

# Triggers

---

Copyright (c) 2010-2016 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to [youngwlim@hotmail.com](mailto:youngwlim@hotmail.com).

This document was produced by using OpenOffice.

# Trigger.h and Trigger.c

---

Based on

Implementation of generic triggers.  
Erich Styger, erich.styger@hslu.ch  
21.03.2011

<http://www.steinerberg.com/EmbeddedComponents/BookSrc/Trigger.c>  
<http://www.steinerberg.com/EmbeddedComponents/BookSrc/Trigger.h>

# Overview of Trigger.h

```
typedef enum { ??? }      TRG_TriggerKind;  
typedef uint16_t          TRG_TriggerTime;  
typedef void *            TRG_CallbackDataPtr;  
typedef void (*TRG_Callback)(TRG_CallbackDataPtr);  
  
uint8_t TRG_SetTrigger(??Kind, ??Time, ??Cback, ???CbDataPtr);  
void    TRG_IncTick(void);  
void    TRG_Init(void);
```

# Overview of Trigger.c

```
typedef struct TRG_TriggerDesc { ??? } TRG_TriggerDesc;
```

```
static TRG_TriggerDesc TRG_Triggers[TRG_NOF_TRIGGERS];
```

```
uint8_t      TRG_SetTrigger(??Kind, ??Time, ??Cback, ???CbDataPtr) { definition }
```

```
static bool  CheckCallbacks(void) { definition }
```

```
void         TRG_IncTick(void) { definition }
```

```
void         TRG_Init(void) { definition }
```

# Trigger.h

```
#define TRG_TICKS_MS  TMR_TICK_MS

typedef enum {
    TRG_LED_BLINK,
    TRG_BTNLED_OFF,
    TRG_BTNSND_OFF,
    TRG_KEYPRESS,
    TRG_NOF_TRIGGERS
} TRG_TriggerKind;

typedef void *TRG_CallbackDataPtr;

typedef void (*TRG_Callback)(TRG_CallbackDataPtr);

typedef uint16_t TRG_TriggerTime;

uint8_t TRG_SetTrigger(TRG_TriggerKind    trigger,
                      TRG_TriggerTime    ticks,
                      TRG_Callback        callback,
                      TRG_CallbackDataPtr data);

void TRG_IncTick(void);

void TRG_Init(void);
```

# Trigger.c – TRG\_SetTrigger( )

```
typedef struct TRG_TriggerDesc {  
    TRG_TriggerTime    ticks;  
    TRG_Callback        callback;  
    TRG_CallbackDataPtr data;  
} TRG_TriggerDesc;
```

```
static TRG_TriggerDesc TRG_Triggers[TRG_NOF_TRIGGERS];
```

```
uint8_t TRG_SetTrigger(TRG_TriggerKind trigger,  
                      TRG_TriggerTime ticks,  
                      TRG_Callback callback,  
                      TRG_CallbackDataPtr data) {  
    EnterCritical();  
  
    TRG_Triggers[trigger].ticks = ticks;  
    TRG_Triggers[trigger].callback = callback;  
    TRG_Triggers[trigger].data = data;  
  
    ExitCritical();  
  
    return ERR_OK;  
}
```

```
typedef enum {  
    TRG_LED_BLINK,  
    TRG_BTNLED_OFF,  
    TRG_BTNSND_OFF,  
    TRG_KEYPRESS,  
    TRG_NOF_TRIGGERS  
} TRG_TriggerKind;
```

```
typedef uint16_t TRG_TriggerTime;  
typedef void (*TRG_Callback)(TRG_CallbackDataPtr);  
typedef void * TRG_CallbackDataPtr;
```

# Trigger.c – CheckCallbacks()

```
static bool CheckCallbacks(void) {
    TRG_TriggerKind    i;
    TRG_Callback        callback;
    TRG_CallbackDataPtr data;
    bool                calledCallback = FALSE;

    for(i=(TRG_TriggerKind)0;i<TRG_NOF_TRIGGERS;i++) {
        EnterCritical();

        if (TRG_Triggers[i].ticks == 0 &&
            TRG_Triggers[i].callback != NULL ) {

            callback = TRG_Triggers[i].callback;
            data      = TRG_Triggers[i].data;

            TRG_Triggers[i].callback = NULL;

            ExitCritical();

            callback(data);
            calledCallback = TRUE;
        } else {
            ExitCritical();
        }
    }
    return calledCallback;
}
```

```
typedef enum {
    TRG_LED_BLINK,
    TRG_BTNLED_OFF,
    TRG_BTNSND_OFF,
    TRG_KEYPRESS,
    TRG_NOF_TRIGGERS
} TRG_TriggerKind;
```

```
typedef uint16_t TRG_TriggerTime;
typedef void (*TRG_Callback)(TRG_CallbackDataPtr);
typedef void * TRG_CallbackDataPtr;
```



