters

in	*array*	- pointer to 2 dimensional
in	*nb*	- Number of items of the array
in	*val*	- Value to evaluate

Returns

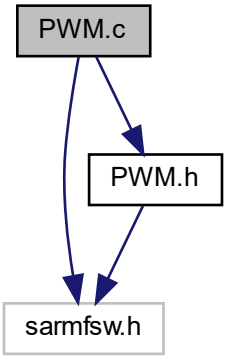
Evaluated value in regard of val

5.7 PWM.c File Reference

Straightforward PWM handling.

```
#include "sarmfsw.h"
#include "PWM.h"
```

Include dependency graph for PWM.c:



Functions

- HAL_StatusTypeDef init_TIM_Base (TIM_HandleTypeDef ∗pTim, const uint32_t freq)

  Init TIM module and start interruptions.
- HAL_StatusTypeDef set_TIM_Freq (TIM_HandleTypeDef ∗pTim, const uint32_t freq)

  Set TIM module frequency.
- HAL_StatusTypeDef init_PWM_Chan (TIM_HandleTypeDef ∗pTim, const uint32_t chan, const uint16_t freq)

  Init TIM PWM module channel with frequency and starts the channel.
- HAL_StatusTypeDef set_PWM_Duty_Scaled (const TIM_HandleTypeDef ∗pTim, const uint32_t chan, const uint16_t duty, const uint16_t scale)

  Set TIM module PWM duty cycle (scaled)
- FctERR logPWM_setPin (logicPWM ∗pPWM, GPIO_TypeDef ∗GPIOx, const uint16_t GPIO_Pin, const bool polarity)

  Set channel pin & polarity for emulated PWM channel.
- FctERR logPWM_setFreq (logicPWM ∗pPWM, TIM_HandleTypeDef ∗pTim, const uint16_t freq, uint16_↩ t granularity)

  Set channel frequency for emulated PWM channel.
- FctERR logPWM_setDuty (logicPWM ∗pPWM, const uint16_t val)

  Set new duty cycle for emulated PWM channel.
- FctERR logPWM_getFreq (uint16_t ∗freq, const logicPWM ∗pPWM)

  Get channel frequency for emulated PWM channel.
- FctERR logPWM_getDutyCycle (float ∗duty, const logicPWM ∗pPWM)

  Get channel Duty Cycle for emulated PWM channel.
- void logPWM_handler (logicPWM ∗pPWM)

  Handler for an emulated PWM channel.

5.7.1 Detailed Description

Straightforward PWM handling.

Author

SMFSW

Copyright

Warning

Shall work for all STM32 F families, L families not totally covered

5.7.2 Function Documentation

5.7.2.1 init_PWM_Chan()

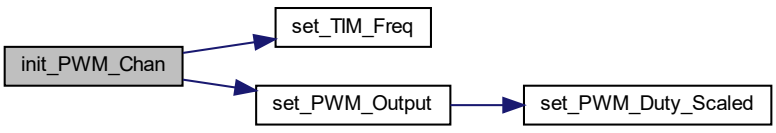
```
HAL_StatusTypeDef init_PWM_Chan (
            TIM_HandleTypeDef * pTim,
            const uint32_t chan,
            const uint16_t freq )
```

Init TIM PWM module channel with frequency and starts the channel.

Parameters

in,out	*pTim*	- pointer to TIM instance for PWM generation
in	*chan*	- Channel to write
in	*freq*	- Desired PWM frequency

Here is the call graph for this function:



Here is the caller graph for this function:

5.7.2.2 init_TIM_Base()

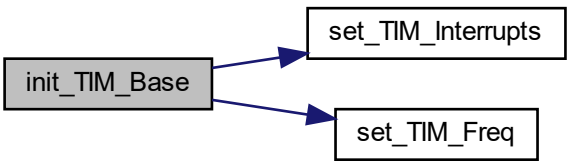
```
HAL_StatusTypeDef init_TIM_Base (
            TIM_HandleTypeDef * pTim,
            const uint32_t freq )
```

Init TIM module and start interruptions.

Parameters

in,out	*pTim*	- pointer to TIM instance
in	*freq*	- Desired TIM frequency

Here is the call graph for this function:



Here is the caller graph for this function:

5.7.2.3 logPWM_getDutyCycle()

```
FctERR logPWM_getDutyCycle (
            float * duty,
            const logicPWM * pPWM )
```

Get channel Duty Cycle for emulated PWM channel.

Parameters

in,out	*duty*	- pointer to duty cycle result
in,out	*pPWM*	- pointer to emulated PWM channel

Returns

Error code

Here is the caller graph for this function:

5.7.2.4 logPWM_getFreq()

```
FctERR logPWM_getFreq (
            uint16_t * freq,
            const logicPWM * pPWM )
```

Get channel frequency for emulated PWM channel.

Parameters

in,out	*freq*	- pointer to frequency result
in,out	*pPWM*	- pointer to emulated PWM channel

Returns

Error code

Here is the caller graph for this function:

5.7.2.5 logPWM_handler()

```
void logPWM_handler (
            logicPWM * pPWM )
```

Handler for an emulated PWM channel.

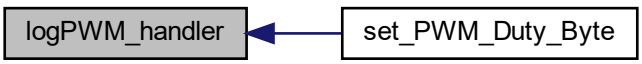
Warning

Shall be called directly from timer interrupt (HAL_TIM_PeriodElapsedCallback)

Parameters

in,out	*pPWM*	- pointer to emulated PWM channel

Here is the caller graph for this function:



5.7.2.6 logPWM_setDuty()

```
FctERR logPWM_setDuty (
            logicPWM * pPWM,
            const uint16_t val )
```

Set new duty cycle for emulated PWM channel.

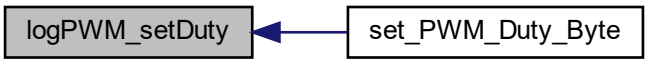
Parameters

in,out	*pPWM*	- pointer to emulated PWM channel
in	*val*	- Duty cycle to apply

Returns

Error code

Here is the caller graph for this function:



5.7.2.7 logPWM_setFreq()

```
FctERR logPWM_setFreq (
            logicPWM * pPWM,
            TIM_HandleTypeDef * pTim,
            const uint16_t freq,
            uint16_t granularity )
```

Set channel frequency for emulated PWM channel.

Warning

For multiple PWMs on same timer with different frequencies, take care of init order (first configured channel will get TIM parameters precedence)

Parameters

in,out	*pPWM*	- pointer to emulated PWM channel
in,out	*pTim*	- pointer to TIM instance for Frequency computation
in	*freq*	- PWM frequency to apply
in	*granularity*	- PWM duty cycle granularity

Returns

Error code

Here is the caller graph for this function:

5.7.2.8 logPWM_setPin()

```
FctERR logPWM_setPin (
            logicPWM * pPWM,
            GPIO_TypeDef * GPIOx,
            const uint16_t GPIO_Pin,
            const bool polarity )
```

Set channel pin & polarity for emulated PWM channel.

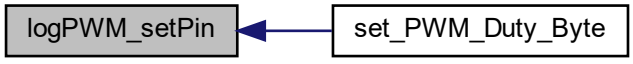
Parameters

in	*GPIOx*	- port for emulated PWM
in	*GPIO_Pin*	- pin for emulated PWM
in,out	*pPWM*	- pointer to emulated PWM channel
in	*polarity*	- 0: low polarity, 1: high polarity

Returns

Error code

Here is the caller graph for this function:



5.7.2.9 set_PWM_Duty_Scaled()

```
HAL_StatusTypeDef set_PWM_Duty_Scaled (
            const TIM_HandleTypeDef * pTim,
            const uint32_t chan,
            const uint16_t duty,
            const uint16_t scale )
```

Set TIM module PWM duty cycle (scaled)

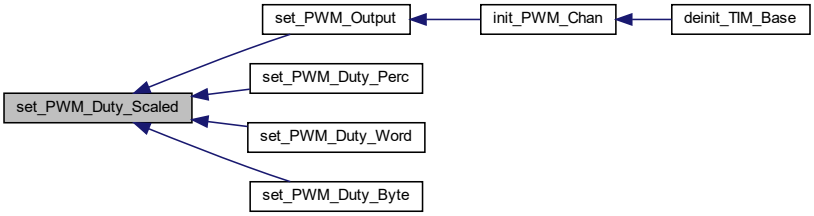
Parameters

in,out	*pTim*	- pointer to TIM instance for PWM generation
in	*chan*	- Channel to write
in	*duty*	- Scaled duty cycle value to write
in	*scale*	- Full scale value

Returns

HAL Status

Here is the caller graph for this function:



5.7.2.10 set_TIM_Freq()

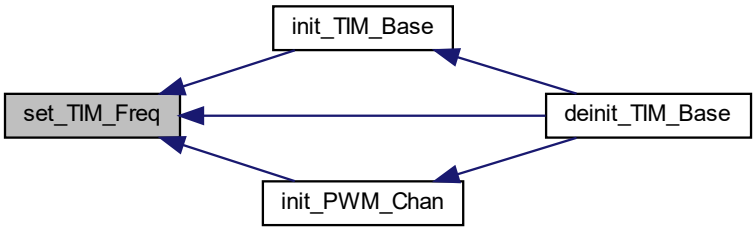
```
HAL_StatusTypeDef set_TIM_Freq (
            TIM_HandleTypeDef * pTim,
            const uint32_t freq )
```

Set TIM module frequency.

Parameters

in,out	*pTim*	- pointer to TIM instance for Frequency computation
in	*freq*	- Desired TIM frequency

Here is the caller graph for this function:

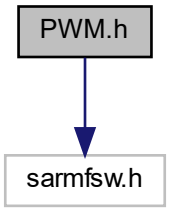


5.8 PWM.h File Reference

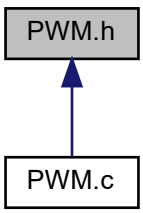
Straightforward PWM handling.

```
#include "sarmfsw.h"
```

Include dependency graph for PWM.h:



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



Data Structures

- struct logicPWM

  Software PWM on GPIO struct.

Typedefs

- typedef volatile struct logicPWM logicPWM

Functions

- HAL_StatusTypeDef set_TIM_Interrupts (TIM_HandleTypeDef *pTim, const bool on)

  Start TIM module interrupts.
- HAL_StatusTypeDef deinit_TIM_Base (TIM_HandleTypeDef *pTim)

  De-Init TIM module and start interruptions.
- HAL_StatusTypeDef init_TIM_Base (TIM_HandleTypeDef *pTim, const uint32_t freq)

  Init TIM module and start interruptions.
- HAL_StatusTypeDef set_TIM_Freq (TIM_HandleTypeDef *pTim, const uint32_t freq)

  Set TIM module frequency.
- HAL_StatusTypeDef init_PWM_Chan (TIM_HandleTypeDef *pTim, const uint32_t chan, const uint16_t freq)

  Init TIM PWM module channel with frequency and starts the channel.
- HAL_StatusTypeDef set_PWM_Output (TIM_HandleTypeDef *pTim, const uint32_t chan, const bool on)

  Set PWM channel output on/off.
- HAL_StatusTypeDef set_PWM_Duty_Scaled (const TIM_HandleTypeDef *pTim, const uint32_t chan, const uint16_t duty, const uint16_t scale)

  Set TIM module PWM duty cycle (scaled)
- HAL_StatusTypeDef set_PWM_Duty_Perc (const TIM_HandleTypeDef *pTim, const uint32_t chan, const uint16_t duty)

  Set TIM module PWM duty cycle (percents)
- HAL_StatusTypeDef set_PWM_Duty_Word (const TIM_HandleTypeDef *pTim, const uint32_t chan, const uint16_t duty)

  Set TIM module PWM duty cycle (u16-bit value)
- HAL_StatusTypeDef set_PWM_Duty_Byte (const TIM_HandleTypeDef *pTim, const uint32_t chan, const uint8_t duty)

  Set TIM module PWM duty cycle (u8-bit value)
- FctERR logPWM_setPin (logicPWM *pPWM, GPIO_TypeDef *GPIOx, const uint16_t GPIO_Pin, const bool polarity)

  Set channel pin & polarity for emulated PWM channel.
- FctERR logPWM_setFreq (logicPWM *pPWM, TIM_HandleTypeDef *pTim, const uint16_t freq, uint16_t granularity)

  Set channel frequency for emulated PWM channel.
- FctERR logPWM_setDuty (logicPWM *pPWM, const uint16_t val)

  Set new duty cycle for emulated PWM channel.
- FctERR logPWM_getFreq (uint16_t *freq, const logicPWM *pPWM)

  Get channel frequency for emulated PWM channel.
- FctERR logPWM_getDutyCycle (float *duty, const logicPWM *pPWM)

  Get channel Duty Cycle for emulated PWM channel.
- void logPWM_handler (logicPWM *pPWM)

  Handler for an emulated PWM channel.

5.8.1 Detailed Description

Straightforward PWM handling.

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

Warning

Shall work for all STM32 F families, L families not totally covered

5.8.2 Typedef Documentation

5.8.2.1 logicPWM

```
typedef volatile struct logicPWM logicPWM
```

5.8.3 Function Documentation

5.8.3.1 deinit_TIM_Base()

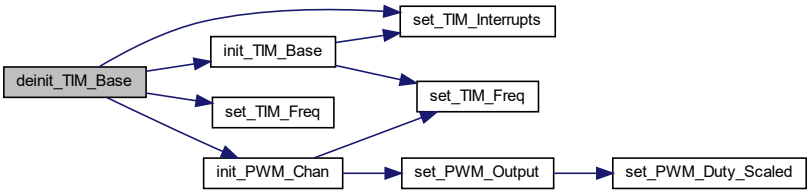
```
HAL_StatusTypeDef deinit_TIM_Base (
            TIM_HandleTypeDef * pTim )  [inline]
```

De-Init TIM module and start interruptions.

Parameters

in,out	*pTim*	- pointer to TIM instance

Here is the call graph for this function:



5.8.3.2 init_PWM_Chan()

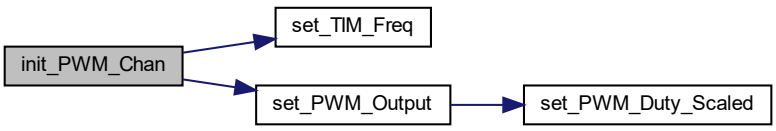
```
HAL_StatusTypeDef init_PWM_Chan (
            TIM_HandleTypeDef * pTim,
            const uint32_t chan,
            const uint16_t freq )
```

Init TIM PWM module channel with frequency and starts the channel.

Parameters

in,out	*pTim*	- pointer to TIM instance for PWM generation
in	*chan*	- Channel to write
in	*freq*	- Desired PWM frequency

Here is the call graph for this function:



Here is the caller graph for this function:

5.8.3.3 init_TIM_Base()

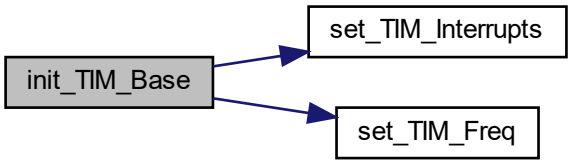
```
HAL_StatusTypeDef init_TIM_Base (
            TIM_HandleTypeDef * pTim,
            const uint32_t freq )
```

Init TIM module and start interruptions.

Parameters

in,out	*pTim*	- pointer to TIM instance
in	*freq*	- Desired TIM frequency

Here is the call graph for this function:



Here is the caller graph for this function:

5.8.3.4 logPWM_getDutyCycle()

```
FctERR logPWM_getDutyCycle (
            float * duty,
            const logicPWM * pPWM )
```

Get channel Duty Cycle for emulated PWM channel.

Parameters

in,out	*duty*	- pointer to duty cycle result
in,out	*pPWM*	- pointer to emulated PWM channel

Returns

Error code

Here is the caller graph for this function:

5.8.3.5 logPWM_getFreq()

```
FctERR logPWM_getFreq (
            uint16_t * freq,
            const logicPWM * pPWM )
```

Get channel frequency for emulated PWM channel.

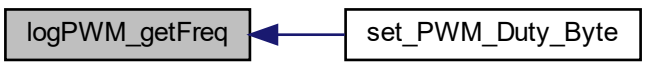
Parameters

in,out	*freq*	- pointer to frequency result
in,out	*pPWM*	- pointer to emulated PWM channel

Returns

Error code

Here is the caller graph for this function:



5.8.3.6 logPWM_handler()

```
void logPWM_handler (
            logicPWM * pPWM )
```

Handler for an emulated PWM channel.

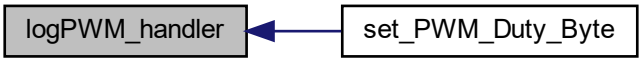
Warning

Shall be called directly from timer interrupt (HAL_TIM_PeriodElapsedCallback)

Parameters

in,out	*pPWM*	- pointer to emulated PWM channel

Here is the caller graph for this function:



5.8.3.7 logPWM_setDuty()

```
FctERR logPWM_setDuty (
            logicPWM * pPWM,
            const uint16_t val )
```

Set new duty cycle for emulated PWM channel.

Parameters

in,out	*pPWM*	- pointer to emulated PWM channel
in	*val*	- Duty cycle to apply

Returns

Error code

Here is the caller graph for this function:

5.8.3.8 logPWM_setFreq()

```
FctERR logPWM_setFreq (
            logicPWM * pPWM,
            TIM_HandleTypeDef * pTim,
            const uint16_t freq,
            uint16_t granularity )
```

Set channel frequency for emulated PWM channel.

Warning

For multiple PWMs on same timer with different frequencies, take care of init order (first configured channel will get TIM parameters precedence)

Parameters

in,out	*pPWM*	- pointer to emulated PWM channel
in,out	*pTim*	- pointer to TIM instance for Frequency computation
in	*freq*	- PWM frequency to apply
in	*granularity*	- PWM duty cycle granularity

Returns

Error code

Here is the caller graph for this function:

5.8.3.9 logPWM_setPin()

```
FctERR logPWM_setPin (
            logicPWM * pPWM,
            GPIO_TypeDef * GPIOx,
            const uint16_t GPIO_Pin,
            const bool polarity )
```

Set channel pin & polarity for emulated PWM channel.