# Trigger.c – IncTick( ), Init( )

```
void TRG_IncTick(void) {
    TRG_TriggerKind i;

    EnterCritical();

    for(i=(TRG_TriggerKind)0; i < TRG_NOF_TRIGGERS; i++) {
        if (TRG_Triggers[i].ticks!=0) {
            TRG_Triggers[i].ticks--;
        }
    }

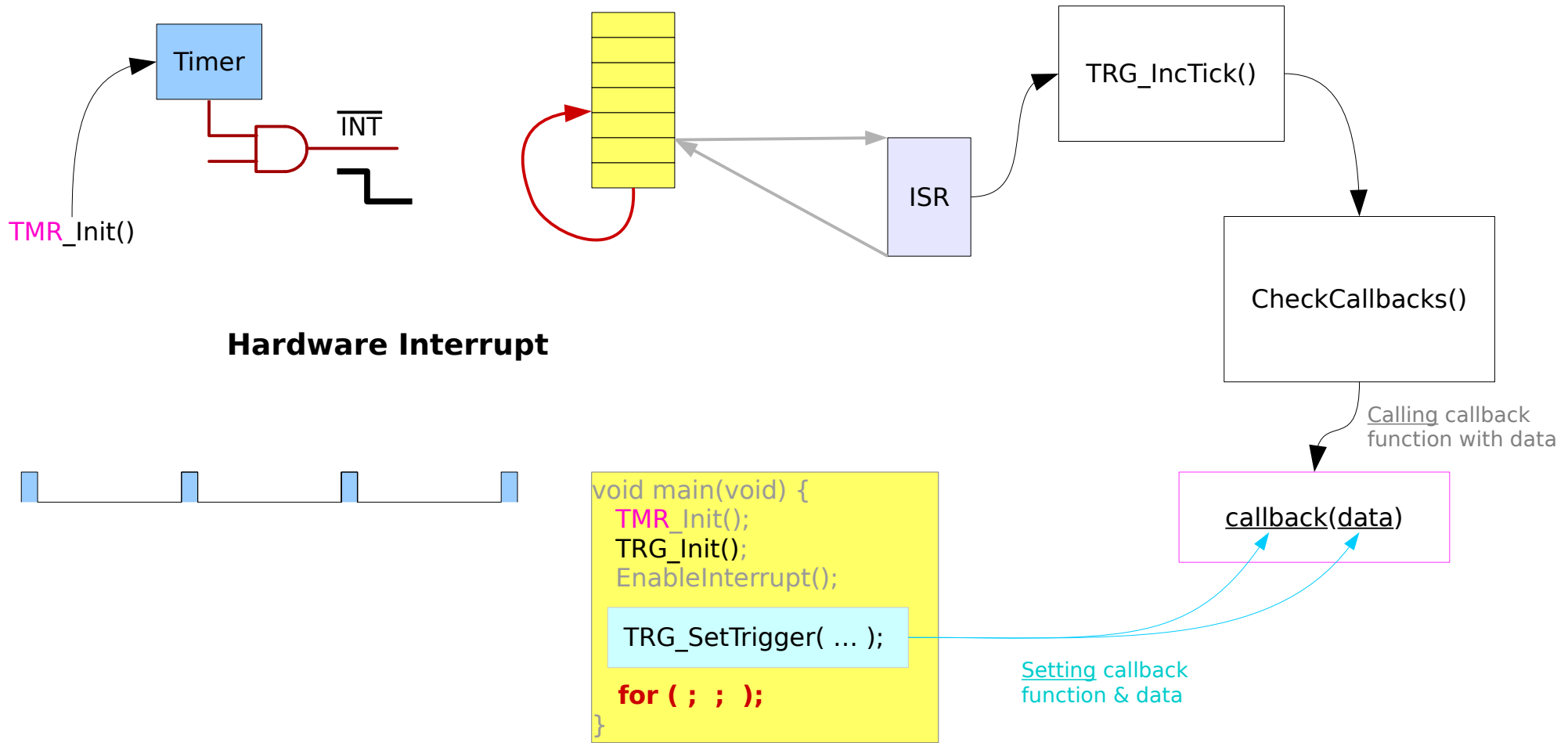
    ExitCritical();

    while(CheckCallbacks()) {}
}
```

```
void TRG_Init(void) {
    TRG_TriggerKind i;

    for(i=(TRG_TriggerKind)0;i<TRG_NOF_TRIGGERS;i++) {
        TRG_Triggers[i].ticks = 0;
        TRG_Triggers[i].callback = NULL;
        TRG_Triggers[i].data = NULL;
    }
}
```

# Timer & Interrupt



# Critical Section Access

```
TRG_Trigger[i].ticks = ticks;  
TRG_Trigger[i].callback = callback;  
TRG_Trigger[i].data = data;
```

```
(TRG_Trigger[i].ticks != 0)
```

```
TRG_Trigger[i].ticks--;
```

```
TRG_SetTrigger( ... );
```