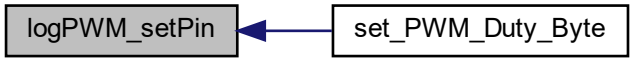
Parameters

in	*GPIOx*	- port for emulated PWM
in	*GPIO_Pin*	- pin for emulated PWM
in,out	*pPWM*	- pointer to emulated PWM channel
in	*polarity*	- 0: low polarity, 1: high polarity

Returns

Error code

Here is the caller graph for this function:



5.8.3.10 set_PWM_Duty_Byte()

```
HAL_StatusTypeDef set_PWM_Duty_Byte (
            const TIM_HandleTypeDef * pTim,
            const uint32_t chan,
            const uint8_t duty )  [inline]
```

Set TIM module PWM duty cycle (u8-bit value)

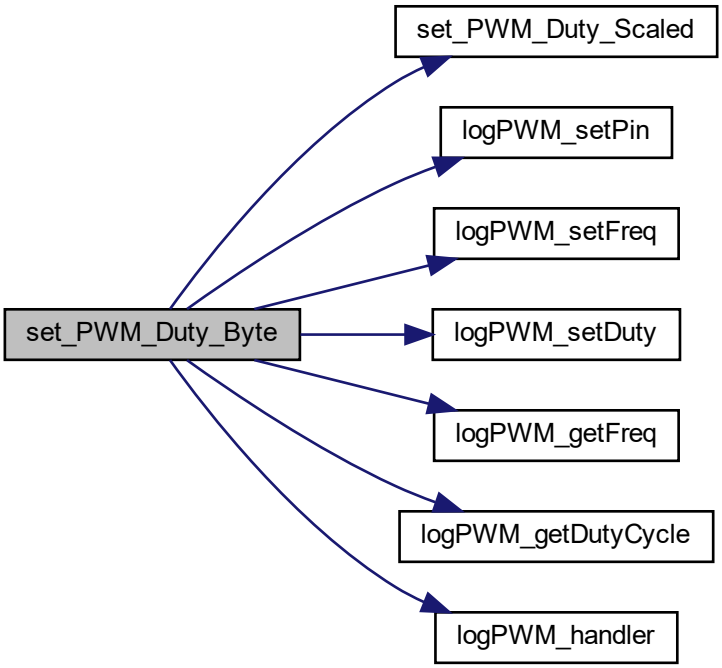
Parameters

in,out	*pTim*	- pointer to TIM instance for PWM generation
in	*chan*	- Channel to write
in	*duty*	- Scaled duty cycle value to write

Returns

HAL Status

Here is the call graph for this function:



5.8.3.11 set_PWM_Duty_Perc()

```
HAL_StatusTypeDef set_PWM_Duty_Perc (
            const TIM_HandleTypeDef * pTim,
            const uint32_t chan,
            const uint16_t duty )  [inline]
```

Set TIM module PWM duty cycle (percents)

Parameters

in,out	*pTim*	- pointer to TIM instance for PWM generation
in	*chan*	- Channel to write
in	*duty*	- Scaled duty cycle value to write

Returns

HAL Status

Here is the call graph for this function:

5.8.3.12 set_PWM_Duty_Scaled()

```
HAL_StatusTypeDef set_PWM_Duty_Scaled (
            const TIM_HandleTypeDef * pTim,
            const uint32_t chan,
            const uint16_t duty,
            const uint16_t scale )
```

Set TIM module PWM duty cycle (scaled)

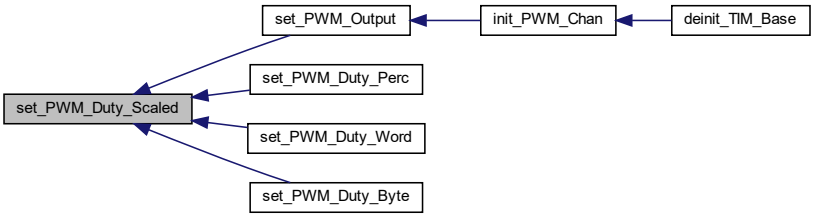
Parameters

in,out	*pTim*	- pointer to TIM instance for PWM generation
in	*chan*	- Channel to write
in	*duty*	- Scaled duty cycle value to write
in	*scale*	- Full scale value

Returns

HAL Status

Here is the caller graph for this function:



5.8.3.13 set_PWM_Duty_Word()

```
HAL_StatusTypeDef set_PWM_Duty_Word (
            const TIM_HandleTypeDef * pTim,
            const uint32_t chan,
            const uint16_t duty )  [inline]
```

Set TIM module PWM duty cycle (u16-bit value)

Parameters

in,out	*pTim*	- pointer to TIM instance for PWM generation
in	*chan*	- Channel to write
in	*duty*	- Scaled duty cycle value to write

Returns

HAL Status

Here is the call graph for this function:

5.8.3.14 set_PWM_Output()

```
HAL_StatusTypeDef set_PWM_Output (
            TIM_HandleTypeDef * pTim,
            const uint32_t chan,
            const bool on )  [inline]
```

Set PWM channel output on/off.

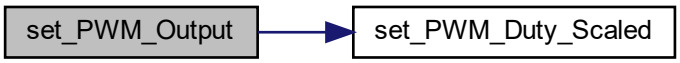
Parameters

in,out	*pTim*	- pointer to TIM instance for PWM generation
in	*chan*	- Channel to write
in	*on*	- Channel Output state 0: off, 1: on

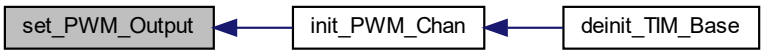
Returns

HAL Status

Here is the call graph for this function:



Here is the caller graph for this function:



5.8.3.15 set_TIM_Freq()

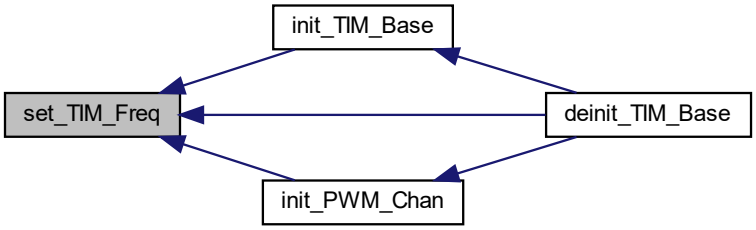
```
HAL_StatusTypeDef set_TIM_Freq (
            TIM_HandleTypeDef * pTim,
            const uint32_t freq )
```

Set TIM module frequency.

Parameters

in,out	*pTim*	- pointer to TIM instance for Frequency computation
in	*freq*	- Desired TIM frequency

Here is the caller graph for this function:



5.8.3.16 set_TIM_Interrupts()

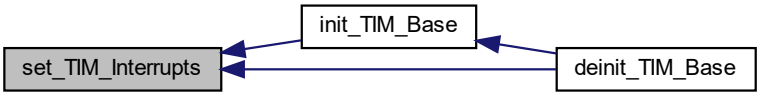
```
HAL_StatusTypeDef set_TIM_Interrupts (
            TIM_HandleTypeDef * pTim,
            const bool on )  [inline]
```

Start TIM module interrupts.

Parameters

in,out	*pTim*	- pointer to TIM instance
in	*on*	- Time Interrupts 0: off, 1: on

Here is the caller graph for this function:

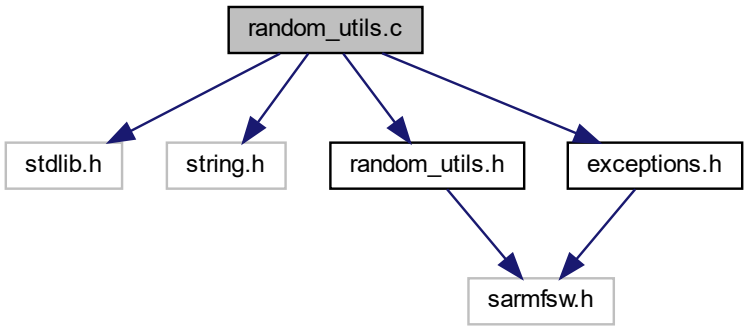


5.9 random_utils.c File Reference

(A little less pseudo) random numbers generation utilities

```
#include <stdlib.h>
#include <string.h>
#include "random_utils.h"
#include "exceptions.h"
```

Include dependency graph for random_utils.c:



Functions

- uint32_t random_Get (uint32_t start)
  Generate a random number based on STM32 unique ID.

5.9.1 Detailed Description

(A little less pseudo) random numbers generation utilities

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

Note

Randomness is enhanced between MCUs (using UID), through calls and time

Warning

Unfortunately, after reset, assuming call will happen at same clock tick, random_Get will give same result (unless start saved/restore from some storage is given as parameter at first call)

5.9.2 Function Documentation

5.9.2.1 random_Get()

```
uint32_t random_Get (
            uint32_t start )
```

Generate a random number based on STM32 unique ID.

Parameters

in	*start*	- Value for a first seed (if a value has been stored for next reset)

Returns

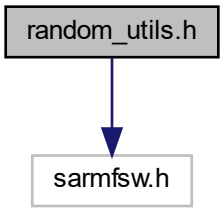
Generated random number

5.10 random_utils.h File Reference

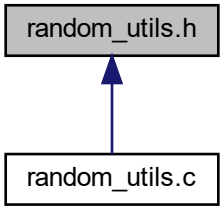
(A little less pseudo) random numbers generation utilities

```
#include "sarmfsw.h"
```

Include dependency graph for random_utils.h:



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



Functions

- uint32_t random_Get (uint32_t start)

  Generate a random number based on STM32 unique ID.

5.10.1 Detailed Description

(A little less pseudo) random numbers generation utilities

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

5.10.2 Function Documentation

5.10.2.1 random_Get()

```
uint32_t random_Get (
            uint32_t start )
```

Generate a random number based on STM32 unique ID.

Parameters

in	*start*	- Value for a first seed (if a value has been stored for next reset)

Returns

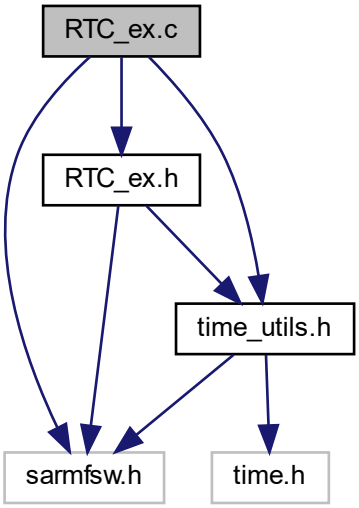
Generated random number

5.11 RTC_ex.c File Reference

Basic RTC handling.

```
#include "sarmfsw.h"
#include "time_utils.h"
#include "RTC_ex.h"
```

Include dependency graph for RTC_ex.c:



Functions

- FctERR RTC_SetTime (const DateTime *time_new)
  Sends new time to RTC peripheral.
- FctERR RTC_GetTime (DateTime *time_now)
  Get time from RTC peripheral.

5.11.1 Detailed Description

Basic RTC handling.

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

5.11.2 Function Documentation

5.11.2.1 RTC_GetTime()

```
FctERR RTC_GetTime (
            DateTime * time_now )
```

Get time from RTC peripheral.

Parameters

in,out	*time_now*	- pointer to DateTime instance

Returns

FctERR - error code

5.11.2.2 RTC_SetTime()

```
FctERR RTC_SetTime (
            const DateTime * time_new )
```

Sends new time to RTC peripheral.

Parameters

in	*time_new*	- pointer to DateTime instance

Returns

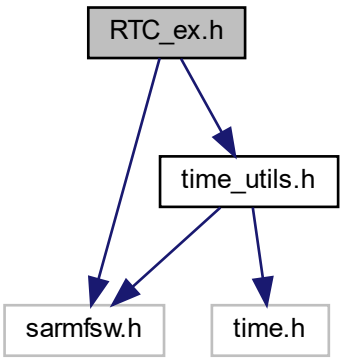
FctERR - error code

5.12 RTC_ex.h File Reference

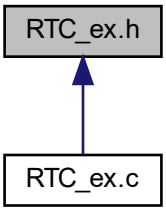
Basic RTC handling.

```
#include "sarmfsw.h"
#include "time_utils.h"
```

Include dependency graph for RTC_ex.h:



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



Functions

- FctERR RTC_SetTime (const DateTime *time_new)

  Sends new time to RTC peripheral.
- FctERR RTC_GetTime (DateTime *time_now)

  Get time from RTC peripheral.

5.12.1 Detailed Description

Basic RTC handling.

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

5.12.2 Function Documentation

5.12.2.1 RTC_GetTime()

```
FctERR RTC_GetTime (
            DateTime * time_now )
```

Get time from RTC peripheral.

Parameters

in,out	*time_now*	- pointer to DateTime instance

Returns

FctERR - error code

5.12.2.2 RTC_SetTime()

```
FctERR RTC_SetTime (
            const DateTime * time_new )
```

Sends new time to RTC peripheral.

Parameters

in	*time_new*	- pointer to DateTime instance

Returns

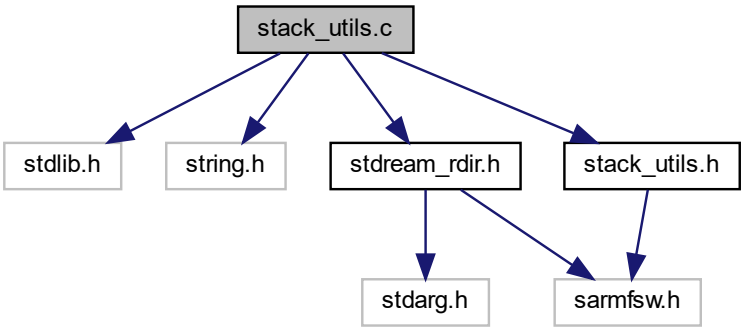
FctERR - error code

5.13 stack_utils.c File Reference

Stack utilities.

```
#include <stdlib.h>
#include <string.h>
#include "stdream_rdir.h"
#include "stack_utils.h"
```

Include dependency graph for stack_utils.c:



Functions

- void print_stack_address (void)

  Prints main stack address.
- void print_global_regs (void)

  Print contents of ARM registers.

5.13.1 Detailed Description

Stack utilities.

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

5.13.2 Function Documentation

5.13.2.1 print_global_regs()

```
void print_global_regs (
             void  )
```

Print contents of ARM registers.

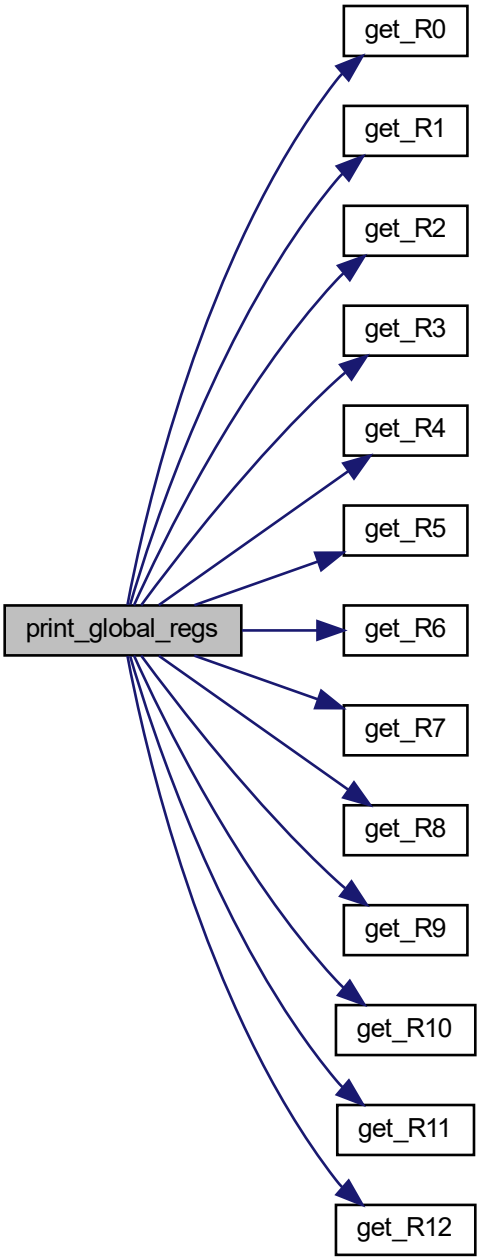
Note

This function is for debug purposes while running only

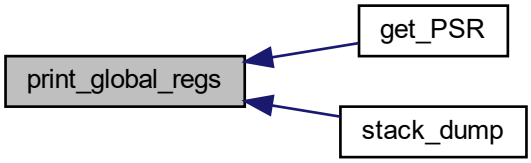
Returns

Nothing

Here is the call graph for this function:



Here is the caller graph for this function:



5.13.2.2 print_stack_address()

```
void print_stack_address (
            void  )
```

Prints main stack address.

Note

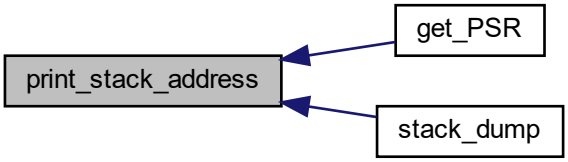
This function is for debug purposes while running only

Returns

Nothing

Here is the call graph for this function:

Here is the caller graph for this function:

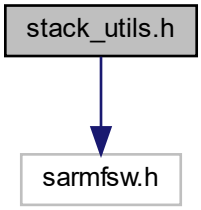


5.14 stack_utils.h File Reference

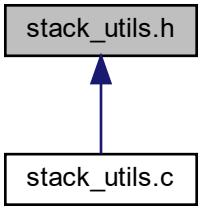
Stack utilities.

```
#include "sarmfsw.h"
```

Include dependency graph for stack_utils.h:



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



Functions

- uint32_t get_MSP (void)
  Get Main Stack Pointer value.
- uint32_t get_SP (void)
  Get Current Stack Pointer value.
- uint32_t * get_pSP (void)
  Get pointer to the current stack.
- uint32_t * get_pMSP (void)
  Get pointer to the main stack.
- uint32_t get_R0 (void)
  Get content of R0 register.
- uint32_t get_R1 (void)
  Get content of R1 register.
- uint32_t get_R2 (void)

Get content of R2 register.

- uint32_t get_R3 (void)

  Get content of R3 register.
- uint32_t get_R4 (void)

  Get content of R4 register.
- uint32_t get_R5 (void)

  Get content of R5 register.
- uint32_t get_R6 (void)

  Get content of R6 register.
- uint32_t get_R7 (void)

  Get content of R7 register.
- uint32_t get_R8 (void)

  Get content of R8 register.
- uint32_t get_R9 (void)

  Get content of R9 register.
- uint32_t get_R10 (void)

  Get content of R10 register.
- uint32_t get_R11 (void)

  Get content of R11 register.
- uint32_t get_R12 (void)

  Get content of R12 register.
- uint32_t get_LR (void)

  Get content of link register.
- uint32_t get_PSR (void)

  Get content of xPSR.
- void print_stack_address (void)

  Prints main stack address.
- void print_global_regs (void)

  Print contents of ARM registers.
- void stack_dump (void)

  prints contents of global registers & stack address

5.14.1 Detailed Description

Stack utilities.