WR

RD/WR

```
TRG_IncTick()
```

```
CheckCallbacks()
```

Shared Data

```
TRG_Trigger
```

	tick	callback	data
0			
1			
2			
3			

RD/WR

```
(TRG_Trigger[i].ticks == 0)  
(TRG_Trigger[i].callback != NULL)
```

```
callback = TRG_Trigger[i].callback;  
data = TRG_Trigger[i].data;  
TRG_Trigger[i].callback = NULL;
```

```
static TRG_TriggerDesc TRG_Triggers[TRG_NOF_TRIGGERS];
```

File Scope : static global variable

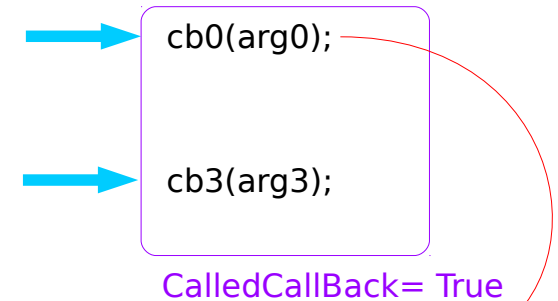
# Invoking Callbacks

TRG\_Trigger

	tick	callback	data
0	0	cb0()	arg0
1	11	cb1()	arg1
2	22	cb2()	arg2
3	0	cb3()	arg3



	tick	callback	data
	0	NULL	arg0
	11	cb1()	arg1
	22	cb2()	arg2
	0	NULL	arg3



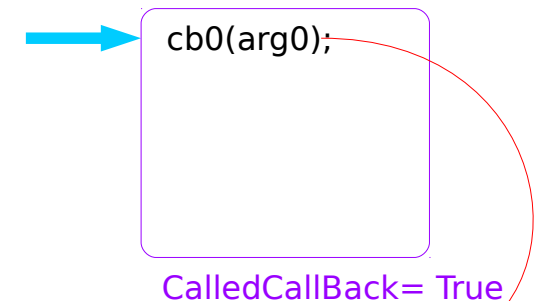
If the callback sets a trigger again for the relative time zero (at the current time), this newly scheduled trigger would be missed .  
**while ( CheckCallbacks() ) { }** would catch this case

`TRG_SetTrigger(0, 0, cb0, arg0);`

	tick	callback	data
	0	cb0()	arg0
	11	cb1()	arg1
	22	cb2()	arg2
	0	NULL	arg3



	tick	callback	data
	0	NULL	arg0
	11	cb1()	arg1
	22	cb2()	arg2
	0	NULL	arg3



`TRG_SetTrigger(0, 0, cb0, arg0);`

`static TRG_TriggerDesc TRG_Triggers[TRG_NOF_TRIGGERS];`

# Trigger.c

```
main() {  
    TRG_SetTrigger(TRG_LED, 0, LED_HeartBeat, NULL);  
}
```

```
TRG_TriggerKind :    TRG_LED,  
TRG_TriggerTime :    0,  
TRG_Callback :      LED_HeartBeat,  
TRG_CallbackDataPt : NULL
```

```
Main() {  
    foo();  
    foo();  
    foo();  
}
```

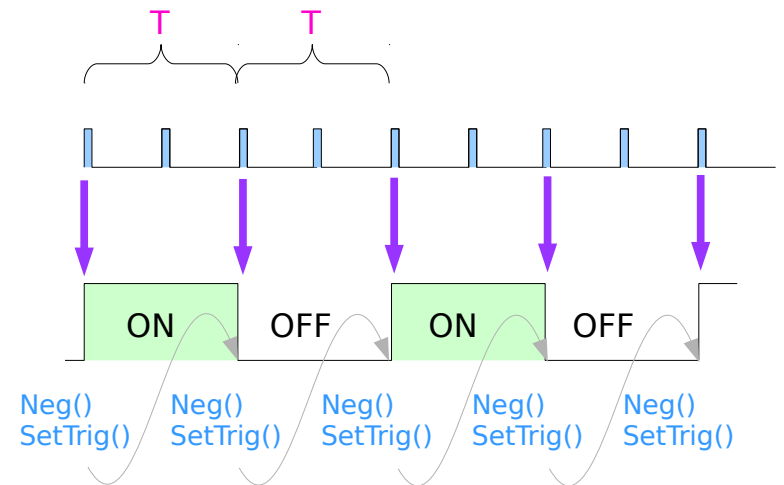
Call foo() multiple times  
with the same led location

```
foo() {  
    static uint8_t led = 1;  
    TRG_SetTrigger(TRG_LED, 0, LED_HeartBeat, &led);  
}
```

0 : immediate trigger

## Callback function

```
LED_HeartBeat() {  
    LED1_Neg();  
    TRG_SetTrigger(TRG_LED, T, LED_HeartBeat, NULL);  
}
```



## References

- [1] [http://wiki.osdev.org/ARM\\_RaspberryPi\\_Tutorial\\_C](http://wiki.osdev.org/ARM_RaspberryPi_Tutorial_C)
- [2] <http://blog.bobuhiro11.net/2014/01-13-baremetal.html>
- [3] <http://www.valvers.com/open-software/raspberry-pi/>
- [4] <https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/downloads.html>