Author

SMFSW

Copyright

5.14.2 Function Documentation

5.14.2.1 get_LR()

```
uint32_t get_LR (
             void  )  [inline]
```

Get content of link register.

Returns

Link register value

5.14.2.2 get_MSP()

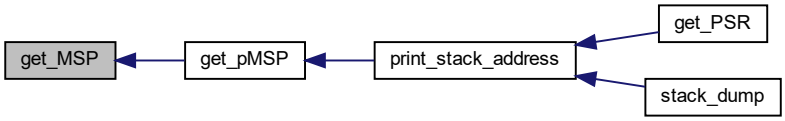
```
uint32_t get_MSP (
             void  )  [inline]
```

Get Main Stack Pointer value.

Returns

Main Stack Pointer value

Here is the caller graph for this function:



5.14.2.3 get_pMSP()

```
uint32_t* get_pMSP (
             void  )  [inline]
```

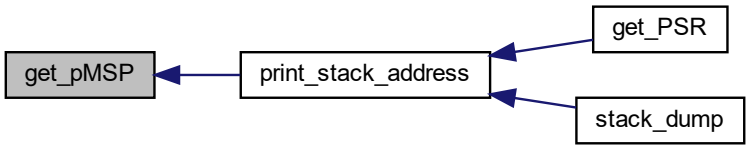
Get pointer to the main stack.

Returns

Main stack pointer

Here is the call graph for this function:

Here is the caller graph for this function:



5.14.2.4 get_pSP()

```
uint32_t* get_pSP (
            void  )  [inline]
```

Get pointer to the current stack.

Returns

Current stack pointer

Here is the call graph for this function:

5.14.2.5 get_PSR()

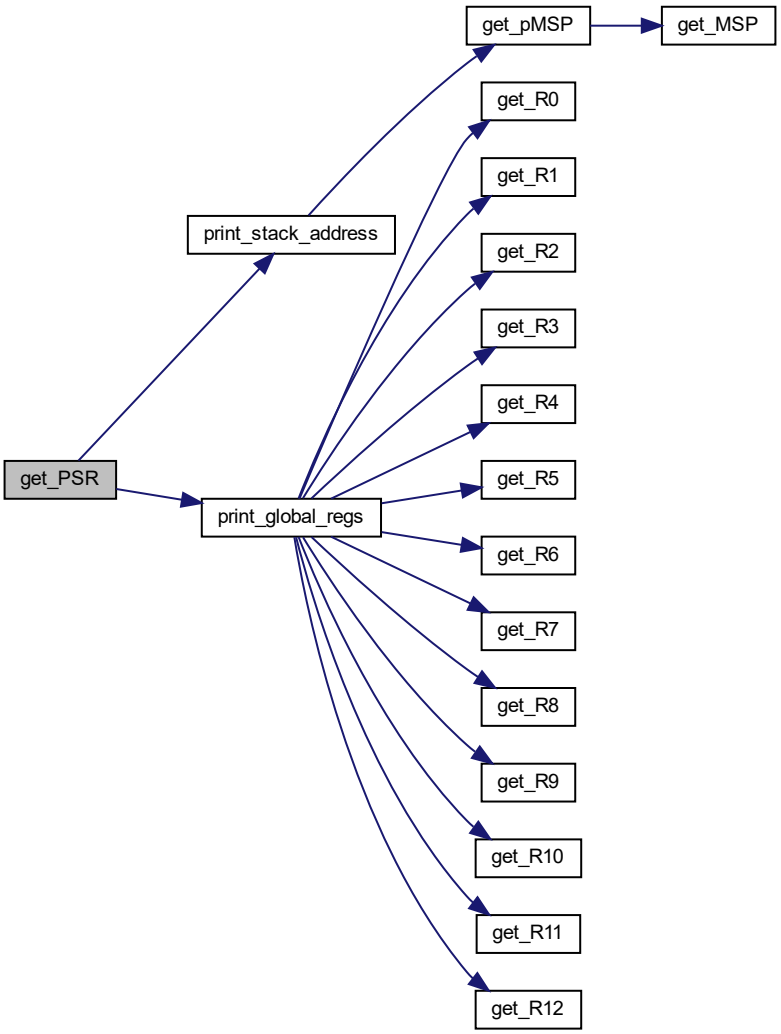
```
uint32_t get_PSR (
            void  )  [inline]
```

Get content of xPSR.

Returns

Current program status register value

Here is the call graph for this function:



5.14.2.6 get_R0()

```
uint32_t get_R0 (
            void  )  [inline]
```

Get content of R0 register.

Returns

R0 register value

Here is the caller graph for this function:

5.14.2.7 get_R1()

```
uint32_t get_R1 (
            void  )  [inline]
```

Get content of R1 register.

Returns

R1 register value

Here is the caller graph for this function:

5.14.2.8 get_R10()

```
uint32_t get_R10 (
            void  )  [inline]
```

Get content of R10 register.

Returns

R10 register value

Here is the caller graph for this function:

5.14.2.9 get_R11()

```
uint32_t get_R11 (
            void  )  [inline]
```

Get content of R11 register.

Returns

R11 register value

Here is the caller graph for this function:

5.14.2.10 get_R12()

```
uint32_t get_R12 (
            void  )  [inline]
```

Get content of R12 register.

Returns

R12 register value

Here is the caller graph for this function:

5.14.2.11 get_R2()

```
uint32_t get_R2 (
            void  )  [inline]
```

Get content of R2 register.

Returns

R2 register value

Here is the caller graph for this function:

5.14.2.12 get_R3()

```
uint32_t get_R3 (
            void  )  [inline]
```

Get content of R3 register.

Returns

R3 register value

Here is the caller graph for this function:

5.14.2.13 get_R4()

```
uint32_t get_R4 (
            void  )  [inline]
```

Get content of R4 register.

Returns

R4 register value

Here is the caller graph for this function:

5.14.2.14 get_R5()

```
uint32_t get_R5 (
            void  )  [inline]
```

Get content of R5 register.

Returns

R5 register value

Here is the caller graph for this function:

5.14.2.15 get_R6()

```
uint32_t get_R6 (
            void  )  [inline]
```

Get content of R6 register.

Returns

R6 register value

Here is the caller graph for this function:

5.14.2.16 get_R7()

```
uint32_t get_R7 (
            void  )  [inline]
```

Get content of R7 register.

Returns

R7 register value

Here is the caller graph for this function:

5.14.2.17 get_R8()

```
uint32_t get_R8 (
            void  )  [inline]
```

Get content of R8 register.

Returns

R8 register value

Here is the caller graph for this function:

5.14.2.18 get_R9()

```
uint32_t get_R9 (
            void  )  [inline]
```

Get content of R9 register.

Returns

R9 register value

Here is the caller graph for this function:

5.14.2.19 get_SP()

```
uint32_t get_SP (
            void  )  [inline]
```

Get Current Stack Pointer value.

Returns

Current Stack Pointer value

Here is the caller graph for this function:

5.14.2.20 print_global_regs()

```
void print_global_regs (
             void  )
```

Print contents of ARM registers.

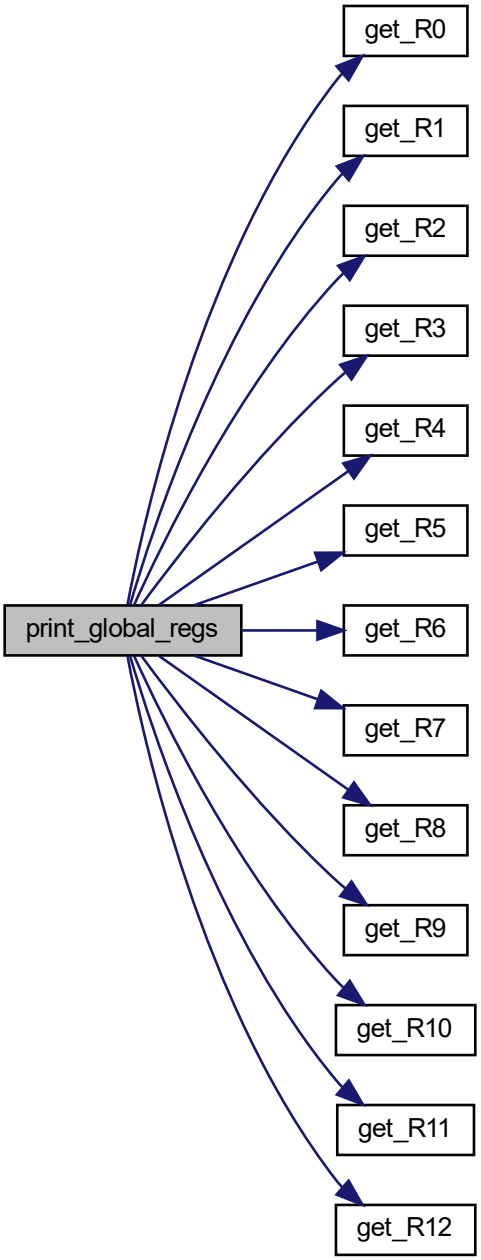
Note

This function is for debug purposes while running only

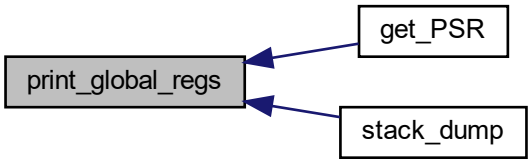
Returns

Nothing

Here is the call graph for this function:



Here is the caller graph for this function:



5.14.2.21 print_stack_address()

```
void print_stack_address (
            void  )
```

Prints main stack address.

Note

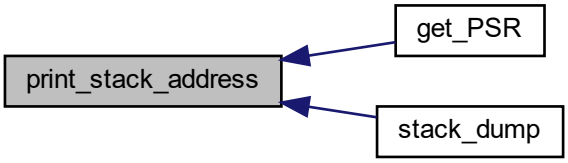
This function is for debug purposes while running only

Returns

Nothing

Here is the call graph for this function:

Here is the caller graph for this function:



5.14.2.22 stack_dump()

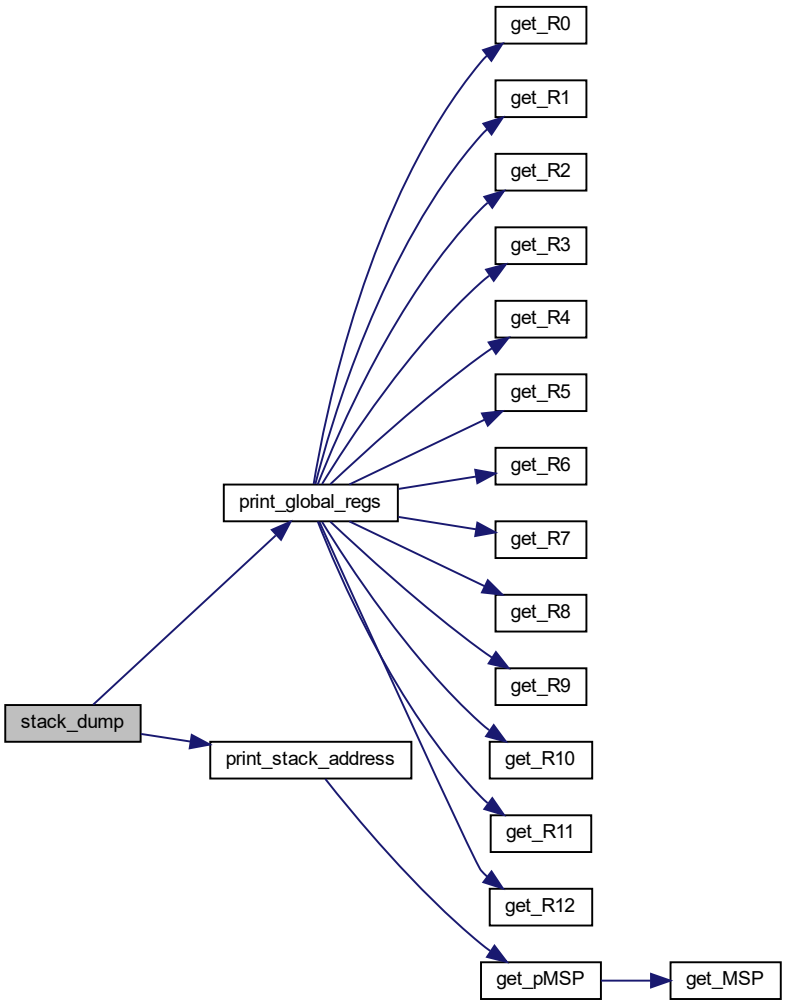
```
void stack_dump (
             void  )  [inline]
```

prints contents of global registers & stack address

Returns

Nothing

Here is the call graph for this function:

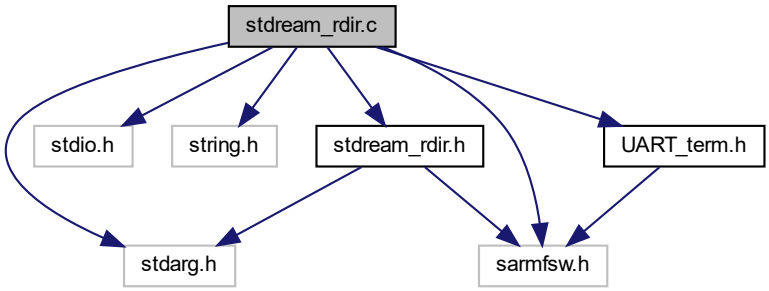


5.15 stdream_rdir.c File Reference

Stream redirection.

```
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
#include "sarmfsw.h"
#include "UART_term.h"
#include "stdream_rdir.h"
```

Include dependency graph for stdream_rdir.c:



Functions

- void ITM_port_send (const int port, const char ∗str, const int len)

  Send string to ITM port.
- int printf_ITM (const char ∗str,...)

  printf like redirected to ITM port 0
- int vprintf_ITM (const char ∗str, va_list args)

  printf like redirected to ITM port 0
- int printf_redir (const char ∗str,...)

  printf like redirected to DBG_SERIAL UART and/or ITM port 0
- int vprintf_redir (const char ∗str, va_list args)

  printf like redirected to DBG_SERIAL UART and/or ITM port 0

Variables

- char dbg_msg_out [128] = ""

  stdream buffer for output
- char dbg_msg_in [32+1] = ""

  stdream buffer for input

5.15.1 Detailed Description

Stream redirection.

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

Note

define ITM_REDIRECT in compiler defines for stings to be printed to ITM0 port
define UART_REDIRECT and DBG_SERIAL in compiler defines with an UART instance to send printf likes strings to UART

5.15.2 Function Documentation

5.15.2.1 ITM_port_send()

```
void ITM_port_send (
            const int port,
            const char * str,
            const int len )
```

Send string to ITM port.

Parameters

in	*port*	- ITM port number
in	*str*	- pointer to message to send
in	*len*	- length of message to send

5.15.2.2 printf_ITM()

```
int printf_ITM (
            const char * str,
             ...  )
```

printf like redirected to ITM port 0

Parameters

in	*str*	- pointer to string to send
in	...	- Variadic string arguments

Returns

Function status

Return values

0	- OK

5.15.2.3 printf_redir()

```
int printf_redir (
            const char * str,
             ...  )
```

printf like redirected to DBG_SERIAL UART and/or ITM port 0

Parameters

in	*str*	- pointer to string to send
in	*...*	- Variadic string arguments

Returns

Function status

Return values

-1	- Problem occured
0	- OK

Here is the call graph for this function:

5.15.2.4 vprintf_ITM()

```
int vprintf_ITM (
            const char * str,
            va_list args )
```

printf like redirected to ITM port 0

Parameters

in	*str*	- pointer to string to send
in	*args*	- Variadic string arguments

Returns

Function status

Return values

0	- OK

5.15.2.5 vprintf_redir()

```
int vprintf_redir (
            const char * str,
            va_list args )
```

printf like redirected to DBG_SERIAL UART and/or ITM port 0

Parameters

in	*str*	- pointer to string to send
in	*args*	- Variadic string arguments

Returns

Function status

Return values

-1	- Problem occured
0	- OK

Here is the call graph for this function:

5.15.3 Variable Documentation

5.15.3.1 dbg_msg_in

```
char dbg_msg_in[32+1] = ""
```

stdream buffer for input

Warning

dbg_msg_in buffer for stdream is limited to **SZ_DBG_IN**

Note

dbg_msg_in is only related to UART_term

5.15.3.2 dbg_msg_out

```
char dbg_msg_out[128] = ""
```

stdream buffer for output

Warning

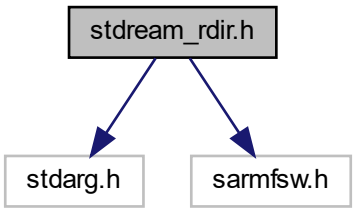
dbg_msg_out buffer for stdream is limited to **SZ_DBG_OUT**

5.16 stdream_rdir.h File Reference

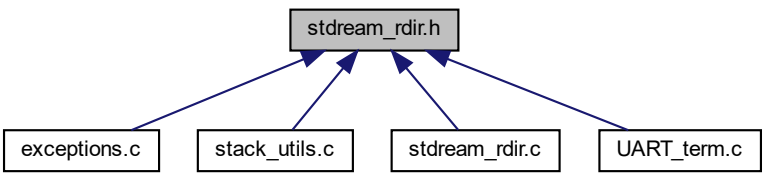
Stream redirection.

```
#include <stdarg.h>
#include "sarmfsw.h"
```

Include dependency graph for stdream_rdir.h:



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



Macros

- #define printf printf_redir
  Shadowing printf.
- #define vprintf vprintf_redir
  Shadowing vprintf.
- #define SZ_DBG_OUT 128
  DEBUG send buffer size.
- #define SZ_DBG_IN 32
  DEBUG receive buffer size.

Functions

- void ITM_port_send (const int port, const char *str, const int len)
  Send string to ITM port.
- int printf_ITM (const char *str,...)
  printf like redirected to ITM port 0
- int vprintf_ITM (const char *str, va_list args)
  printf like redirected to ITM port 0
- int printf_redir (const char *str,...)
  printf like redirected to DBG_SERIAL UART and/or ITM port 0
- int vprintf_redir (const char *str, va_list args)
  printf like redirected to DBG_SERIAL UART and/or ITM port 0

Variables

- char dbg_msg_out [128]
  stdream buffer for output
- char dbg_msg_in [32+1]
  stdream buffer for input

5.16.1 Detailed Description

Stream redirection.

Author

SMFSW

Copyright

Note

define ITM_REDIRECT in compiler defines for stings to be printed to ITM0 port
define UART_REDIRECT and DBG_SERIAL in compiler defines with an UART instance to send printf likes strings to UART

5.16.2 Macro Definition Documentation

5.16.2.1 printf

```
#define printf printf_redir
```

Shadowing printf.

5.16.2.2 SZ_DBG_IN

```
#define SZ_DBG_IN 32
```

DEBUG receive buffer size.

5.16.2.3 SZ_DBG_OUT

```
#define SZ_DBG_OUT 128
```

DEBUG send buffer size.

5.16.2.4 vprintf

```
#define vprintf vprintf_redir
```

Shadowing vprintf.

5.16.3 Function Documentation

5.16.3.1 ITM_port_send()

```
void ITM_port_send (
            const int port,
            const char * str,
            const int len )
```

Send string to ITM port.

Parameters

in	*port*	- ITM port number
in	*str*	- pointer to message to send
in	*len*	- length of message to send

5.16.3.2 printf_ITM()

```
int printf_ITM (
             const char * str,
              ...  )
```

printf like redirected to ITM port 0

Parameters

in	*str*	- pointer to string to send
in	...	- Variadic string arguments

Returns

Function status

Return values

0	- OK

5.16.3.3 printf_redir()

```
int printf_redir (
             const char * str,
              ...  )
```

printf like redirected to DBG_SERIAL UART and/or ITM port 0

Parameters

in	*str*	- pointer to string to send
in	...	- Variadic string arguments

Returns

Function status

Return values

-1	- Problem occured
0	- OK

Here is the call graph for this function:

5.16.3.4 vprintf_ITM()

```
int vprintf_ITM (
            const char * str,
            va_list args )
```

printf like redirected to ITM port 0

Parameters

in	*str*	- pointer to string to send
in	*args*	- Variadic string arguments

Returns

Function status

Return values

0	- OK

5.16.3.5 vprintf_redir()

```
int vprintf_redir (
            const char * str,
            va_list args )
```

printf like redirected to DBG_SERIAL UART and/or ITM port 0

Parameters

in	*str*	- pointer to string to send
in	*args*	- Variadic string arguments

Returns

Function status

Return values

-1	- Problem occured
0	- OK

Here is the call graph for this function:

5.16.4 Variable Documentation

5.16.4.1 dbg_msg_in

```
char dbg_msg_in[32+1]
```

stdream buffer for input

Warning

dbg_msg_in buffer for stdream is limited to **SZ_DBG_IN**

Note

dbg_msg_in is only related to UART_term

5.16.4.2 dbg_msg_out

```
char dbg_msg_out[128]
```

stdream buffer for output

Warning

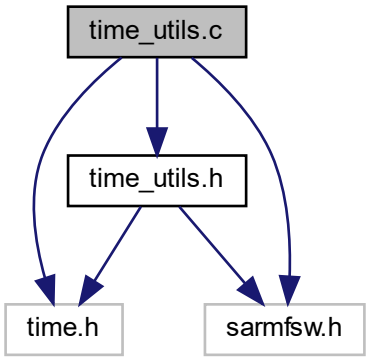
dbg_msg_out buffer for stdream is limited to **SZ_DBG_OUT**

5.17 time_utils.c File Reference

Time related utilities.

```
#include <time.h>
#include "sarmfsw.h"
#include "time_utils.h"
```

Include dependency graph for time_utils.c:



Functions

- DateTime time_t2DateTime (const time_t time)
  Convert time_t to DateTime.
- time_t DateTime2time_t (const DateTime *time)
  Convert DateTime to time_t.
- DateTime diffDateTime (const DateTime *time2, const DateTime *time1)
  Calculate DateTime difference.

5.17.1 Detailed Description

Time related utilities.

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

5.17.2 Function Documentation

5.17.2.1 DateTime2time_t()

```
time_t DateTime2time_t (
            const DateTime * time )
```

Convert DateTime to time_t.

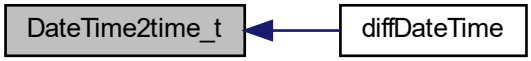
Parameters

in	*time*	- DateTime representation (broken down time)

Returns

time_t representation

Here is the caller graph for this function:



5.17.2.2 diffDateTime()

```
DateTime diffDateTime (
            const DateTime * time2,
            const DateTime * time1 )
```

Calculate DateTime difference.

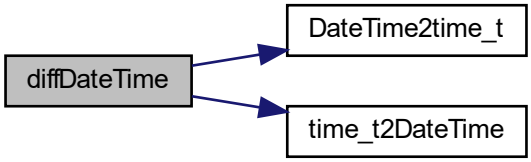
Parameters

in	*time2*	- pointer to closest DateTime representation (broken down time)
in	*time1*	- pointer to oldest DateTime representation (broken down time)

Returns

DateTime difference

Here is the call graph for this function:



5.17.2.3 time_t2DateTime()

```
DateTime time_t2DateTime (
            const time_t time )
```

Convert time_t to DateTime.

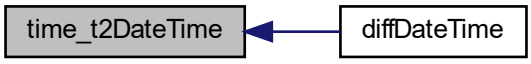
Parameters

in	*time*	- time_t representation

Returns

Broken down time representation (DateTime)

Here is the caller graph for this function:

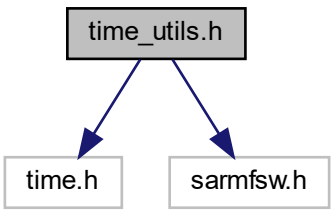


5.18 time_utils.h File Reference

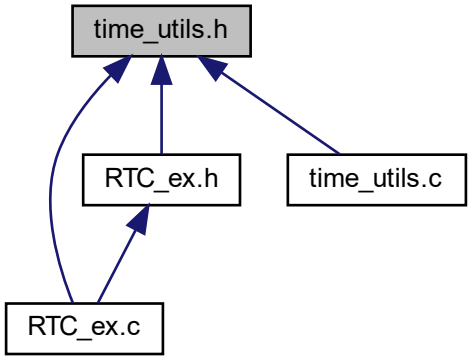
Time related utilities.

```
#include <time.h>
#include "sarmfsw.h"
```

Include dependency graph for time_utils.h:



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



Data Structures

- struct DateTime

  Basic Date & Time struct.

Typedefs

- typedef struct DateTime DateTime

Functions

- DateTime time_t2DateTime (const time_t time)

  Convert time_t to DateTime.
- time_t DateTime2time_t (const DateTime *time)

  Convert DateTime to time_t.
- DateTime diffDateTime (const DateTime *time2, const DateTime *time1)

  Calculate DateTime difference.

5.18.1 Detailed Description

Time related utilities.

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

5.18.2 Typedef Documentation

5.18.2.1 DateTime

```
typedef struct DateTime DateTime
```

5.18.3 Function Documentation

5.18.3.1 DateTime2time_t()

```
time_t DateTime2time_t (
            const DateTime * time )
```

Convert DateTime to time_t.

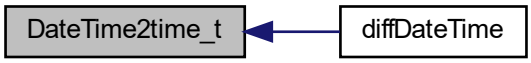
Parameters

in	*time*	- DateTime representation (broken down time)

Returns

time_t representation

Here is the caller graph for this function:



5.18.3.2 diffDateTime()

```
DateTime diffDateTime (
            const DateTime * time2,
            const DateTime * time1 )
```

Calculate DateTime difference.

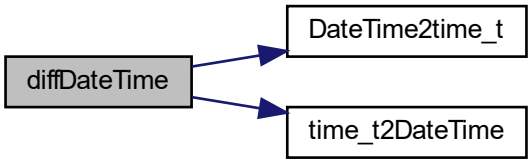
Parameters

in	*time2*	- pointer to closest DateTime representation (broken down time)
in	*time1*	- pointer to oldest DateTime representation (broken down time)

Returns

DateTime difference

Here is the call graph for this function:



5.18.3.3 time_t2DateTime()

```
DateTime time_t2DateTime (
            const time_t time )
```

Convert time_t to DateTime.

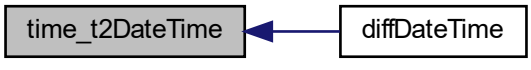
Parameters

in	*time*	- time_t representation

Returns

Broken down time representation (DateTime)

Here is the caller graph for this function:

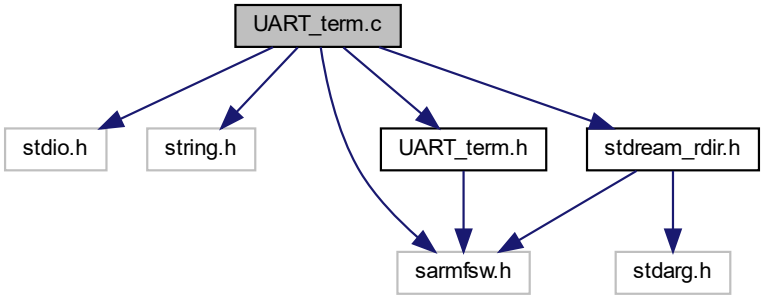


5.19 UART_term.c File Reference

UART terminal.

```
#include <stdio.h>
#include <string.h>
#include "sarmfsw.h"
#include "stdream_rdir.h"
#include "UART_term.h"
```

Include dependency graph for UART_term.c:



Functions

- FctERR SERIAL_DBG_Launch_It_Rx (UART_HandleTypeDef *huart)

  Start UART SERIAL DEBUG Rx interruptions.
- FctERR SERIAL_DBG_Flush_RxBuf (UART_HandleTypeDef *huart)

  Clear buffer in used for SERIAL DEBUG.
- FctERR SERIAL_DBG_Message_Handler (const char *msg, const uint8_t len)

  Treat fully received message.
- void HAL_UART_RxCpltCallback (UART_HandleTypeDef *huart)

  Rx Transfer completed callback.
- void HAL_UART_TxCpltCallback (UART_HandleTypeDef *huart)

  Tx Transfer completed callback (clear uart_out buffer)

Variables

- char breakout_char = '!'

  breakout char (message complete)
- UART_HandleTypeDef * dbg_uart = 1

  Instance of UART debug terminal.

5.19.1 Detailed Description

UART terminal.

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

Note

UART redirection is enabled when UART_REDIRECT symbol is defined at project level
define DBG_SERIAL in compiler defines with an UART instance to send printf likes strings to UART

5.19.2 Function Documentation

5.19.2.1 HAL_UART_RxCpltCallback()

```
void HAL_UART_RxCpltCallback (
            UART_HandleTypeDef * huart )
```

Rx Transfer completed callback.

Parameters

huart	UART handle.

Return values

None	

5.19.2.2 HAL_UART_TxCpltCallback()

```
void HAL_UART_TxCpltCallback (
            UART_HandleTypeDef * huart )
```

Tx Transfer completed callback (clear uart_out buffer)

Parameters

huart	- UART handle

Return values

None	

5.19.2.3 SERIAL_DBG_Flush_RxBuf()

```
FctERR SERIAL_DBG_Flush_RxBuf (
            UART_HandleTypeDef * huart )
```

Clear buffer in used for SERIAL DEBUG.

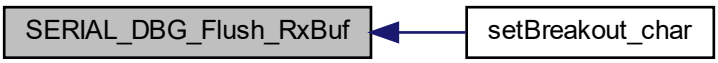
Parameters

in	*huart*	- UART handle (reserved for future use if needed)

Returns

Error code

Here is the caller graph for this function:



5.19.2.4 SERIAL_DBG_Launch_It_Rx()

```
FctERR SERIAL_DBG_Launch_It_Rx (
            UART_HandleTypeDef * huart )
```

Start UART SERIAL DEBUG Rx interruptions.

Parameters

in	*huart*	- UART handle

Returns

Error code

Here is the caller graph for this function:

5.19.2.5 SERIAL_DBG_Message_Handler()

```
FctERR SERIAL_DBG_Message_Handler (
            const char * msg,
            const uint8_t len )
```

Treat fully received message.

Weak Functions This function is implemented as weak to be implemented in projects (weak one only prints & flushes the buffer)

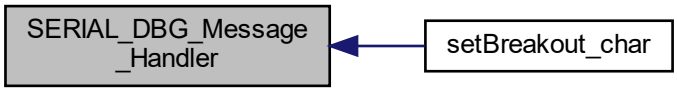
Parameters

in	*msg*	- pointer to received message
in	*len*	- received message length

Returns

Error code

Here is the caller graph for this function:



5.19.3 Variable Documentation

5.19.3.1 breakout_char

```
char breakout_char = '!'
```

breakout char (message complete)

Note

Default user breakout char set to '!' and '\r' is built-in default breakout char

5.19.3.2 dbg_uart

```
UART_HandleTypeDef* dbg_uart = 1
```

Instance of UART debug terminal.

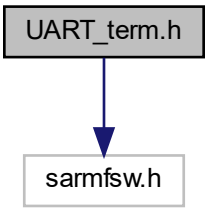
UART debug terminal instance.

5.20 UART_term.h File Reference

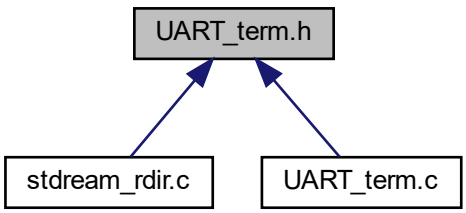
UART terminal header.

```
#include "sarmfsw.h"
```

Include dependency graph for UART_term.h:



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



Macros

- #define STDREAM__UART_TX_IT

  To be defined to send to uart using interrupts.

Functions

- char getBreakout_char (void)

  Get UART Rx breakout character.
- void setBreakout_char (const char breakout)

  Set a new breakout character.
- FctERR SERIAL_DBG_Launch_It_Rx (UART_HandleTypeDef *huart)

  Start UART SERIAL DEBUG Rx interruptions.
- FctERR SERIAL_DBG_Flush_RxBuf (UART_HandleTypeDef *huart)

  Clear buffer in used for SERIAL DEBUG.
- FctERR SERIAL_DBG_Message_Handler (const char *msg, const uint8_t len)

  Treat fully received message.
- FctERR SERIAL_DBG_Wait_Ready (UART_HandleTypeDef *huart)

  Waiting for UART global state to be ready for next transmission.
- HAL_StatusTypeDef SERIAL_DBG_Send (UART_HandleTypeDef *huart, const char *str, const int len)

  Sends string to UART.

Variables

- char breakout_char

  breakout char (message complete)
- UART_HandleTypeDef * dbg_uart

  UART debug terminal instance.

5.20.1 Detailed Description

UART terminal header.

Author

SMFSW

Copyright

MIT (c) 2017-2018, SMFSW

Note

UART redirection is enabled when UART_REDIRECT symbol is defined at project level
define DBG_SERIAL in compiler defines with an UART instance to send printf likes strings to UART

5.20.2 Macro Definition Documentation

5.20.2.1 STDREAM__UART_TX_IT

```
#define STDREAM__UART_TX_IT
```

To be defined to send to uart using interrupts.

5.20.3 Function Documentation

5.20.3.1 getBreakout_char()

```
char getBreakout_char (
            void  )  [inline]
```

Get UART Rx breakout character.

Returns

Breakout character

5.20.3.2 SERIAL_DBG_Flush_RxBuf()

```
FctERR SERIAL_DBG_Flush_RxBuf (
            UART_HandleTypeDef * huart )
```

Clear buffer in used for SERIAL DEBUG.

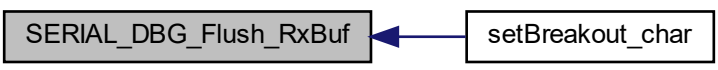
Parameters

in	*huart*	- UART handle (reserved for future use if needed)

Returns

Error code

Here is the caller graph for this function:



5.20.3.3 SERIAL_DBG_Launch_It_Rx()

```
FctERR SERIAL_DBG_Launch_It_Rx (
            UART_HandleTypeDef * huart )
```

Start UART SERIAL DEBUG Rx interruptions.

Parameters

in	*huart*	- UART handle

Returns

Error code

Here is the caller graph for this function:

5.20.3.4 SERIAL_DBG_Message_Handler()

```
FctERR SERIAL_DBG_Message_Handler (
            const char * msg,
            const uint8_t len )
```

Treat fully received message.

Weak Functions This function is implemented as weak to be implemented in projects (weak one only prints & flushes the buffer)

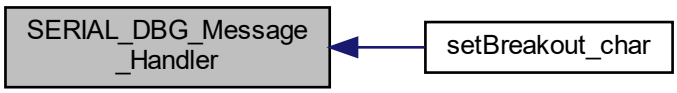
Parameters

in	*msg*	- pointer to received message
in	*len*	- received message length

Returns

Error code

Here is the caller graph for this function:



5.20.3.5 SERIAL_DBG_Send()

```
HAL_StatusTypeDef SERIAL_DBG_Send (
            UART_HandleTypeDef * huart,
            const char * str,
            const int len )  [inline]
```

Sends string to UART.

Parameters

in	*huart*	- UART handle
in	*str*	- pointer to string to send
in	*len*	- length of string

Returns

HAL Status

5.20.3.6 SERIAL_DBG_Wait_Ready()

```
FctERR SERIAL_DBG_Wait_Ready (
            UART_HandleTypeDef * huart )  [inline]
```

Waiting for UART global state to be ready for next transmission.

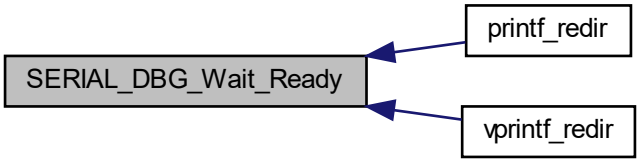
Parameters

in	*huart*	- UART handle

Returns

Error code

Here is the caller graph for this function:



5.20.3.7 setBreakout_char()

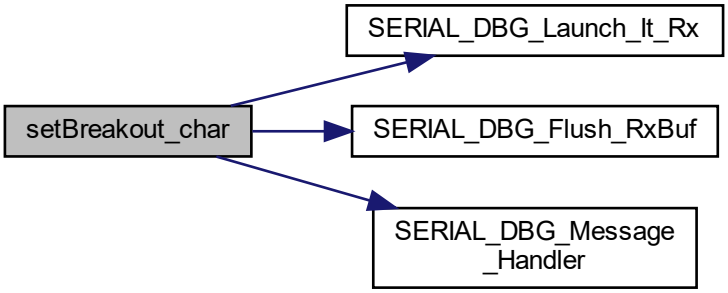
```
void setBreakout_char (
            const char breakout )  [inline]
```

Set a new breakout character.

Parameters

in	*breakout*	- new breakout character

Here is the call graph for this function:



5.20.4 Variable Documentation

5.20.4.1 breakout_char

```
char breakout_char
```

breakout char (message complete)

Note

Default user breakout char set to '!' and '\r' is built-in default breakout char

5.20.4.2 dbg_uart

```
UART_HandleTypeDef* dbg_uart
```

UART debug terminal instance.

UART debug terminal instance.

Index